Kombinatorische Optimierungsverfahren

zum

backbone Assignment von NMR Daten

an Proteinen

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Chemie der
Johann Wolfgang Goethe Universität
in
Frankfurt am Main

von

Volker Apelt

aus Bad Homburg v.d.H

Frankfurt am Main (2001)
(D F 1)

Die vorliegende Arbeit wurde am Institut für Organische Chemie der Johann Wolfgang Goethe Universität in Frankfurt am Main unter Anleitung von Prof. Dr. Griesinger in der Zeit vom Dezember 1993 bis November 1999 angefertigt.

Bei Herrn Professor Dr. Griesinger möchte ich mich für die anregenden Diskussionen und die bereitgestellten experimentellen Daten bedanken.

Herrn Peter Gröschke danke ich für die Diskussionen über die Methodik der Aminosäureerkennung und die lange Zusammenarbeit beim Aufbau und der Verwaltung der PCs und der Novell−Netzwerke des Arbeitskreises.

Christian Richter danke ich für die Zusammenarbeit bei der Verwaltung der NT Rechner und der Novell Netze und für die Diskussionen über Assignment Programme.

Bettina Elshorst und Dr. Holger Försterling möchte ich für die zur Verfügung gestellten Daten des C21W Komplexes danken.

Dr. Tanja Paraç danke ich für Überlassung der NMR Daten des Trigger Projektes und für die Unterstützung bei der Verifizierung des automatischen gegenüber dem manuellen Assignment.

Bei Dr. Markus Maurer möchte ich mich für die Hilfe bei der Interpretation der NMR Daten und die anregenden Diskussionen zur Lösung von Computerproblemen bedanken.

Den Co−Administratoren des UNIX Netzwerkes Dr. Peter Schmidt, Dr. Jens Quandt, Oliver Schedletzky, Jochen Junker, Dr. Thomas Geppert und Burkardt Luy danke ich für die anregenden Diskussionen über C Programmierung und die Bereitschaft bei Computerproblemen sofort einzuspringen.

Meinen Eltern möchte ich für die Unterstützung und ihre Geduld während der Entstehung der Arbeit danken.

Allen Mitgliedern des Arbeitskreises danke ich für die gute und freundliche Atmosphäre .

Inhaltsverzeichnis

Einführung

Für viele aktuelle wissenschaftliche Projekte ist die Kenntnis der dreidimensionalen Struktur von Proteinen eine Grundvoraussetzung. Für das Verständnis der Interaktion von chemischen Stoffen mit der Oberfläche eines Proteins ist zum Beispiel die Polarität und die Form der Oberfläche des Proteins von entscheidender Bedeutung, da sich Botenstoffe an der Oberfläche der Proteine anlagern oder Enzyme nur dann ihr katalytisch wirksames Zentrum ausbilden, wenn sie korrekt gefaltet sind. Daher reicht die Kenntnis der Primärstruktur nicht aus, um die Eigenschaften eines Proteins zu erklären.

Die Chemische und Pharmazeutische Industrie versucht für das *Rational Drug Design*, dem Entwurf neuer Medikamente am Computer, genaue dreidimensionale Modelle für die an einer Krankheit beteiligten Proteine zu erhalten, um dann Medikamente zu entwerfen, die sich besser an die aktiven Zentren des Proteins binden als das eigentliche Substrat. Andererseits wird so auch die Wirkungsweise schon bekannter Medikamente untersucht.

Entscheidende Voraussetzung dafür ist eine dreidimensionale Strukturaufklärung des Proteins, die die Struktur auf Bruchteile eines Atomdurchmessers genau bestimmen kann. Die einzigen Verfahren, die eine Strukturaufklärung bis auf wenige Picometer leisten können, sind NMR−Spektroskopie und Kristallographie.

Proteine, die kristallographisch untersucht werden sollen, müssen Kristalle bilden. Viele Proteine kristallisieren aber nur unter besonderen Bedingungen, zum Beispiel nur aus besonderen Lösungsmitteln. Im Gegensatz zur Kristallographie, kann mittels NMR die Struktur besonders schonend vermessen werden, da die Probe in einem nahezu natürlichen, wäßrigen Medium bleibt. Daher eignet sich prinzipiell jedes hinreichend lösliche Protein mit einer Masse unter 50 kD für eine NMR−Untersuchung. Außerdem muß in der NMR−Spektroskopie keine Verformung der Struktur durch Gitterkräfte befürchtet werden, wie sie in einem Kristall auftreten. Die vermessene Struktur entspricht daher in einer NMR Untersuchung eher der natürlichen Form, als in einer kristallographischen Untersuchung.

Voraussetzung für diese NMR Messungen ist eine isotopenmarkierte, heteronuklear vollständig gelabelte Proteinprobe, in der die N− und C− Atome im *backbone* und den Seitenketten des Proteins vollständig durch ihre Isotope ^{13}C und ^{15}N ersetzt wurden, da nur diese Isotope magnetisch attraktiv sind. Diese Proteine kann man aus Bakterienkulturen, die auf gelabeltem Substrat gewachsen sind, erhalten.

NMR Experimente messen durch die chemische Verschiebung σ Strukturinformationen eines Moleküls. In der chemischen Verschiebung ist der einzelne Atomtyp und die chemische Umgebung des Atoms kodiert. Durch Verkettung der Information mehrerer einzelner Atome zu einem Spinsystem mittels einer n−dimensionalen Pulssequenz erhält man Informationen über eine Gruppe benachbarter Atome, d.h. einen Aus−

schnitt der chemischen Struktur. Die Wahl der Pulssequenz bestimmt den Ausschnitt der Primärstruktur den man detektiert. Die Pulssequenz bildet quasi einen Strukturfilter. Ein n−dimensionales Spektrum vermißt dabei nicht nur einen einzelnen Bereich im Molekül, sondern alle topologisch gleichen Stellen.

Die Strukturaufklärung eine Proteins mit NMR Experimenten erfolgt in mehreren Schritten. Zuerst werden mehrere n−dimensionale heteronukleare NMR−Spektren aufgenommen und aufbereitet (*prozessiert*). Die Spektren werden so ausgewählt, daß sie verschiedene überlappende Teilstrukturen des Proteins als Spinsysteme messen. Typische Assignment−Experimente für Proteine messen die Verschiebungen der N, NH, CO, CA und CB Atome und decken dadurch Teilstrukturen im backbone des Proteins innerhalb einer oder zweier benachbarter Aminosäuren ab. Dabei werden nur Experimente aufgenommen, die Magnetisierung entlang der chemischen Bindung (J−Kopplungen) transferieren. Bax, Griesinger und andere [Grzesiek92c,Croasmun94] haben in den letzten Jahrzehnten verschiedene Systeme von Spektren entwickelt, die unterschiedliche Strategien für die Zuweisung implementieren. Für die vorliegende Arbeit wurde ein Experimentset ausgewählt, das für das automatisierte ASSIGNMENT verschiedene Vereinfachungen gegenüber den anderen Sets ermöglicht.

Nach der Messung werden, mittels einer Peak−Picking−Routine oder von Hand, die Koordinaten und das Volumen der Kreuzsignale (Peaks) bestimmt. Für schlechte, d.h. signalschwache oder stark überlagerte Spektren, müssen diese Listen von Hand korrigiert werden, da die Peak−Picking−Routine dann meist versagt.

Der nächsten Schritt ist das sogenannte ASSIGNMENT. In ihm werden die Frequenzen der Peaks den Teilstrukturen im Protein zugeordnet. Jede Teilstruktur entspricht einer Gruppe benachbarter Atome im Molekül. Der Ausschnitt, den ein Peak repräsentiert, wurde durch die Wahl der Pulssequenzen definiert.

Da die chemische Verschiebung eines Atoms hauptsächlich vom Isotop und der chemischen Umgebung in der Primärstruktur des Proteins bestimmt wird und und die Sekundär− und Tertiärstruktur geringeren Einfluß auf σ haben, sind sich die Verschiebungen der topologisch identischen Atome der Aminosäuren des gleichen Aminosäuretyps ähnlich. Daher kann, ohne weitere Informationen, der einzelne Peak nicht eindeutig einer bestimmten Stelle im Molekül zugeordnet werden, wenn eine Aminosäure mehrfach im Protein auftritt. In einem größeren Molekül ist diese Mehrdeutigkeit die Regel.

Da sich die Teilbereiche der verschiedenen Spektren aber überlappen, kann man die gleiche Frequenz in den Peaks der verschiedenen Spektren wiederfinden. Die Peaks der verschiedenen Experimente bilden dadurch ein Netz, daß das gesamte Rückrat und einen Teilbereich der Seitenketten abdeckt. Dieses Strukturnetz versucht man während des Assignments zu rekonstruieren. Das Problem, die einzelnen Peaks bestimmten Strukturen im Molekül zu zuordnen, ist das sogenannte ASSIGNMENT−Problem. Für große Proteine wird diese Information so komplex und mehrdeutig, daß sie von Hand nur mit großem Zeitaufwand aufgeklärt werden kann. Je nach Qualität der Spektren und der Größe der Proteine nimmt die Zuweisung bis zu mehreren Monaten in Anspruch.

Wenn das Assignment gelöst ist, wird mit einer zweiten Gruppe von NMR−Experimenten der räumliche Abstand zwischen verschiedenen Aminosäure gemessen. Dabei werden mittels NOE−Experimenten Peaks erzeugt, die die Frequenzen der beiden Endpunkte (Atome der Aminosäuren) tragen und in ihrem Volumen

die Distanzinformation enthalten. Im Gegensatz zu den Assignment−Experimenten, bei denen die Verknüpfungsinformation nur entlang der chemischen Bindung als J−Kopplung übertragen wird, wird hier die Magnetisierung in einem Schritt auch durch den Raum übertragen. Um die Distanzen bestimmten Atompaaren im Protein zuordnen zu können, braucht man hier die eindeutige Zuweisung der einzelnen Frequenzen und Peaks zu einzelnen Atomen in den Aminosäuren aus dem Assignment.

Diese Abstandszuordnungen sind die Grundlage für die *Distance Geometry*− und *Molecular Dynamics* −Rechnungen, die aus den Abstandsinformationen die wahrscheinlichste dreidimensionale Struktur bestimmen. Die Abstandsinformationen werden dabei als Nebenbedingungen in der Berechnung verwendet.

In den letzten Jahren wurden verschiedene Ansätze zu Lösung des N−ASS Problems veröffentlicht. Die Ansätze reichen von einer graphischen Benutzerführung bis zu Batchprogrammen, die eine Zuweisung semiautomatisch erzeugen.

Für die semiautomatische Zuweisung werden zuerst die Peaks in Gruppen zusammengefaßt. Die Zuordnung zu den Aminosäuren wurde in zwei grundlegend verschiedenen Ansätzen versucht.

Die Programme CONTRAST [Markley94] und AUTOASSIGN2 [Zimmerman95] versuchen die gesamte Zuweisung durch Verlängerung schon gefundener Sequenzen zu erreichen. Dabei müssen sie zwangsläufig alle Kombinationsmöglichkeiten ausprobieren, um die beste, d.h. vollständigste Zuweisung zu finden. Für große Proteine (>100 Aminosäuren), wird die Anzahl der zu testenden Zuweisungen so groß, daß sie nicht mehr in akzeptabler Zeit getestet werden können.

Dieses Zeitproblem umgehen Lukin *et al.* in [Lukin97] und PASTA von Leutner *et al.* [Leutner98] indem sie auf die Garantie der Vollständigkeit der gefunden Lösung verzichten und ein heuristisches Verfahren benutzen. Lukin *et al.* benutzen einen *simulated annealing* Algorithmus, während in PASTA *threshold accepting* als Verfahren verwendet wird. Außer in der Akzeptanzfunktion unterscheiden sich die Algorithmen auch in der Gruppierung der Peaks und den erlaubten Veränderungen, den sogenannten Zügen, die aus einer gefundene Lösung eine neue Lösung erzeugen. Beide Verfahren finden eine optimale Lösung, wenn man vorher die Peakgruppen so auswählt, daß pro Aminosäure genau eine Peakgruppe existiert und jeder Peak in nur einer Gruppe auftaucht. Die Verfahren können zwar auf die Zuweisung einer Aminosäure verzichten, nicht aber auf die einer Peakgruppe. Außerdem müssen alle Gruppen schon in der Startzuweisung vorhanden sein. Diese Programme können daher nur die Verteilung der Gruppen auf die Aminosäuren verändern und optimieren. Die Auswahl und Zusammensetzung der Gruppen können sie nicht optimieren. In mehrdeutigen Spektren mit mehrfacher Überlappung der Peaks und einem schlechten Signal/Rausch−Verhältnis muß diese Entscheidung daher schon vor der Optimierung der Zuweisung von Hand vorgenommen werden.

In Kapitel 1 wird eine Definition des NMR−Assignmentproblems (N−ASS) vorgestellt und die Theorie des Assignmentproblems mit Blick auf die Implementierung eines Optimierungsverfahrens diskutiert. Dazu werden in Abschnitt 1.2 unterschiedliche Implementierungen aus der Literatur untersucht, um eine geeignete Basisstruktur für das kombinatorische Optimierungsverfahren zu finden. Dabei wird sich herausstellen, daß nur eine Gruppierung der Peaks in Protospinsysteme ein erfolgreiches Optimierungsverfahren verspricht.

Danach wird die Größe und Struktur des Lösungsraumes der Optimierungsaufgabe mit dieser Struktur untersucht und anhand von Praxisdaten aus verschiedenen Proteinen und Assignmentproblemen berechnet.

Im Anschluß werden die theoretischen Grundlagen verschiedener kombinatorischer Optimierungsverfahren vorgestellt. Dabei wird ein Blick auf ein verwandtes Problem, das *traveling salesman* Problem (TSP), und die daran erprobten Verfahren geworfen. Dabei stellt es sich heraus, daß das N−ASS aus verschiedenen Gründen schwieriger als das TSP ist. Daher können die Erfahrungen aus der TSP−Literatur nur als Anregung für die Implementierung des N−ASS benutzt werden. Deshalb wird in dieser Arbeit ein neuer Algorithmus vorgestellt, der die Vorzüge von drei verschiedenen erprobten Algorithmen vereint.

In Kapitel 2 wird die Implementierung der ausgewählten Optimierungsverfahren vorgestellt. Um die Optimierung implementieren zu können, müssen die Eingabedaten vorbereitet werden. Dazu werden verschiedene Algorithmen vorgestellt, die die einzelnen Schritte ab dem *peak picking* bis zum Assignment unterstützen und automatisieren. Die Algorithmen automatisieren die Suche nach den Spinsystemen und können sie den einzelnen Aminosäuren zuweisen. Dabei wird die Auswahl der Peakgruppen erst während der Optimierung der Zuweisung getroffen. Außerdem wird eine Bewertungsfunktion für das Assignment und die Spinsysteme vorgestellt. Danach wird die Implementierung des RANDOM und des GENETIC Algorithmus erklärt.

Kapitel 3 überprüft die Korrektheit der Implementierung und den Einfluß von Fehlern in den Ausgangsdaten auf das Ergebnis der Optimierung. Außerdem wird eine optimale Einstellung der frei wählbaren Parameter des RANDOM und GENETIC Algorithmus gesucht. Die optimale Parametrisierung muß dabei zugleich robust und schnell sein, um die Lösung auch unter Praxisbedingungen zu finden. Neben den Parametern werden auch verschiedene Varianten des Crossover−Operators für GENETIC. Kapitel 3.6.2 vergleicht das Ergebnis dieser Untersuchung mit der theoretischen Vorhersage aus Kapitel 2.10.3 und erklärt die Unterschiede.

Für GENETIC wird außerdem untersucht, ob man die Parallelität des genetischen Algorithmus nicht zu einer Verkürzung der Rechenzeit oder zu einer Steigerung der Robustheit mit schlechten Ausgangsdaten nutzen kann. Dazu wird der Parameterraum in mehrere Teilbereiche aufgeteilt, in denen der Algorithmus durch die Parametrisierung dem einzelnen Individuum unterschiedlich viel CPU−Zeit zuteilt. Kapitel 3.6.8.1.3 stellt die Ergebnisse des Vergleichs der ratenbegrenzten Konfiguration mit anderen Konfigurationen vor und bewertet den Nutzen der einzelnen Konfigurationsvarianten.

Kapitel 4 wird den Algorithmus mit dem Programm (PASTA) von Leutner *et. al*. anhand eines theoretischen Datensatzes vergleichen.

Im vorletzten Kapitel 5 wird GENETIC an einem realen Problem getestet. Dabei wird die Berechnung in mehreren Stufen verfeinert und mit der unabhängig erstellten, manuellen Lösung verglichen.

Kapitel 1 Theorie

Im folgenden Abschnitt werden die theoretischen Grundlagen der Optimierungsaufgabe diskutiert.

1.1 Das ASSIGNMENT Problem

Die folgende Definition setzt voraus, daß für eine Proteinlösung ein Satz von 3D NMR−Experimenten aufgenommen wurde. Die resultierenden Spektren wurden außerdem bereits fourier−transformiert und prozessiert. Die 3D−Koordinaten der intensivsten, lokalen 3D−Maxima und Minima im Spektrum sind die Peaks.

Definition: **ASSIGNMENT Problem**: **N−ASS**. Suche die wahrscheinlichste Zuweisung zwischen den Atomen im Protein und den Frequenzen der Peaks im Spektrum, entsprechend den Transferwegen im NMR−Spektrum.

Zwischen dem *peak picking* und der Zuweisung werden die Peaks aus den verschiedenen Spektren meist, in einem vorbereitenden Schritt, zu Protospinsystemen zusammengefaßt. Die Regeln, nach denen die Protospinsysteme erstellt werden, hängen dabei von den NMR−Experimenten ab.

Definition: **Protospinsystem**: Ein Protospinsystem ist eine Gruppe von Peaks, die einen experimentabhängigen Satz von Bedingungen erfüllt. Durch die Nebenbedingungen wird den Peaks gleichzeitig eine Funktion im Protospinsystem zugewiesen. Typische Nebenbedingungen fassen die Peaks einer Aminosäure, d.h. die mit der gleichen HN Frequenz zusammen.

In der Praxis wird die manuelle Zuweisung iterativ ausgeführt. Sowohl das *peakpicking,* als auch die Protospinsystemsuche, sind zu keinem Zeitpunkt wirklich abgeschlossen, sondern werden von zu einem späteren Zeitpunkt gefundenen Fakten wieder in Frage gestellt und erneut bewertet.

```
Algorithmus: Manuelles ASSIGNMENT
parameter:
   P        Protein
   Exp      Spektren
   S        Zuweisungstabelle
   S_best   Zuweisungstabelle
return:
   S        Zuweisungstabelle
variablen:
begin
   loop until satisfied
      suche nicht gepickte Peaks im Spektrum.
      fasse Peaks zu Protospinsystemen zusammen,
         dabei können auch bestehende und zugewiesene Protospinsysteme zerlegt
         oder schon gepickte Peaks gelöscht werden.
      bestimme den möglichen AS-Typ und die möglichen Spinsystemnachbarn.
      weise die Protospinsysteme einem Bereich im Protein zu.
      ordne die Qualität der aktuellen Zuweisung relativ zu den früher gefundenen Zuweisungen
ein.
      S_best = S  wenn S besser als S_best ist.
   end loop
end
```

1. Die harten theoretischen Nebenbedingungen des Experiments und des Proteins *müssen* in der Zuweisung erfüllt sein.

   d.h. Die Magnetisierungstransferpfade müssen eingehalten sein, Proline erzeugen keine ^{1}HN –, Glycine keine CB−Frequenz, relative Vorzeichen der Peakvolumen usw.
2. Die weichen theoretischen Nebenbedingungen des Experiments und des Proteins *sollten* erfüllt sein. Ein Beispiel sind die Volumenverhältnisse zwischen Peaks im gleichen Experiment.
3. Die Zuweisung soll möglichst viele Peaks und Frequenzen erklären.
4. Peaks sollen möglichst nur einem Atom zugeordnet werden.
5. Die Qualität der Peaks soll möglichst hoch sein.
6. Die Frequenz eines Atoms soll in den verschiedenen Peaks möglichst gut übereinstimmen.
7. Die Zuweisung soll möglichst zusammenhängend sein. Eine Zuweisung, die einen durchgehenden Abschnitt des Proteins zuweist, ist besser als eine, die gleich viele Aminosäuren erklärt, aber aus zwei Teilen besteht.
8. Die CA− und CB−Frequenzen der Spinsysteme sollen mit der bekannten Frequenzverteilung der Aminosäure, der sie zugewiesen wurden, übereinstimmen.

Tabelle 1−1 Die Bewertungskriterien für ein manuelles Assignment.

Gesucht ist die Zuweisung, die gleichzeitig die meisten Peaks zuweist, am wenigsten Peaks doppelt nutzt und die bekannten Frequenzmuster der Aminosäuretypen am besten beachtet. Dies führt zu einer 1:n Zuweisung, da die gleiche Frequenz von mehreren Atomen stammen kann, jedes Atom aber nur eine Frequenz besitzt.

Im ersten Schritt werden dazu Peaks mit gleichen Frequenzen in Gruppen, den sogenannten Protospinsystemen, zusammengefaßt. Jedes Protospinsystem muß noch weitere Nebenbedingungen einhalten, die sich aus den theoretischen Anforderungen der verwendeten NMR−Experimente ergeben. Beim N−ASS gehören die Peaks eines Protospinsystems zur gleichen Aminosäure oder zu ihrem i−1 Nachbarn in der Proteinsequenz.

Es gibt keine absolute, theoretisch herleitbare Qualitätsfunktion für die Bewertung während des manuellen Assignments. Eine Zuweisung ist dann wahrscheinlich, wenn sie die Kriterien in Tabelle 1−1 erfüllt. Die relative Bewertungsreihenfolge der Kriterien ist nicht festgeschrieben, sondern hängt vom Erfahrungshintergrund der Person ab, die die Zuweisung vornimmt. Auch die Bewertung einer Verletzung eines Kriteriums ist nicht festgeschrieben. Nur das erste Kriterium darf nicht verletzt werden.

Ein Grund für das Fehlen einer theoretischen Bewertungsfunktion ist die mangelnde Unterstützung für eine solche Funktion in der zugänglichen Software wie FELIX (MSI/Biosym) oder Aurelia (Neidig *et. al.*, [Neidig95]), die weder ein direkte Dokumentation noch eine kontinuierliche Bewertung der einzelnen Kriterien unterstützt. Eine kontinuierliche Bewertung des Assignments ist manuell nicht durchführbar, da tausende von Zuweisungen bewertet werden müßten.

1.2 Automatisches Assignment

Um die Definition des manuellen Assignments auf Seite 5 in einen Algorithmus umzusetzen, muß die Problemstellung exakter definiert werden. Im folgenden werden daher der Begriff der Zuweisung, der Qualitätsfunktion und des Automatischen Assignments definiert. Vorgegeben sind ein Protein *Pr* und ein Datensatz mit Peaks aus NMR−Experimenten *Ex*.

Definition: *NMR−Assignment, Zuweisung:* Eine Zuweisung $\boldsymbol{S_i}$ ist eine Zuordungsvorschrift, die Peaks in den Spektren des experimentellen Datensatzes *Ex* entweder keinem oder einem bis mehreren Atomen im Protein *Pr* zuordnet. Die Zuordnung muß die harten physikalischen Nebenbedingungen der NMR−Experimente und der Probe erfüllen.

Die Bewertungsfunktion *Q* muß $\boldsymbol{S_i}$ so bewerten, daß die Ordnungsreihenfolge von einer Bewertung nach den Kriterien in Tabelle 1−1 nicht unterschieden werden kann. Die Bewertungsfunktion wird auch als Qualitätsfunktion bezeichnet.

Definition: *Bewertungsfunktion* $Q(\boldsymbol{S_i},Pr,Ex)$: Q ist eine Funktion, die eine Zuweisung $\boldsymbol{S_i}$, unter Berücksichtigung des Proteins *Pr* und des Experimentsets *Ex* auf einen Qualitätswert im Bereich 0 bis $+\infty$ abbildet. Bei gleichem Datensatz und Protein ist Q eine Ordnungsfunktion. Für zwei Zuweisungen $\boldsymbol{S_1}$ und $\boldsymbol{S_2}$ gilt daher für eine Bewertung nach den Kriterien in Tabelle 1−1:

$Q(\boldsymbol{S_1}) = Q(\boldsymbol{S_2})$, wenn $\boldsymbol{S_1}$ *identisch* mit $\boldsymbol{S_2}$ ist.

$Q(\boldsymbol{S_1}) > Q(\boldsymbol{S_2})$, wenn $\boldsymbol{S_1}$ nicht identisch mit $\boldsymbol{S_2}$ ist, aber $\boldsymbol{S_1}$ *besser* als $\boldsymbol{S_2}$ *bewertet* wird.

$Q(\boldsymbol{S_1}) = Q(\boldsymbol{S_2})$, wenn $\boldsymbol{S_1}$ nicht identisch mit $\boldsymbol{S_2}$ ist, aber $\boldsymbol{S_1}$ *gleich* $\boldsymbol{S_2}$ *bewertet* wird.

D.h. der Zahlenwert von *Q* nimmt mit steigender Qualität zu.

Mit diesen Definitionen lautet das NMR Assignment Problem dann:

Definition: *NMR Assignment Problem* (abgekürzt **N−ASS**): Suche die Zuweisung(en) $\boldsymbol{S_i}$ bei der die Qualitätsfunktion $Q(\boldsymbol{S_i},Pr,Exp)$ maximal ist.

Die vorliegende Arbeit untersucht einen Algorithmus, der das N−ASS Problem effizient für theoretische und reale Spektren und Proteine löst.

1.2.1 Das Experimentset

Das Experimentset besteht aus einer Gruppe von NMR Experimenten, die für eine Zuweisung an einer Proteinlösung aufgenommen werden. Für die Messungen wurden die verbesserten Pulssequenzen aus der Dissertation von Markus Maurer [Maurer2000] verwendet. Für die vorliegende Arbeit wurden die Experimente D−HNCO [Maurer2000, Kay90, Grzesiek92b], ein 3D−HNCA [Maurer2000, Archer91], ein 3D−HN(CO)CACB [Maurer2000, Grzesiek92a] und ein 3D−HNCACB [Maurer2000, Grzesiek92a] ausgewählt, da sich bei ihnen die Peaks eines Protospinsystems über die ^{13}N− und ^{1}HN− Frequenzen besonders leicht zusammenfassen lassen, da diese Frequenzen in allen Peaks vorhanden sind.

Dieses Experimentset wird im folgenden als *Bax Experimentset* bezeichnet, da die ursprüngliche Form dieser Experimente in der Gruppe um Ad Bax [Kay90, Archer91, Grzesiek92a, Grzesiek92b] entwickelt wurde. Die Details des Experimentsets werden in Kapitel 2.3 besprochen.

1.2.2 Die Zuweisung

Eine manuelle Zuweisung wird als eine Tabelle dargestellt, die eine Zeile für jede Aminosäure im Protein und eine Spalte für jeden Atomtyp der Aminosäure enthält, das von den NMR−Experimenten gemessen werden kann. Die Zellen der Tabelle enthalten die Frequenz der Atome in [ppm] d.h. die Frequenz relativ zu einer Referenzfrequenz oder sie sind leer. Eine Zelle kann jede Frequenz aufnehmen, die im Spektrum auf−tritt und mit den Bedingungen des Experimentsets übereinstimmt. Die Anzahl der Zeilen ist gleich der Anzahl der Aminosäuren im Protein.

Diese Darstellung entspricht nicht den tatsächlich gemessenen Entitäten. Die Frequenzen der Atome werden im Peakpicking nicht unabhängig von einander bestimmt, sondern treten als mindestens n+1−dimensionale Tupel von Meßwerten auf, da die Peaks jeweils das n−dimensionale Maximum in einem Spektrum sind. Die ersten n Dimensionen sind die Koordinaten, d.h. Frequenzen der Peaks. Die n+1−te Dimension ist entweder die Höhe des Peakzentrums oder das Volumen des Peaks. Weitere Dimensionen können noch hinzukommen, wenn weitere Eigenschaften bestimmt werden.

Eine Implementierung der N−ASS Zuweisung muß daher die Peaks als kleinste Einheit berücksichtigen. Eine automatische Implementierung wird daher eine Tabelle aus einer Zeile pro Aminosäure und einer Spalte pro möglichem Spinsystem erzeugen. Die Zellen der Tabelle können entweder einen Peak enthalten oder leer sein. Diese Tabelle kann jederzeit in die Frequenztabelle der manuellen Zuweisung umgerechnet werden, indem aus den Frequenzen der Peaks, die vom gleichen Atom herrühren, der Mittelwert gebildet wird.

1.2.3 Granularität der Implementierung.

Das N−ASS erfordert ein kombinatorisches Optimierungsverfahren, da die zu optimierende Eigenschaft durch eine Auswahl von Elementen aus einer abzählbaren Menge **P** einzelner unterscheidbarer Ele−mente zusammengesetzt werden muß. Um das N−ASS als kombinatorisches Optimierungsverfahren zu implementieren, muß man entscheiden, welche Daten im Algorithmus kombiniert und vertauscht werden. Die Wahl der Daten hat einen entscheidenden Einfluß auf die Qualität der Lösungen, die Geschwindigkeit der Berechnung und die Robustheit des Algorithmus.

Als kleinste kombinierbare Datenstruktur bieten sich die Peaks an, da sie die im *peak picking* gewonne−nen, Basisdaten sind. Einzelne Frequenzen können nicht vertauscht werden, da sie nur im Zusammenhang mit den restlichen Frequenzen des n−dimensionalen Peaks einen Sinn ergeben. Man kann die Peaks aber auch zu größeren Gruppen zusammenfassen und diese Gruppen als kombinierbare Einheiten im Optimie−rungsverfahren einsetzen. Dabei erhöht man den Organisationsgrad der kombinierten Datenstruktur, indem man die Einheiten der vorhergehenden Stufe zu einer neuen Datenstruktur zusammenfaßt.

Das N−ASS kann so auf verschiedenen Komplexitätsstufen als kombinatorisches Optimierungsverfah−ren implementiert werden. Die verschiedenen Stufen unterscheiden sich durch den Organisationsgrad der Peaks. Von Stufe zu Stufe nimmt der Organisationsgrad zu. Dabei entstehen die Elemente der Stufe *n* durch Gruppierung der Elemente der Stufe *n−1*. Den höchsten möglichen Organisationsgrad hat die gesamte Sequenz als eine Einheit.

Die nächst komplexeren Einheiten, nach den Peaks, sind einzelne Protospinsysteme, die, mittels einer Mustersuche, aus den Peaks gebildet werden. Auf der nächst höheren Stufe kann man mehrere Spinsysteme zu einer Teilsequenz zusammenfassen und diese Teile permutieren.

Welche Stufe angemessen ist, hängt von der Qualität der Spektren und dem untersuchten Protein ab.

Stufen des Organisationsgrades	permutierte Struktur	Substrukturen
1	Peaks oder Links zwischen Peaks	Frequenzen, Volumen
2	Protospinsysteme	Peaks
3	verkettete Protospinsysteme	Protospinsysteme
4	die gesamte Sequenz	verkettete Protospinsysteme

Tabelle 1−2 Organisationsgrad und Strukturname

- Vorteile der ersten Stufe.

  Permutieren der Peaks entspricht der natürlichen Granularität der Basisdaten. Die Implementierung ist eine Tabelle, die der Ergebnistabelle des manuellen Assignments entspricht. Die Elemente sind die gemessenen Entitäten. Die Aggregation der Elemente zu den nächst höheren Komplexitätsstufen übernimmt der Optimierungsalgorithmus durch seine Bewertungsfunktion und die Mutationsaktionen.

  Die erste Komplexitätsstufe kann aber nur schwer mit Filtern auf der Protospinsystemebene behandelt werden. Diese Filter müßten nach jeder Veränderung d.h. jedem Zyklus neu angewandt werden. Die Protospinsysteme müssen selbst dann neu bewertet werden, wenn sie schon früher einmal bewertet wurden. Protospinsysteme, die zur gleichen Konkurrenzgruppe gehören, können nicht gemeinsam betrachtet (bewertet) werden, da sie nie gleichzeitig vorliegen. Daher können konkurrierende Peakgruppen nicht erkannt und getrennt werden! Dabei tritt aber auch kein Verlust suboptimaler Protospinsysteme durch zu enge Filter ein.

  Diese Implementierung wurde von Lukin *et al* [Lukin97] versucht. Sie fanden, daß diese Implementierung fast nie zum globalen Optimum konvergiert. Lukin *et al.* schließen daraus, daß durch zufällig verknüpfte, entartete Peaks zusätzliche lokale Optima entstehen, die durch tiefe, schwer zu überwindende Barrieren getrennt sind. Dadurch bleibt der Optimierungsalgorithmus schon in Bereichen stecken, die manuell leicht gelöst werden können.

  Diese Stufe ist daher nur dann sinnvoll, wenn keine Filter notwendig sind. Die gemessenen Spektren dürfen daher nur wenig von den theoretischen abweichen.

- Vorteile der zweiten Stufe:

  In der zweiten Stufe werden die Peaks innerhalb der gleichen oder benachbarten Aminosäure (d.h. die in einer Zeile der Assignmenttabelle) zu einem Protospinsystem zusammengefaßt. Auf diese Protospinsysteme kann man dann weitere Bearbeitungsschritte anwenden, zum Beispiel Filter, die alle Protospinsysteme einer Aminosäure gemeinsam betrachten und eine optimierte Vorauswahl treffen. Alle Nachteile der ersten Stufe werden hier zu Vorteilen. Protospinsysteme werden nur einmal erzeugt und bewertet, und können mit beliebig vielen Filtern nachbearbeitet werden.

Nachteil ist der Verlust pseudo−suboptimaler Protospinsysteme durch zu enge Filter. Daher kann es vorkommen, daß die globale theoretische Lösung nicht mehr im Lösungsraum vorhanden ist, da eines der lokal suboptimalen Protospinsysteme durch die Filter entfernt wurde, das aber als einziges in die globale Lösung paßt. Ob solche Datensätze im manuellen Assignment gelöst werden, ist fraglich, denn hier steht man vor dem gleichen Problem. Manuell versucht man dieses Problem durch ein iteratives Vorgehen zu lösen. Im automatischen Assignment kann dies durch iteratives Verbessern der Filter erreicht werden.

Diese Stufe wird von Leutner et. al. in PASTA [Leutner98] und von Lukin *et al* [Lukin97] benutzt. Dabei unterscheiden sich die Protospinsysteme in ihrem Umfang und ihrer Bedeutung. PASTA benutzt Protospinsysteme, die jeweils die Peaks mit den Frequenzen der *i.*ten Aminosäure und des *i−1*−Nachbarn erfassen, während Lukin dieses Protospinsystem in zwei Protospinsysteme aufteilt, die jeweils nur die Frequenzen der i− oder der i−1 −Aminosäure enthalten.

- Vorteile der dritten Stufe:

  Diese Implementierung wurde in [Markley94, Fesik94, Ikura90a, Zimmermann95] untersucht. Diese Implementierung ist nur für qualitativ hochwertige Spektren mit wenigen überlappenden Protospinsystemen durchführbar, da die Anforderungen an den Speicherplatz exponentiell steigt, wenn man auch suboptimale Protospinsystemketten (Sequenzen) berücksichtigen (speichern) will. Außerdem müssen vorher alle Filter, die für die zweite Stufe verwendet werden, angewandt werden, um ein möglichst kleines Set von zu verknüpfenden Protospinsystemen zu erhalten. Dabei tritt wieder der oben genannte Verlust suboptimaler Protospinsysteme durch zu enge Filter ein. Danach müssen zusätzlich die entsprechenden Filter auf die erzeugten Sequenzen angewandt werden, dabei tritt erneut ein weiterer Verlust der Interpretationsmöglichkeiten auf. Im Endeffekt ist diese Implementierung nur anwendbar, wenn die globale theoretische Lösung an jeder Aminosäure gleichzeitig auch das lokale Maximum ist. Zumindest für die AS−Typenerkennung ist diese Bedingung in keinem der untersuchten Spektren erfüllt.

In dieser Arbeit werden daher Protospinsysteme als Elemente der Optimierung verwendet. Diese Implementierung ist mit der geforderten Peaktabelle vereinbar.

1.2.4 Zuweisung und Lösungsraum

Jede mögliche Zuweisung ist eine Lösung $\boldsymbol{S_i}$ im Lösungsraum$\boldsymbol{S}$ des N−ASS Problems. Die $\boldsymbol{S_i}$ werden im folgenden Text auch Zustände oder Individuen genannt. Der Lösungsraum ist in der Implementierung auch der Zustandsraum.

Eine Lösung $\boldsymbol{S_i}$ des N−ASS besteht aus einer Permutation der Protospinsysteme über die Positionen der Aminosäuren. Eine Implementierung eines Zustandes ist eine Vektor, der für jede Aminosäure im Protein eine Position enthält, die entweder leer ist oder mit einem Protospinsystem belegt sein kann. Dabei ist, ohne Nebenbedingungen, jede Kombination aus Protospinsystemen und leeren Stellen erlaubt.

Das N−ASS erfordert Nebenbedingungen, die die Kombination von benachbarten Protospinsystemen aufgrund der Frequenzen der Peaks beschränken und dabei verbieten, daß ein Peak mehrfach im gleichen Individuum verwendet wird, ohne eine entsprechende Entartung zu besitzen. Diese Beschränkung "vererben" sie an die Protospinsysteme. Jedes Protospinsystem darf nur entsprechend der Entartung des Peaks

mit der niedrigsten Entartung im Vektor auftauchen. D.h. nur Protospinsysteme, in denen alle Peaks mindestens den Entartungsgrad 2 besitzen, können zweifach in das Protein aufgenommen werden.

Jede Lösung muß noch verschiedene zusätzliche Nebenbedingungen erfüllen, die durch das berechnete Problem definiert werden. Dabei werden einzelne Lösungen, die ohne Nebenbedingungen zulässig sind, im Optimierungsproblem mit Nebenbedingungen zu illegalen Lösungen. Alle Lösungen, die die Nebenbedingungen erfüllen, bilden den Raum der *erfüllbaren Lösungen* $\boldsymbol{S^e}$. $\boldsymbol{S^e}$ ist eine Untermenge des allgemeinen Lösungsraumes ohne Nebenbedingungen $\boldsymbol{S}$.

$$\boldsymbol{S}^e \subseteq \boldsymbol{S} \qquad (1\text{−}1)$$

$\boldsymbol{S^n}$ ist der Raum der *nicht−erfüllbaren Lösungen*. Es gilt:

$$\boldsymbol{S} = \boldsymbol{S}^e \cup \boldsymbol{S}^n \qquad (1\text{−}2)$$

$$\emptyset = \boldsymbol{S}^e \cap \boldsymbol{S}^n \qquad (1\text{−}3)$$

Der Lösungsraum ohne Nebenbedingungen ist daher mit Inseln aus verbotenen Zuständen aus $\boldsymbol{S^n}$ durchsetzt.

1.2.5 Die Größe des Lösungsraumes der Optimierungsaufgabe

Alle Protospinsysteme, die aus den Peaklisten der Experimente erzeugt werden können, bilden den Protospinsystempool. Jede Kombination von Protospinsystemen bildet eine mögliche Lösung $\boldsymbol{S_i}$ im Lösungsraum $\boldsymbol{S}$ des Optimierungsproblems. Die Aminosäuren entsprechen den belegbaren Positionen, die Protospinsysteme den Elementen. Eine Lösung kann mehrere leere Stellen enthalten. Da doppelt zugewiesene Protospinsysteme nur selten auftreten, wird für die folgende Größenberechnung vereinfacht angenommen, daß jedes Protospinsystem nur einmal vorkommen kann. Wenn die Protospinsysteme entartet sind ,d.h. mehrfach auftreten können, wird dadurch der Lösungsraum größer.

Der Lösungsraum hängt daher von der Zusammensetzung des Protospinsystempools ab. Unter idealen Bedingungen, wenn die gemessenen Frequenzen der Peaks exakt mit den theoretischen übereinstimmen, die Frequenzen des Ca und Cb Atoms innerhalb des Erwartungsbereichs ihres Aminosäuretyps liegen, alle Peaks vorhanden sind und keine falschen Peaks (noise−peaks) existieren, entstehen nur *perfekte*, d.h. theoretische Protospinsysteme. Der perfekte Protospinsystempool enthält für jede Aminosäure des Proteins maximal ein Protospinsystem. Unter realen Meßbedingungen entstehen alle oben genannten Defekte. Die Protospinsysteme müssen daher mit einer Toleranz gegenüber fehlerhaften Protospinsystemen gesucht werden. Daher wird pro Aminosäure meist mehr als ein Protospinsystem als Möglichkeit in den Pool aufgenommen. Der Pool enthält dann auch Fehldeutungen.

E	Element, das einer Position in der Permutation zugeordnet wird. (Protospinsystem)
p	Anzahl der belegbaren Positionen in einer Permutation. (zuweisbare Aminosäuren im Protein)
k	Anzahl der verteilbaren Elemente. (Protospinsysteme)
f	Anzahl der *leeren* Positionen in einer Lösung (0<=f<=p)
b	Anzahl der *belegten* Positionen in einer Lösung (0<=b<=p *und* b+f==p)
$\boldsymbol{S_0}$	Der leere Zustand.
$\boldsymbol{S_{b,p}}$	eine einzelne Lösung mit b Elementen und p Positionen. ein Zustand im Zustandsraum.

$T^{op}_{\boldsymbol{S}_1,\boldsymbol{S}_2}$ Operation, die $\boldsymbol{S}_1$ in $\boldsymbol{S}_2$ umwandelt. $\boldsymbol{S}_1 \in M_i \wedge \boldsymbol{S}_2 \in M_j$

M_b Menge aller $\boldsymbol{S}_{b,p}$ mit *b* belegten Positionen ohne Wiederholung eines Elements.

$\boldsymbol{S}^{\subset}_b$ Menge aller $\boldsymbol{S}_{b,p}$ mit *b* belegten Positionen, die die gleichen Elemente enthalten.

$\boldsymbol{N}$ Nachbarschaft

a(X) Anzahl der Elemente in der Menge *X*

Die Anzahl der möglichen Zustände im Lösungsraum hängt von der Anzahl der belegbaren Positionen, den möglichen Protospinsystemen, die einer Position zugeordnet werden können, und den Nebenbedingungen für die Kombination von Zuordnungen ab. Die Permutationen mit gleichem Belegungsgrad *b* bilden die Menge M_b.

Ohne Nebenbedingungen für die Anordnung der Elemente gilt für die Anzahl der möglichen Kombinationen $a_k(M_p)$ für eine vollständig belegte (f=0, b=p) Permutation:

$$a_k(M_p) = \frac{k!}{(k-p)!} \qquad \textit{mit } k > p \textit{ und } f=0 \textit{ (mit Fehldeutungen)} \qquad (1-4)$$

$$a_k(M_p) = k! \qquad \textit{mit } k == p \textit{ und } f=0 \textit{ (nur die perfekten Protospinsysteme)} \qquad (1-5)$$

Eine Permutation von *b* aus *k* Elementen auf *p* Positionen kann man in zwei Schritte, in die Wahl der belegten Positionen und die Wahl der zugewiesenen Elemente, zerlegen. Die Anzahl der möglichen Zustände $a(M_b)$ ist dann das Produkt der beiden Mengen.

Die Anzahl der Elementvariationen, d.h. *unter Beachtung* der Reihenfolge der Elemente, ist

$$a_E(M_b) = \frac{k!}{(k-b)!b!} \qquad (1-6)$$

Die Anzahl der Positionskombinationen, d.h. *ohne* Beachtung der Reihenfolge der Positionen, ist

$$a_{pos}(M_b) = \frac{p!}{(p-b)!} \qquad (1-7)$$

Für eine Permutation mit *b* belegten Positionen und *k* möglichen Elementen ist daher die Anzahl der möglichen Zustände

$$a(M_b) = a_E(M_b) \cdot a_{pos}(M_b) = \frac{p!}{(p-b)!} \cdot \frac{k!}{(k-b)!b!} \qquad (1-8)$$

Die Gesamtzahl aller Permutationen ohne Nebenbedingungen für *b=0* bis *b=p* ist daher

$$a(M_{b_{a,p}}) = \sum_{b=0}^{p} \frac{p!}{(p-b)!} \cdot \frac{k!}{(k-b)!b!} \qquad (1-9)$$

Protein	Länge	Belegbare Aminosäuren *p*	Nachbarschafts−verknüpfungen (LINK)	Protospinsysteme *k*		
				perfekt	generierte	mit AS Typen, je AS
CheY	128	124	118	124	2314	548
C21w.hf	148	145	142	145	6125	>600
Trigger mit HIS Tag	113	110	107	107	158 − 224	14
Trigger ohne HIS Tag	103	100	97	100	158 − 224	14

Tabelle 1−3 Parameter p *und* k *für die untersuchten Proteine und Konfigurationen ohne Nebenbedingungen. "*belegbare Aminosäuren*" sind die Aminosäurepositionen, die ein Protospinsystem des Experimentsets enthalten können (Gl. 2−5). "*perfekt" *bezeichnet die Anzahl der Spinsysteme, die theoretisch existieren sollten. "*Generiert" *ist die Anzahl, die mit der Standardkonfiguration der Protospinsystemsuche bei größter Toleranz gefunden wurde. »*mit AS Typen*« ist die Anzahl, die im Mittel, nach der Reduktion der generierten Protospinsystemlisten des AS−Caches mittels der AS Typenerkennung, einer Aminosäure zugewiesen wurde. Die Anzahl der möglichen Protospinsysteme je Aminosäure sinkt dabei durchschnittlich um eine Größenordnung.*

b	CheY, perfekt	CheY, generiert	C21w.hf, perfekt	C21w.hf, generiert	Trigger ohne HIS, generiert	Trigger ohne HIS, mit AS Typen
0	1	1	1	1	1	1
1	15376	286936	21025	888125	16274	1442
2	116311752	40816502532	217987200	391599180000	130305918	956046
3	$\sim 5.77 \cdot 10^{12}$	$\sim 3.84 \cdot 10^{15}$	$\sim 1.48 \cdot 10^{12}$	$\sim 1.14 \cdot 10^{17}$	$6.84 \cdot 10^{11}$	$3.86 \cdot 10^{8}$
...						
$b_{max}-1$	$\sim 1 \cdot 10^{209}$	$\sim 1 \cdot 10^{414}$	$\sim 1 \cdot 10^{254}$	$\sim 1 \cdot 10^{546}$	$\sim 1 \cdot 10^{207}$	$\sim 1 \cdot 10^{12}$
b_{max}	$\sim 1 \cdot 10^{207}$	$\sim 1 \cdot 10^{415}$	$\sim 1 \cdot 10^{251}$	$\sim 1 \cdot 10^{548}$	$\sim 1 \cdot 10^{207}$	$\sim 1 \cdot 10^{10}$
Σ	$\sim 1 \cdot 10^{209}$	$\sim 1 \cdot 10^{415}$	$\sim 1 \cdot 10^{254}$	$\sim 1 \cdot 10^{548}$	$\sim 1 \cdot 10^{208}$	$\sim 1 \cdot 10^{40}$

Tabelle 1−4 **a(M_i)** *bei verschiedenen Belegungsgraden* b *für die untersuchten Proteine und Konfigurationen ohne Nebenbedingungen (entspricht der allCX Konfiguration oMa). Die letze Zeile enthält die Summe aller Zustände* ***a(S).*** *Es liegt in der gleichen Größenordnung wie* $\max(a(M_b))$ *.*

Durch Nebenbedingungen, die die erlaubten Positionen je Protospinsystem einschränken, wird die Anzahl der Zustände in jeder M_n Ebene reduziert. Für das N−ASS gibt es zwei Gruppen von Nebenbedingungen, die Resourcenkonkurrenz und die Positionsbeschränkung durch die Aminosäuretypen. $a(M_i)$ in Tabelle 1−4 ist daher die Obergrenze für die Anzahl der Zustände mit Nebenbedingungen. Die wahre Anzahl der Zustände unter Beachtung der Nebenbedingungen liegt niedriger als $a(M_i)$. Sie hängt von den Nebenbedingungen ab und kann allgemein nur durch direktes Abzählen aller Zustände berechnet werden. Selbst für "Trigger,generiert", mit nur 10^{208} Zuständen (verglichen mit CheY, siehe Tabelle 1−4), ist dies unmöglich.

Eine Abschätzung über die gemittelte Anzahl der Protospinsysteme pro Aminosäure ergibt für "Trigger", nach Berücksichtigung der AS−Typenbeschränkungen, daß diese Reduktion aber sehr effizient ist (hier 168 Größenordnungen)!

1.2.6 Topologie des Zustandsraumes

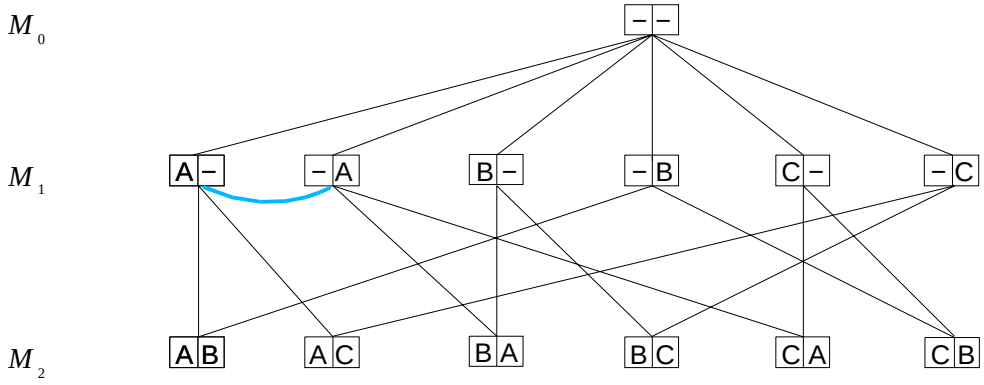


Abbildung 1−1 Vollständiger Zustandsraum für p = 2 *mit den Elementen* { A, B, C } *für die Operationen* T^+ *und* T^- *. − bezeichnet eine leere Position. Die Kästchen bezeichnen jeweils einen Zustand* $\boldsymbol{S_i}$ *. Die geraden Kanten sind alle* $T^{\pm}$ *Operationen, die in diesem Zustandsraum erlaubt sind. Die Operation* $T^{swap}_{1\Leftrightarrow 2}$ *wurde als Bogen eingetragen.*

Abbildung 1−1 zeigt den vollständigen Graph für ein Permutationsproblem mit 2 Positionen und 3 Elementen. Die Permutationen $\boldsymbol{S_i}$ bilden einen Graphen mit den Permutationen als Eckpunkten (Knoten) und den Transformationsoperationen T^{op} als Kanten. Die Kanten und Knoten des Zustandsraumes tragen je ein Gewicht $\boldsymbol{w_i}$, das im Kontext der Graphentheorie der Weglänge oder einer Qualität entspricht. Im N−ASS entspricht die Qualität der Knoten der Qualität eines $\boldsymbol{S_i}$, während die Kantenqualität die Qualitätsveränderung bei einer bestimmten Operation ist.

Alle mittels irgendeiner Transformation erreichbaren Zustände sind die Nachbarn einer Permutation. Die Nachbarschaftsbeziehungen zwischen den Zuständen hängen daher von der Art der Transformationsoperatoren ab. Die Kanten zwischen den Zuständen werden durch die Operationen definiert. Die Topologie des Zustandsraumes hängt daher von dem Set der definierten Operationen T^{op} ab.

Definition: *Nachbarschaft* ***N*** *einer Lösung* $\boldsymbol{S_i}$: Die Nachbarn ***N*** einer Lösung $\boldsymbol{S_i}$ sind die $\boldsymbol{S_j}$, die von $\boldsymbol{S_i}$ mit einer beliebigen Transformationsoperation ***T*** erreicht werden können.

$$\boldsymbol{S}_i \in \boldsymbol{S}, \forall \boldsymbol{S}_j \in \boldsymbol{N} : T(\boldsymbol{S}_i) \rightarrow \boldsymbol{S}_j \qquad (1-10)$$

1.2.7 Die Operationen im Lösungsraum

Die einfachsten Operationen sind die Null Operation T^{NULL} und die Additions− und Subtraktionsoperation, T^+ und T^-. Alle mittels der T^+ − und T^- −Transformationen erreichbaren $\boldsymbol{S_i}$ sind Nachbarn einer Lösung. Jeder Punkt im Zustandsraum ist über eine oder mehrere Operationen $T^{\pm}$ von $\boldsymbol{S}_0$ zu erreichen. Der Zustandsraum ist daher ein geschlossener, gewichteter und gerichteter Graph. T^+ ist die inverse Operation zu T^- und umgekehrt.

Operation	Definition:
T^{NULL}	überführt den Zustand in sich selbst.
T^{+}	fügt ein Element in eine leere Position ein.
T^{-}	entfernt ein Element von einer belegten Position.

Tabelle 1−5 Die Elementaroperationen im Zustandsraum

$T^{\pm}_{E,pos}$ bezeichnet die Operation, die an der Position *pos* das Element *E* hinzufügt oder entfernt.

$T^{\pm}_{E,}$ bezeichnet die Operation, die an einer beliebigen Position das Element *E* hinzufügt oder entfernt.

Jede andere Operation kann in diese Operationen zerlegt werden. Ein Beispiel für eine zusammengesetzte Operation ist $T^{swap}_{i \Leftrightarrow j}$. Die swap−Operation setzt sich aus den 4 Elementaroperationen $\{T^{-}_{cx_{1,i}}, T^{-}_{cx_{2,j}}, T^{+}_{cx_{2,i}}, T^{+}_{cx_{1,j}}\}$ zusammen. Swap−Operationen verbinden nur Zustände mit gleichem Belegungsgrad, d.h. Anfangs− und Endpunkt gehören beide der gleichen Menge $\mathbf{M_n}$ an. Die swap−Operation ist ihre eigene inverse Operation.

1.2.8 Differenz zweier Zustände in *S*

Für die Crossover Operatoren in Kap. 2 wird die Differenz zweier Zustände benötigt.

Definition: **Differenz:** Die Differenz zweier Zustände ist ein *Weg (Pfad)* im Zustandsraum entlang der Kanten in dem Graph, der von den Elementaroperationen, ohne T^{NULL}, aufgespannt wird. Der Weg wird unifiziert, indem die Operationen zuerst nach der Position im Zustand und dann in der Reihenfolge T^{-} und T^{+} sortiert werden.

Beispiel: Die Differenz der Zustände, die in Abb.1−1 durch die Operation $T^{swap}_{1 \Leftrightarrow 2}$ verbunden werden, ist $diff(\boldsymbol{S}_1 \rightarrow \boldsymbol{S}_2) = \{T^{-}_{A,1}, T^{+}_{NULL,1}, T^{-}_{NULL,2}, T^{+}_{A,2}\}$.

$diff(\boldsymbol{S}_1 \rightarrow \boldsymbol{S}_2)$ ist gleichzeitig der kürzeste Weg der $\boldsymbol{S_1}$ und $\boldsymbol{S_2}$ verbindet. Seine Länge ist die Anzahl der Operationen und ist damit proportional der Hamming−Distanz wie sie für binäre Vektoren definiert ist.

1.2.9 Anzahl der Nachbarn, die durch die gleiche Operation $T^+_{E_1}$ ver−bunden werden

Die S_i in benachbarten Mengen M_b und M_{b+1} werden durch T^+ Operationen mit verschiedenen Elementen E verbunden. Einige der T^+ fügen dabei das gleiche neue Element in verschiedene S_i ein. $a(T^+_E)$ ist die Anzahl der T^+ −Operationen, die das Element E in einen Zustand S_i einfügen. Wenn ein Element in einen Zustand an verschiedenen Stellen eingefügt werden kann, ist $a(T^+_E)$ größer 1.

Definition: $T^+_{E_1}$: Die T^+ −Operation, die das Element E_1 einfügt.

Von der leeren Menge M_0 zur Menge M_1 mit genau einem Element gehen k verschiedene Operationen $T^+_{E_1}$ aus, die jeweils nur ein Element einfügen. Gleichzeitig gibt es genau p verschiedene Startpositionen im gleichen S_i , auf die die Operation wirken kann.

$$a(T^+(M_0,M_1)) = p \cdot k \qquad (1-11)$$

Von der Menge M_1 zur Menge M_2 gilt daher Gl. 1−12 für die Anzahl der Operationen, die von einem S_i in M_1 starten. Dies ist gleichzeitig die Größe der $T^+_{E_1}$ Nachbarschaft einer einzelnen Permutation.

$$a(T^+(M_1,M_2)) = (p-1)\cdot(k-1) \qquad (1-12)$$

Anzahl der belegten Positionen	**Anzahl T^+ −Nachbarn des Zustandes S_i in der Menge M_{b+1}**	**Anzahl unterscheid−barer T^+ Operatio−nen mit gleichem E.**	**Anzahl der T^+ −Nach−barschaftsbeziehungen zwischen der M_b und M_{b+1} .**	**Anzahl der T^+ −Operationen, die von M_{b-1} ankom−men.**
b	a(T^+ Nachbarn)	$a(T^+_{S_i,\,S_{i+1}})$	$a(M_b \to M_{b+1})$	$a(M_{b-1} \to M_b)$
0	p	k	k* p	−
1	p−1	(k−1)	(p−1)*(k−1)	k
i	p−i	(k−i)	(p−i)*(k−i)	a(M_i) * i
b_{max}−1	1	k−b_{max}	k−b_{max}	
b_{max}	−	−	−	

Tabelle 1−6 Anzahl der Kanten und Operationen in Abhängigkeit von b*.*

1.2.10 Anzahl der Nachbarn eines Zustandes mit T^+ / T^-

Bei einem Protospinsystempool, ohne Nebenbedingungen für die Positionen der Elemente, gehen von jedem $\boldsymbol{S}_i$ genau b T^- Operationen zu einem Zustand in $\mathbf{M}_{b-1}$. Von einer Position in $\boldsymbol{S}_i$ starten $(k-b)$ T^+ Operationen. In ein $\boldsymbol{S}_i$ können genau $(p-b)$ verschieden neue Elemente in einer T^+ Operation eingefügt werden. Berücksichtigt man alle ungenutzten Elemente, gibt es $(k-b)\,(p-b)$ Kanten.

- Anzahl der $T^-_{E,}$ Operationen, die von einem $\boldsymbol{S}_i$ starten.

$$a(T^-_{E,}(\boldsymbol{S}_b \rightarrow \boldsymbol{S}_{b-1})) = b \qquad (1\text{–}13)$$

- Anzahl der $T^+_{,pos}$ Operationen, die von einer festen leeren Position *pos* in $\boldsymbol{S}_i$ starten.

$$a(T^+_{,pos}(\boldsymbol{S}_b \rightarrow \boldsymbol{S}_{b+1})) = (k-b) \qquad (1\text{–}14$$

- Anzahl der $T^+_{E,}$ Operationen, die ein bestimmtes Element E in $\boldsymbol{S}_i$ einfügen.

$$a(T^+_{E,}(\boldsymbol{S}_b \rightarrow \boldsymbol{S}_{b+1})) = (p-b) = f \qquad (1\text{–}15)$$

Die Anzahl der T^+ Operationen, die von einer Permutation $\boldsymbol{S}_i$ starten, ist daher

$$a(T^+(\boldsymbol{S}_b \rightarrow \boldsymbol{S}_{b+1})) = (k-b)\cdot(p-b) \qquad (1\text{–}16)$$

Die Anzahl der Nachbarn, die mit einer Veränderung erreicht werden können, ist dann

$$a(T^\pm(\boldsymbol{S}_b \rightarrow \boldsymbol{S}_{b\pm1})) = b+(k-b)\cdot(p-b) \qquad (1\text{–}17)$$

oder

$$a(T^\pm(\boldsymbol{S}_b \rightarrow \boldsymbol{S}_{b\pm1})) = b+(p-b)\cdot(p-b) = b+(p-b)^2 \quad \text{wenn p=k ist} \qquad (1\text{–}18)$$

Die Anzahl der möglichen Transformationen von einer $\mathbf{M}_b$ Ebene zu den benachbarten Ebenen, ohne Berücksichtigung der Nebenbedingungen durch Resourcenkonflikte, ist daher

mit theoretischen, perfekten Protospinsystemen (k=p):

$$\begin{aligned} a(T^\pm(\boldsymbol{S}_b \rightarrow \boldsymbol{S}_{b\pm1})) &= (b+(p-b)^2)\cdot\left(\frac{p!}{(p-b)!}\cdot\frac{p!}{(p-b)!b!}\right) \\ &= (b+(p-b)^2)\cdot\left(\frac{p!^2}{(p-b)!^2\,b!}\right) \end{aligned} \qquad (1\text{–}19)$$

mit Protospinsystemen, die Defekte und Fehldeutungen enthalten können (k>p):

$$a(T^\pm(\boldsymbol{S}_b \rightarrow \boldsymbol{S}_{b\pm1})) = (b+(k-b)(p-b))\cdot\left(\frac{p!}{(p-b)!}\cdot\frac{k!}{(k-b)!b!}\right) \qquad (1\text{–}20)$$

1.2.11 Anzahl der T^{swap} Operation in M_b und *S*

- Die Anzahl der T^{swap} in $\mathbf{M_b}$ kann durch eine Separation berechnet werden. Dazu wird $a(T^{swap}(M_b))$ in die Anzahl der Kombinationen der vertauschbaren Elemente, die Anzahl der Kombinationen der Startpositionen für T^{swap} und den Zustand der restlichen Sequenz aufgeteilt.
- Anzahl der Protospinsystempaare, die aus dem Protospinsystempool *ohne Beachtung* der Reihenfolge gebildet werden kann.

$$a_{paare} = \frac{k!}{(k-2)!} \qquad (1-21)$$

- Anzahl der Positionspaare, die in der Sequenz *unter Beachtung* der Reihenfolge ausgewählt werden kann.

$$a_{pos} = \frac{p!}{(p-2)!} \qquad (1-22)$$

- Anzahl der Permutationen mit den restlichen Elementen über die restlichen Positionen in dem noch ungenutzten Anteil des Belegungsgrades der Sequenz *unter Beachtung* der Reihenfolge.

$$a_{perm} = \frac{k!}{(k-2)!} \cdot \frac{(p-2)!}{((p-2)-(b-2))!(b-2)!} \qquad (1-23)$$

Die Anzahl der T^{swap} in $\mathbf{M_b}$ ist dann:

$$\begin{aligned} a(T^{swap}(M_b)) &= a_{paare} \cdot a_{pos} \cdot a_{perm} \\ &= \frac{k!}{(k-2)!} \cdot \frac{p!}{(p-2)!} \cdot \frac{(p-2)!}{((p-2)-(b-2))!(b-2)!} \end{aligned} \qquad (1-24)$$

Die Anzahl der T^{swap} in der gesamten Lösungsmenge ist dann:

$$a(T^{swap}(\boldsymbol{S})) = \sum_{b=2\ldots p} a(T^{swap}(M_b)) \qquad (1-25)$$

Tabelle 1−7 listet Zahlenwerte für Gl. 1−25 für das Problem "Trigger" auf.

Problem	**p**	**k**	**b**	$a(T^{swap}(\boldsymbol{S}))$
Trigger ohne HIS, ohne AS−Typen	113	224	103	10^{234}
Trigger ohne HIS, mit AS−Typen	113	158	103	10^{211}
Trigger mit HIS, ohne AS−Typen	113	224	113	10^{255}

*Tabelle 1−7 Anzahl der T^{swap} in **S***

1.2.12 Optimale Lösung

Definition: *Globale Lösung* $\boldsymbol{S_{global}}$: Jede Lösung $\boldsymbol{S_i}$ mit der maximalen Qualität aller $\boldsymbol{S_j}$ in $\boldsymbol{S}$.

$$\boldsymbol{S}_i \in \boldsymbol{S}, \forall \boldsymbol{S}_j \in \boldsymbol{S} : Q(\boldsymbol{S}_i) \geq Q(\boldsymbol{S}_j) \qquad (1-26)$$

Definition: *Lokales Optimum* $\boldsymbol{S_{lokal}}$: Eine Lösung $\boldsymbol{S_i}$ deren Qualität größer oder gleich der aller benachbarten $\boldsymbol{S_j}$ ist.

$$\boldsymbol{S}_i \in \boldsymbol{S}, \forall \boldsymbol{S}_j \in \boldsymbol{N} : Q(\boldsymbol{S}_i) \geq Q(\boldsymbol{S}_j) \qquad (1-27)$$

Ob ein $\boldsymbol{S_i}$ ein lokales Optimum ist, hängt daher von der Nachbarschaft und damit von den definierten Transformationsoperationen ab.

1.3 Kombinatorische Optimierungsverfahren

Das N−ASS Problem fällt in die Klasse der kombinatorischen Optimierungsprobleme, da der Lösungsraum aus einer finiten Menge diskreter Zustände besteht. Es gibt weder eine ableitbare, zusammenhängende Qualitätsfunktion, noch können Zwischenzustände zwischen den Zuständen des Lösungsraumes konstruiert werden.

1.3.1 Qualitätsfunktion Q(S_i)

Die Qualitätsfunktion des Optimierungsverfahrens ordnet die S_i relativ zu andern S_i ein. Für das N−ASS implementiert sie z.B. die Bewertungsfunktion aus Kapitel 1. Q(S_i) muß für zwei S_i nur die relative Reihenfolge der Qualitätsskala wiederspiegeln, d.h. die absoluten Zahlenwerte Q(S_i) sind nebensächlich.

In dieser Arbeit werden bessere Qualitätswerte mit höheren Zahlenwerten belegt. Dadurch hat die leere Zuweisung den Qualitätswert 0. Alle anderen S_i haben eine höhere Qualität und einen höheren Zahlenwert.

1.3.2 Das *traveling salesman*−Problem

Das N−ASS ist mit einem bekannten kombinatorischen Optimierungsproblem, dem *traveling salesman* Problem (TSP), verwandt.

Beim Travelling Salesman Problem (**TSP**) wird die optimale Rundreise für einen Handlungsreisenden gesucht, der mehrere Städte zu möglichst niedrigen Kosten bereisen will. Jede Stadt darf daher nur einmal besucht werden. Außerdem sind die Kosten proportional zur Länge des Weges zwischen zwei Städten.

Das **TSP** (Travelling Salesman Problem) ist definiert als [Reinelt94, S.6]

Definition: *TSP*: Suche den kürzesten Rundweg, durch einen gegebenen geschlossenen ungerichteten Graphen **D**, der jeden Knoten genau einmal berührt.

Nr. der Stadt	Vorgänger	Nachfolger
1	4	3
2	3	4
3	1	2
4	3	1

Vor der **k-opt** Operation.

Nr. der Stadt	Vorgänger	Nachfolger
1	**2**	3
2	**4**	**1**
3	1	**4**
4	3	**2**

Nach der **k-opt** Operation.

Tabelle 1−8 Eine Implementierung des TSP für 4 Städte mit einer `k-opt` *Operation.*

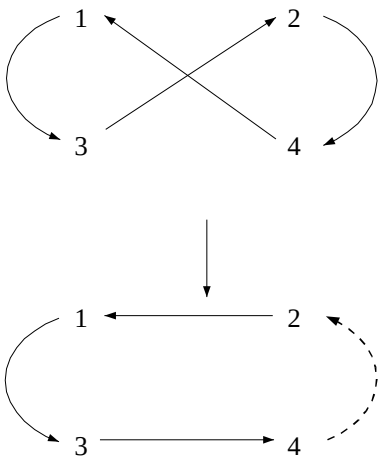


Abbildung 1−2 Beispiel für die `k-opt` *Operation im TSP.*

1.3.2.1 Repräsentation eines Weges für das TSP

Eine typische Repräsentation eines Weges für das TSP ist eine Tabelle, in der jede Stadt (Knoten) durch eine Zeile mit den Nummern der vorhergehenden und nachfolgenden Stadt vertreten wird.

Das TSP und das N−ASS Problem versuchen eine begrenzte Menge von Resourcen (Städte oder Protospinsysteme) in eine Reihenfolge zu bringen, die eine Zielfunktion optimiert. Im Unterschied zum N−ASS sind die Verbindungen (Kanten) zwischen den Städten nicht gerichtet. Im TSP darf sich daher die Reihenfolge der Städte in einer Tour umdrehen, ohne daß die Tour ungültig wird. Diese Eigenschaft macht man sich beim so genannten `k-opt` Optimierungsschritt zu Nutze.

1.3.2.2 `k-opt` Operation

Die **`k-opt` Operation** ist der Grundbaustein der meisten Optimierungsalgorithmen für das TSP. Dabei werden *k* Kanten zufällig ausgewählt und die beteiligten *2*k* Städte danach so verknüpft, daß die Länge der entstehenden Kanten minimal ist. Dabei kann sich die Richtung der Kanten oder von Teilstrecken umdrehen, diese Operation ist daher nur für ungerichtete Graphen geeignet. Optimiert wird die Tour dadurch, daß die Summe der Kantenlängen durch die Aktion verringert werden muß.

Das N−ASS unterscheidet sich vom TSP auf mehrere Weise. Übertragen auf die Terminologie des TSP ist das N−ASS ein Problem, in dem zwar die Existenz von p Städten bekannt ist, aber die Landkarten zu ungenau sind, um diese von anderen Landmarken, wie Berge, Vorstädte oder Ortsteile, zu unterscheiden. Der Reisende kann daher auch falsche Städte anlaufen. Außerdem dürfen die Wege zwischen den Städten nur in einer bestimmten Richtung durchlaufen werden. Zusätzlich passen die Städte nur auf bestimmte Positionen im Pfad. Die beiden letzten Eigenschaften des N−ASS verbieten den `k-opt` Schritt als Bestandteil der Optimierung.

Das N−ASS ist daher ungleich komplexer als ein TSP mit gleicher 'Städteanzahl' und man erwartet daher einen wesentlich trügerischeren Lösungsraum mit tiefen lokalen Optima.

1.3.3 NP−Hart

Um die zu erwartende Laufzeit eines Entscheidungsproblems zu charakterisieren, teilt man die Probleme in die Klassen P (Polynominale Komplexität) und NP (Nichtdeterministische Polynominale Komplexität) ein [Knuth97, Reinelt94, Hopcroft79]. Ein Algorithmus A hat *polynominale Zeitkomplexität*, wenn es ein Polynom p(n) gibt, das die garantierte maximale Laufzeit $\mathbf{t_A}$ beschreibt, so daß die Komplexität $t_A(n)=O(p(n))$ ist. Entscheidungsprobleme, für die ein Algorithmus mit polynominaler Zeitkomplexität auf einer deterministischen Turing−Maschine (TM) existiert, gehören der Klasse P an. Entscheidungsprobleme, für die ein solcher Algorithmus auf einer nichtdeterministischen Turing−Maschine [Aho74, Hopcroft79] existiert, gehören zu NP [Reinelt94, Lawler85, Hopcroft79]. Während die deterministische Turing−Maschine ein formales Modell für alle existierenden Computer ist, kann die nichtdeterministische TM bisher nur als Gedankenexperiment 'realisiert' werden.

Deterministische TM und nichtdeterministische TM unterscheiden sich dabei nur in der Methode, wie sie eine anstehende Wahlmöglichkeit abarbeiten. Eine deterministische TM muß sich bei jeder Wahlmöglichkeit für einen der Zweige entscheiden oder sie führt jeden Zweig der Auswahl nacheinander aus, während die

nichtdeterministische TM alle Wahlmöglichkeiten *gleichzeitig* bearbeitet. Die nichtdeterministische Turing−Maschine kann man daher simulieren, indem man eine deterministische Turing−Maschine ablaufen läßt und sie bei jeder Verzweigung mit *m* Wahlmöglichkeiten durch *m* identische Kopien der aktuellen Maschine ersetzt, und jeden Zweig von genau einer Maschine bearbeiten läßt. Nach *n* Entscheidungen mit jeweils *m* Wahlmöglichkeiten erhält man so n^m deterministische Turing−Maschinen, die alle *gleichzeitig* je einen Teil−zweig bearbeiten[1].

Da man jedes Optimierungsproblem in ein Entscheidungsproblem umwandeln kann und die Optimie−rung das Entscheidungsproblem lösen muß, ist eine Aussage über die Komplexität des Entscheidungspro−blems eine Aussage über die Komplexität des Optimierungsproblems.

Für Probleme der Klasse P gibt es sequentielle Algorithmen, deren Laufzeit sich, bei einer linearen Ver−größerung des Problems, nur polynominal vergrößern. Ein NP−Problem kann dagegen nur mittels eines gedachten, nichtdeterministischen Algorithmus in polynominaler Zeit gelöst werden, d.h. auf einer nichtde−terministischen Turing−Maschine. Die nichtdeterministische Turing Maschine wählt quasi alle zulässigen Veränderungen parallel aus, da sie bei jeder zu treffenden Entscheidung alle Entscheidungszweige gleich−zeitig durchläuft. Da man diese Maschine nicht in heutigen Computern implementieren kann, muß man sie in sequentiellen Algorithmen simulieren, indem man alle parallel gewählten Entscheidungen sequentiell absucht. Die Laufzeit des entsprechenden sequentiellen Algorithmus vergrößert sich aber immer schneller als die des zugehörigen NP Algorithmus, sie steigt meist exponentiell.

Daher möchte man wissen, ob man den nichtdeterministischen Algorithmus mittels einem determinisi−schen Algorithmus in polynominaler Zeit simulieren kann (P=NP)? Für das TSP weiß man, daß es zur Klasse NP gehört, obendrein ist es NP−vollständig, d.h. es kann nur in polynominaler Zeit gelöst werden, wenn man einen nichtdeterministischen Computer verwendet [Lawler85, S. 58].

Nach Lawler [Lawler85, S. 58] ist es allgemeine Auffassung, das P≠NP für das TSP ist, d.h. es gibt keinen deterministischen Algorithmus, der das TSP in polynominaler Zeit mit einem P−Algorithmus exakt lösen kann.

Um dennoch eine vertretbare Lösung zu erhalten, muß man auf ein *heuristisches* Verfahren auswei−chen. Heuristische Verfahren sind quasi−Implementierungen der nichtdeterministischen Algorithmen. Sie verzichten auf eine vollständige Implementierung der parallelen Suche. Dabei verliert man zwar die Garantie einer exakten Lösung des Problems, erhält dafür aber eine gute Lösung in polynominaler Zeit. Gute Heuri−stiken finden sogar meist das globale Maximum.

Deterministische Heuristiken wählen von einem bestimmten Punkt des Lösungsraumes immer die glei−che Veränderung, während die z*ufallsgesteuerten* jede der erlaubten Veränderungen nur mit einer bestimm−ten Wahrscheinlichkeit auswählen. Sie wählen quasi alle Veränderungen parallel aus. Die Wahrscheinlich−keit für ein Veränderung ist meist von Start und Zielzustand abhängig. [Lawler85, S. 53]. Daher sind sie angenäherte Implementierungen des nichtdeterministischen Computers und können eine Näherungslösung in polynominaler Laufzeit finden.

1 Trotzdem ist dies keine praktisch durchführbare Implementierung der allgemeinen, nichtdeterministischen TM, da diese nicht nur kleine *n* sondern jede beliebige Entscheidungstiefe *n* bearbeiten können muß.

1.3.4 Lösungsansätze

Der einfachste Algorithmus ist eine vollständige Suche. Dabei garantiert der Algorithmus, daß das globale Maximum gefunden wird, da jeder Punkt im Lösungsraum mindestens einmal besucht wird. Den Lösungs−raum vollständig zu durchsuchen, ist aber, angesichts der Größe des Lösungsraums des N−ASS Problems, unmöglich. Für das N−ASS Problem"Trigger" ist der Lösungsraum in der Größenordnung 10^{208} ohne AS−Typenbeschränkungen und 10^{40} mit Beschränkung (siehe Tabelle 1−4).

Daher wurde nach einem heuristischen Algorithmus gesucht, der nur einen Teil des Lösungsraumes durchsucht.

1.3.5 Lokale Suche

Die meisten Heuristiken implementieren das Prinzip der lokalen Suche, indem sie die Umgebung einer erreichten Lösung nach besseren Lösungen absuchen. Dabei wird ein einmal erzieltes Ergebnis mittels einer Produktionsregel inkrementell verändert.

Definition: *lokale Suche*: Ein Algorithmus, der die direkte *k*−Schritt Nachbarschaft des aktuellen Indi−viduums, nach einen besseren Individuum durchsucht, führt eine lokale Suche durch. Nach−barn, die nur über mehr als k Schritte erreichbar sind, werden nicht untersucht.

Die *hill−climber* bilden eine Klasse von Algorithmen, die sich während ihrer lokalen Suche nur aufwärts, d.h nur zu einem $\boldsymbol{S_i}$ mit höherer Qualität, bewegen. Eine globale *hill−climber Umgebung* ist daher eine Nachbar−schaft in der jede Verbesserung näher zum globalen Maximum führt. Zum *Einzugsbereich eines Optimums* gehören alle Individuen, die mit dem Optimum mit einem *Pfad* verbunden sind. Für einen *hill−climber* ist der Einzugsbereich ein Pfad, auf dem jede Veränderung auch eine Verbesserung darstellt.

Ein *greedy* Verfahren akzeptiert die erste erlaubte Veränderung, die die Qualität verbessert. Andere Verfahren durchsuchen immer die gesamte Nachbarschaft und wählen dann die beste Veränderung. Wenn ein solches Verfahren in den *ausschließlichen* Einzugsbereich eines lokalen Optimums gerät, findet es nicht mehr aus diesem Trichter heraus. Das Verfahren *konvergiert* dann *vorzeitig*.

Die Erfahrungen mit den *greedy*−Algorithmen zeigen, daß sie meist vorzeitig konvergieren, wenn die Qualitätshyperfläche rauh ist, d.h. wenn es viele kleine lokale Suboptima gibt. Daher muß man dem Algo−rithmus erlauben, Gebiete mit schlechterer Qualität zu durchqueren, um dem lokalen Optimum zu entkom−men. Dies wird in verschiedenen Algorithmen implementiert. Ob ein heuristisches Verfahren das Maximum findet oder in einem lokalen Maximum stecken bleibt, hängt von der gewählten Produktionsregel ab.

1.3.6 Produktionsregel

Eine lokale Suche braucht eine Produktionsregel, die das nächste zu bewertende Individuum erzeugt.

Die einfachste Möglichkeit ein neues Individuum zu erzeugen bietet das *Monte Carlo Verfahren*. Dabei wird die nächste Lösung jeweils erwürfelt und ein besserer Zustand nur zufällig erreicht. Die aufeinander fol−genden Lösungen sind dabei von einander unabhängig. Ob eine akzeptabel Lösung gefunden wird, hängt von der Größe des Lösungsraumes ab.

Andere Produktionsregeln verwenden eine inkrementelle Veränderung des Ausgangszustandes. Dazu gehören `k-opt` im TSP, *Lin−Kernighan*, *simulated annealing* und *threshold accepting*.

1.3.7 Lin−Kernighan Verfahren

Ein bekanntes Verfahren ist das *Lin−Kernighan* Verfahren (LK). Es verwendet den `k-opt` Zug für das TSP [Lin&Kernighan73, Reinelt94, S.123]. Es ist der Nachfolger einer Reihe von deterministischen Optimie−rungsverfahren für das TSP, die den `k-opt` Zug als Grundbaustein benutzen.

`k-opt` betrachtet jeweils eine Gruppe von `k` zufällig ausgewählten Wegen und verknüpft die damit ver−bundenen Städte so, daß die Summe der Wege minimal wird. Reines `k-opt` ist daher ein hill−climber.

Bei den *deterministischen* Verfahren kann man zu jedem Zeitpunkt den nächsten Schritt des Algorith−mus voraussagen. Ein deterministisches `k-opt` Verfahren startet mit einem Monte−Carlo Zustand und pro−biert erschöpfend alle diesem Zustand zulässigen `k-opt` Schritte aus. Das Verfahren endet, wenn alle zulässigen `k-opt` Schritte getestet wurden und keine Verbesserung des aktuellen Zustandes gefunden wurde.

Die Operation, die `2-opt` im N−ASS am nächsten kommt, ist die T^{swap} −Operation. `2-opt` ist zwar ähnlich T^{swap}, aber nicht jeder `2-opt` Zug ist im N−ASS technisch möglich, da `k-opt` voraussetzt, daß die Richtung der Strecken umgedreht werden darf. Für das "Trigger" Problem gibt es schätzungsweise 10^{212} bzw. 10^{44} T^{swap} Operationen. Es ist daher nicht möglich alle systematisch durch zu probieren.

Durch das deterministische Vorgehen sind außerdem nur bestimmte Ausschnitte des Lösungsraumes für das Verfahren zugänglich. Diese Verfahren konvergieren daher meist vorzeitig , da nicht jeder Startpunkt im Einzugsbereich des globalen Optimums liegt. Verschiedene Endpunkte erreicht man nur mit unterschied−lichen Startzuständen.

Lin−Kernighan erweitert `k-opt`, indem es zusätzlich Aktionen erlaubt, die die Qualität der erreichten Lösung verschlechtern. Die verschlechternde Aktion soll hier als `k-change` bezeichnet werden. Sie unter−scheidet sich von `k-opt` nur, indem die Summe der Weglängen hinterher nicht nur kleiner, sondern auch größer sein darf. `k-change` darf die Qualität der Startlösung verschlechtern.

LK führt von einer Startlösung erst mehrere *k−change* Aktionen und danach einige `k-opt` Aktionen aus und bewertet die Qualität der erreichten Lösung erst, nachdem die letzte Aktion ausgeführt wurde. Ist die Qualität dann besser als die des Startzustandes, wird die gesamte Aktionskette akzeptiert, sonst verworfen. Außerdem versucht LK mehrere unterschiedliche Aktionsketten, bevor es sich für eine entscheidet.

Durch die *k−change* Aktionen und die parallel bewerteten Aktionsketten kann LK leichter aus lokalen Optima entkommen als alle anderen Heuristiken oder deterministischen Verfahren. Dadurch verbessert sich sowohl die Qualität der Lösungen als auch die erforderliche Laufzeit. Dabei hängt die Qualität der Lösungen nur wenig vom gewählten Startzustand ab.

Lin−Kernighan gehört damit zu den besten bekannten Heuristiken für das TSP.

1.3.8 Simulated Annealing

```
Simulated Annealing Algorithmus:
begin
   Erzeuge einen Anfangszustand S.
   wähle eine Anfangstemperatur ϑ und einen Wiederholungsfaktor r.
   loop:  bis das Abbruchkriterium erfüllt ist.
      führe den Metropolis Algorithmus(S,r)aus.
      wende ein Abkühlungsschema auf ϑ an.
      berechne r neu.
   end loop
End

Metropolis Algorithmus (Zustand S, zahl n )
   kB ist die Boltzman Konstante.
   ϑ  Temperatur.
begin
   loop: iteriere n mal.
      lege eine Kopie von S an -> S'
      erzeuge eine kleine Veränderung in S'
      bewerte S' und berechne die Energiedifferenz ΔQ
      wenn ΔQ >= 0  oder (ΔQ < 0 und random( 0...1) < exp( -ΔQ/(kB ϑ) ) )
         akzeptiere S' als neues S
   end loop
end
```

Simulated Annealing (SA) [Kirkpatrick83] ist ein lokales Suchverfahren, das seine Aktionen nichtdeterministisch wählt. Es wählt die Veränderung in Abhängigkeit von der Qualitätsverbesserung durch die Veränderung aus. Die Wahrscheinlichkeit, mit der die Aktion $\boldsymbol{S}_i$ −> $\boldsymbol{S}_j$ akzeptiert wird, ist:

$$P_{i,j}(T^{op},\theta)=\begin{cases}1 & \Delta\ Q_{i,j}\geq 0\\ e^{-\frac{\Delta\ Q_{i,j}}{k_b\theta}} & \Delta\ Q_{i,j}<0\end{cases} \qquad \textit{(1−28)}$$

$$\text{für } \Delta\ Q_{i,j}=Q(\boldsymbol{S}_i)-Q(\boldsymbol{S}_j)$$

Die Optimierung startet mit einer hohen Temperatur ϑ und wird im Verlauf der Optimierung bis auf einen Wert nahe Null gesenkt. Da ein SA als eine Markow−Kette dargestellt werden kann, kann man beweisen, daß ein SA garantiert global konvergiert, wenn man ϑ unendlich langsam abkühlt, damit die Ergodizität der Kette garantiert ist [Hajek85, Laarhoven88]. In der Praxis darf man die Temperatur meist wesentlich schneller absenken, muß dann aber ein Kühlschema an das Problem anpassen.

Die Qualität der Optimierung hängt daher in nicht trivialer Weise von dem gewählten Kühlschema ab [Reinelt94,S.156]. Mit einem ungeeigneten Kühlschema konvergiert SA vorzeitig oder benötigt eine zu hohe Rechenzeit.

1.3.9 Threshold Accepting

```
Threshold Accepting Algorithmus:
begin
   Erzeuge einen Anfangszustand S.
   wähle eine Schwellwert θ und einen Wiederholungsfaktor r.
   iteriere bis das Abbruchkriterium erfüllt ist.
      iteriere r mal.
         lege eine Kopie von S an -> S'
         erzeuge eine kleine Veränderung in S'
         bewerte S' und berechne die Energiedifferenz ΔE
         wenn ΔE ≤ θ
            kopiere S' nach S
      wende ein Abkühlungsschema auf θ an.
      berechne r neu.
end
```

Eine Vereinfachung des SA Algorithmus ist das *threshold accepting* Verfahren (TA) [Dueck&Scheuer90]. TA akzeptiert jede Veränderung, die zu einem Zustand führt, der die Qualitätsdifferenz Θ zum besten bekannten Zustand $\boldsymbol{S}_{best}$ nicht überschreitet. Die Wahrscheinlichkeit, daß eine Aktion akzeptiert wird, ist daher:

$$P_{i,j}(T^{op},\Theta)=\begin{cases}0 & \Delta\ Q_{best,j}\geq\Theta\\ 1 & \Delta\ Q_{best,j}<\Theta\end{cases} \qquad (1\text{–}29)$$

$$\text{mit}\quad \Delta\ Q_{best,j}=Q(\boldsymbol{S}_{best})-Q(\boldsymbol{S}_j)$$

Die Qualitätsschwelle Θ startet mit einem hohen Wert und wird während der Optimierung abgesenkt, wenn die Veränderungen, in einer vorgegebenen Anzahl von Aktionen, keine Verbesserung des besten bekannten Individuums $\boldsymbol{S}_{best}$ finden.

Das TA erfordert daher kein angepaßtes Kühlschema, um zu vergleichbar guten Ergebnissen wie SA mit unendlich langsamer Abkühlung zu führen [Reinelt94].

1.4 Genetischer Algorithmus

```
Allgemeiner Genetischer Algorithmus:
begin
   Erzeuge m Anfangszustände im Pool  S1 ... Sm.
   wähle eine Anzahl n ≤ m
   iteriere bis das Abbruchkriterium erfüllt ist.
         Mutation)
            verändere einige Zustände im Pool mittels eines zweiten Algorithmus.
         Rekombination)
            erzeuge einen oder mehrere neue Zustände durch Rekombination
                  von 2 existierenden Zuständen.
         Selektion)
            entferne einige Zustände aus dem Pool.
   der beste Zustand S repräsentiert die Lösung.
end
```

Ein Genetischer Algorithmus (GA) [Goldberg89] ist ein Optimierungsverfahren, das die Darwinschen Prinzipien der Evolution und Selektion nutzt. Es unterscheidet sich von den vorher besprochenen Verfahren, indem es nicht eine einzelne Lösung $\boldsymbol{S}_i$ betrachtet, sondern eine große Anzahl unabhängiger $\boldsymbol{S}_i$, eine sogenannte Population POP. In der Terminologie der Genetischen Algorithmen sind die $\boldsymbol{S}_i$ die Individuen, die einzelnen Zustandsvariablen der Individuen sind die Gene. Alle zusammen bilden sie das Chromosom eines Individuums. POP(t) ist die Population der Individuen, die zu einem Zeitpunkt t vorhanden sind. Die Gene aller Individuen in der Population bilden zusammen den Genpool der Population.

$$\mathrm{POP}(t) = \{\boldsymbol{S}_1,\ldots,\boldsymbol{S}_n\}:\boldsymbol{S}_i\in\boldsymbol{S} \qquad (1\text{−}30)$$

Ein GA erfordert drei verschiedene Operatoren, die Mutation, Rekombination und Selektion.

Mutationsoperatoren wirken auf einzelne Individuen ein und verändern einen Teil ihrer Gene. Diese Operatoren brauchen die Qualität des Individuums nicht zu verbessern sondern nur zu verändern. Daher kann als Mutationsoperator auch ein anderer Optimierungsalgorithmus oder ein Produktionsalgorithmus verwendet werden.

Der Selektionsoperator wählt aus der Population POP(t) die Individuen aus, die in die Population des nächsten Zeitpunktes POP(t+1) übernommen werden. Damit die Population zum globalen Optimum konvergiert, muß die Selektion Individuen mit hoher Fitness bevorzugen [Michalewicz92]. Als Fitneßfunktion kann die Qualitätsfunktion der vorhergehenden Optimierungsverfahren benutzt werden.

Die Rekombination wird durch die Crossover−Operation realisiert.

1.4.1 Crossover Operation

Die Crossover Operation ist eine Anleihe der Informatik in der Biologie. Dort werden, während der Meiose Chromosomen durch ein Bruch−Fusions−Ereignis umorganisiert. Sie tauschen dabei zwischen parallel liegenden Chromosomen lange Stücke aus. Weder die Chromosomen noch die ausgetauschten Stücke brauchen die gleiche Funktion in beiden Ursprungschromosomen zu besitzen. Auch können die entstehenden Chromosomen defekt sein. Das entstehende Individuum ist dann möglicherweise nicht lebensfähig.

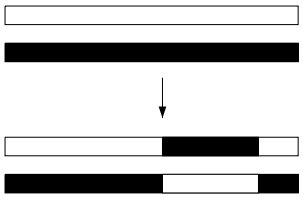


Abbildung 1−3 Klassisches Crossover

Im Algorithmus nehmen die Zustände S_i die Stelle der Chromosomen ein. Ein Crossover−Operator erzeugt aus mehreren »Eltern« einen neues Individuum, ein »Kind«. Dabei soll durch die Entnahme von Genmaterial bei den Elternzuständen ein Individuum mit einer Parameterkombination entstehen, die bisher nicht im Pool vorhanden ist. Obwohl beide Eltern wahrscheinlich noch nicht optimal sind, aber beide einen Teil der gesuchten Lösung enthalten, erhofft man sich von der Rekombination, daß ein neues Individuum entsteht, das besser als die Eltern ist, oder eines, das an eine bisher nicht untersuchte Stelle des Zustands−raumes projiziert wird und sich dort zu einem besseren Individuum entwickeln kann.

Da die Kinder aus dem Genmaterial der Eltern bestehen, befinden sich neue Individuen irgendwo auf dem Pfad $\mathrm{diff}(S_i \rightarrow S_j)$ durch den Lösungsraum, der die Eltern verbindet. Ein Crossover Operator kann daher nur Bereiche des Lösungsraumes erreichen, die sich "innerhalb" der Kugelwolke der Individuen in POP(t) befinden. Damit ein Algorithmus mit der Crossover Operation zum globalen S_{global} konvergiert, muß in der Anfangspopulation entweder schon jeder erforderliche Einzelparameter der globalen Lösung enthalten sein, oder es muß ein Mechanismus bestehen, der neue Gene in den Genpool einführt. Diese Aufgabe übernimmt die Mutation der Individuen. Außerdem muß die Mutationsrate hinreichend hoch sein, um eine vorzeitige Konvergenz durch genetische Verarmung zu verhindern.

1.4.2 Konfliktauflösung

Durch die zufällige, blinde Erzeugung von neuen Zuständen im Crossover können auch Zustände ent−stehen, die nicht alle Nebenbedingungen des Lösungsraumes erfüllen. Diese Zustände sind illegal und gehören nicht zur Lösungsmenge S^e sondern zu S^n. Die illegalen Zustände können auf zwei Arten behandelt werden [Michalewicz92, S. 312++].

1. Das illegale Individuum wird beibehalten, in der Erwartung, daß eine der nachfolgenden Veränderungen das Individuum zurück in den legalen Raum führt. Das Individuum wird gleichzeitig mit einer Strafe belegt, damit seine Qualität niedriger als die ähnlicher legaler Individuen ist. Dazu muß eine zweite Bewertungsfunktion für illegale Individuen erzeugt werden. Diese Bewertungsfunktion muß auf die Bewertungsfunktion für legale Individuen abgestimmt sein.

   Diese Behandlung hat weitreichende Konsequenzen für den Rest des Algorithmus. Zum Beispiel muß sichergestellt werden, daß nur das beste *legale* Individuum garantiert für die nächste Population bewahrt wird, da sonst eine Relationsfunktion für die Qualitätswerte der legalen und illegalen Individuen notwendig ist. Diese Ordnungsfunktion muß eine sinnvolle Reihenfolge zwischen schlechten legalen und guten ille−galen Individuen erzeugen, um zum Beispiel die Rangfolge für den nächsten Reproduktionszyklus fest−zulegen.

2. Das Individuum kann durch einen Reparaturalgorithmus in ein legales Individuum überführt werden. Die einfachste Implementierung ist, so lange einige illegale Elemente (Gene) zu entfernen, bis ein legales Individuum entstanden ist. Da das Problem von einem der zuletzt veränderten Elemente verursacht sein muß, kann man solange diese Veränderungen zurücknehmen bis das Individuum legal wird. Die neuen Elemente sind aber gleichzeitig auch die erwünschten Veränderungen. Daher möchte man nicht diese, sondern andere schon vorher vorhandene Elemente entfernen. Dazu müssen diese illegalen Elemente

effizient zu identifizieren sein. In der vorliegenden Implementierung wird dies durch die Kontrolle der Resourcen erreicht (siehe Kapitel 2.1.1).

Da nur legale Zustände erzeugt werden, ist nur eine Bewertungsfunktion notwendig.

Für die vorliegende Arbeit wurde das 2.te Verfahren ausgewählt, da es einfacher und schlüssiger zu implementieren ist.

Kapitel 2 Implementierung

Im folgenden Abschnitt wird die gewählte Implementierung vorgestellt. Dabei wird ein *bottom−up* Ansatz für die Erklärung der Besonderheiten der Implementierung gewählt.

2.1 Peaks

Die Koordinaten eines Peak sind die Frequenzen an denen er im Spektrum gefunden wurde. Zur besseren Vergleichbarkeit werden sie in *ppm* angegeben. Jeder Peak hat außerdem ein Volumen und einen Entartungsgrad.

Die Peaks der verschiedenen Experimente werden in ein einheitliches Koordinatensystem übertragen. Dazu werden alle Peaks auf 4 Dimensionen erweitert. Die Reihenfolge der Dimensionen ist [^{1}HN , ^{15}N ,CO, CA und CB]. CA und CB teilen sich eine Dimension, da sie in den Experimenten in der gleichen Dimension erscheinen und nur durch Interpretation getrennt werden können.

2.1.1 Peaks und Entartungsgrad

In den NMR−Experimenten für das *backbone* Assignment erzeugt jedes Spinsystem genau eine Gruppe von Peaks [Maxima] im Spektrum. Die in dieser Arbeit verwendeten Experimente erzeugen sogar nur einen Peak je Spinsystem. Für n−dimensionale Spektren ist es daher wahrscheinlich, daß jeder Peak an einer isolierten, nur ihm eigenen Position (Frequenz) im Spektrum erscheint.

Im Abhängigkeit von der Struktur des untersuchten Proteins (und anderer physikalischer Parameter wie der Temperatur, der Feldinhomogenität, pH ...) kann die Distanz zwischen den Frequenzen einzelner Aminosäuren trotzdem nahe Null betragen. In dieser Situation überlappen die Peaks der verschiedenen Aminosäuren soweit, daß man sie nicht mehr voneinander unterscheiden kann. Diese Peaks werden dann als ein Peak gepickt. Unter Umständen kann man diese Peaks anhand ihrer Größe (Ausdehnung) und ihrem Volumen (Intensität) als entartet identifizieren.

Für die Mustersuche wurde eine Implementierung gesucht, die eine doppelte Benutzung der nicht entarteten Peaks effizient verhindert. Dazu wurden in der Implementierung der Peaks jedem zwei Zähler zugeordnet. Der erste, der Entartungsgrad, wird einmal zu Anfang der Lebenszeit des Peak−Objekts auf die Anzahl der nicht entarteten Peaks gesetzt, die unter dem Signal an der gleichen Stelle verborgen sind. Dieser Zähler ist in der Standardeinstellung für jeden Peak gleich 1. Der zweite enthält die aktuelle Anzahl der *Besitzer* dieses Peaks. Zu Anfang ist sie 0. Jeder *Benutzer* des Peaks muß diesen *erwerben (*`aquire`*)*,

wenn er Besitzer des Peaks werden will. Ein Peak kann nur erworben werden, wenn sein zweiter Zähler niedriger als der erste ist. Andernfalls schlägt die `aquire` Aktion fehl und der Benutzer wird nicht zum Besitzer. Ein Besitzer muß die erworbenen Peaks später wieder loslassen (`release`) damit auch andere Benutzer des Peaks die doppelte Benutzung des Peaks innerhalb ihres Kontextes überprüfen können. Der zweite Zähler wird in beiden Methoden entsprechend aktualisiert.

Potentielle Besitzer der Peaks sind die Protospinsysteme, die durch Erwerben all ihrer Peaks effizient testen können, ob sie Peaks doppelt erwerben oder ob sie Peaks verwenden, die gleichzeitig von anderen Protospinsystemen beansprucht werden.

Als Konsequenz dieser Implementierung muß der Entartungsgrad für die entarteten Peaks manuell gesetzt werden. Die Bestimmung des Entartungsgrades ist nicht Teil der Optimierungsalgorithmen.

Peaks werden im folgenden auch als Resourcen bezeichnet, da sie als Objekte in einer abzählbaren Menge vorhanden sind und von den Protospinsystemen konkurrierend beansprucht werden.

Definition: *Resource*: Ein Objekt das nur in begrenzter Anzahl zu Verfügung steht. Resourcen können nur entsprechend ihrer Anzahl erworben werden. Resourcen sind Peaks und alle daraus gebildeten Objekte wie Knoten, Protospinsysteme usw.

Definition: *Benutzer*: Ein Objekt das eine Resource benutzt und besitzen kann. Ein Benutzer wird zum Besitzer wenn es eine Resource erwirbt. Knoten, Protospinsysteme, Zustandsobjekte usw.

Definition: *Besitzer*: Ein Objekt das eine Resource besitzt. Knoten, Protospinsysteme usw.

Definition: *Buchhaltungskontext*: Benutzer besitzen ihre Resourcen in einem Buchhaltungskontext. Die Resourcen werden innerhalb eines Buchhaltungskontext abgerechnet. Jedes Zustandsobjekt $\boldsymbol{S}_i$ hat seinen eigenen Buchhaltungskontext.

2.1.2 Das Peak Netz

In den verschiedenen Stufen des Assignment Prozesses wird zu einem Peak ein Nachbar entlang einer oder mehrerer Dimensionen gesucht. Außerdem wird an verschiedenen Stellen die Topologie der Nachbarschaft bewertet. Daher sollen die Abstände nicht permanent neu berechnet, sondern einmal vorberechnet und dann nur nachgeschlagen werden.

2.1.2.1 Implementierung

Eine mögliche Implementierung ist eine Matrix, die für jedes Peakpaar die Frequenzdifferenz der Peaks speichert. Diese Implementierung verbraucht sehr viel Speicher. Für 980 Peaks mit 4 Dimensionen müssen 980 * 980 * 4 = 3841600 Distanzen gespeichert werden. Bei 8 Byte pro Distanz wird so 29,3 MB belegt. Außerdem muß man die Matrix entlang einer Dimension vollständig durchsuchen um alle Nachbarn eines Peaks oder den Nachbarn mit der niedrigsten Distanz zu finden.

Statt als Distanz kann man die Nachbarschaft auch als Relation zwischen zwei Peaks beschreiben. Die Relation ist dann die *Kante* in einem ungerichteten Graphen. Die *Knoten* repräsentieren die Peaks. Sie enthalten eine kleine Tabelle, die alle Kanten zu den Nachbarn dieses Peaks enthalten. Jede Kante verbindet zwei Peaks entlang einer Dimension. (Kante:=Peak+Dimension <−−Q−−> Peak+Dimension) . Jeder Kante ist eine Qualität zugeordnet, die zu Anfang als Funktion der Frequenzdifferenz und des Isotops berechnet wird. Ein Peak ist dabei nur mit den Peaks verbunden, die mit ihm direkt benachbart sind. Die Suche nach

den Nachbarn eines Peaks wird dadurch stark beschleunigt, da nicht mehr alle, sondern nur noch benachbarte Peaks durchsucht werden. Für einen HNCO Peak in "Trigger" müssen daher statt ~870 nur noch durchschnittlich 10 Peaks untersucht werden.

Diese Information wird als Datenstruktur abgelegt, die die Nachbarschaftsbeziehung aller Peaks als ungerichteten Graphen speichert. Dies ist das sogenannte *PeakNet*.

Definition: *Basisknoten(BP)*: Ein Basisknoten (BP) ist ein Knoten im PeakNet, der einen Peak des Basisexperimentset enthält. Ein nicht−BP ist ein Peak, der nicht dem Basisexperiment angehört. Einzelne Basispeaks werden auch mit ihrer Identifikationsnummer und dem Prefix BP benannt. Beispiel: BP101 ist der Basispeak 101.

Für das Bax Experimentset ist dies das HNCO Experiment, da es am empfindlichsten ist und die beste Auflösung hat und für jede Aminosäure nur einen Peak erzeugt.

2.1.2.2 Topologien

Um die verschiedenen Programme des Assignments weiter zu beschleunigen, werden jeweils nur die für diesen Schritt benötigten Kanten angelegt. Dadurch ergeben sich unterschiedliche Topologien für das PeakNet. Die folgenden Definitionen beschreiben die Topologien für das Bax Experimentset.

- Zuordnung zum Basisknoten:

  Jeder Basisknoten repräsentiert einen Peak im HNCO Spektrum. Bei der Zuordnung zum BP werden nur Kanten zwischen HNCO und nicht−HNCO Peaks entlang der ^{1}HN und der ^{15}N Dimensionen angelegt. Wenn zwischen dem Basispeak und dem zweiten Peak nur eine Kante existiert oder eine der beide Kanten eine Qualität kleiner 0,5 hat, werden beide Kanten entfernt.

- intranodale Verknüpfung (AS−Protospinsystem) :

  Zusätzlich zum Basisknoten werden alle Kanten entlang der ^{13}C Dimension zu Peaks, die zum gleichen Basisknoten gehören, angelegt.

- internodale Verknüpfung (Link− Protospinsysteme):

  Zusätzlich zur intranodalen Verknüpfung werden die ^{13}C Verknüpfungen zwischen allen nicht−HNCO Peaks angelegt.

2.1.2.3 Kantenqualität

Für jeden Peak wird im PeakNet ein Knoten angelegt. Die automatische Verknüpfung verbindet alle Peaks, die innerhalb eines maximal Abstandes mit dem aktuellen Startpeak liegen. Die Qualität der Kante wird dann mit der Abstandsfunktion Gl. 2−1 initialisiert. Kanten mit einer Qualität unter 0.5 werden nicht angelegt.

Alle Verknüpfungsalgorithmen verwenden die lineare Abstandsfunktion Gl. 2−1.

$$\begin{aligned} q(ppm_1, ppm_2, \Delta_{max}) &= \frac{\Delta_{max} - |ppm_1 - ppm_2|}{\Delta_{max}} \\ Q_{dist,lin}(ppm_1, ppm_2, \Delta_{max}) &= \max(q(ppm_1, ppm_2, \Delta_{max}), 0) \end{aligned} \qquad (2-1)$$

Für zweidimensionale Vergleiche wird die Gl.B−1 im Anhang verwendet.

2.2 Protospinsysteme

Protospinsysteme sind die Grundbausteine des RANDOM Algorithmus.

Definition: *Protospinsystem*: Protospinsysteme sind Gruppen von Peaks, die zusammen als mögliches Spinsystem gedeutet werden. Jeder Peak im Protospinsystem ist mindestens einer Funktion (Bedeutung) zugeordnet. Ein Protospinsystem ist die Menge aller (Peak,Bedeutungs)−Paare die ihm zugeordnet sind. In einem Protospinsystem ist jeder Bedeutung nur ein Peak zugeordnet (n:1−Beziehung). Ein Protospinsystem ist Besitzer seiner Resourcen und daher selbst eine Resource.

Diese Definition des Protospinsystems erweitert die Definition des Protospinsystems im manuellen Assignment aus Kapitel 1 um die Funktion als Resource und Besitzer und um die 1:n Beziehung zwischen Peak und Bedeutung. Die Regeln, nach denen die Protospinsysteme gebildet werden, hängen von dem NMR Experimentset ab. Die Strukturbedingungen für ein Protospinsystem werden auch als (Struktur−)Muster bezeichnet.

Definition: *gleiche Frequenz*: Wenn zwei Frequenzen mit Gl. 2−1 einen Qualitätswert größer 0 ergeben, sind sie *frequenzgleich*. Der Toleranzfaktor der Frequenz Δ hängt von der Funktion der zugrunde liegenden Atome ab. Frequenzpaare werden mit Gl. B−2 getestet. Die Standardwerte für Δ sind in Tabelle 2−1 aufgeführt.

Isotop / Dimension	^{1}HN	^{15}N	**CO**	**CA/CB**
Δ [ppm]	0.05	0.55	0.55	1

Tabelle 2−1 Standardwerte für Δ in ppm

2.2.1 Protospinsystempool

Die Protospinsysteme bilden zusammen den Protospinsystempool. Der Pool kann, je nach angewandten Filtern, für das gleiche Problem unterschiedlich zusammengesetzt sein.

Kurzbezeichnung	**Beschreibung**
oPa	Ein Pool, der nur aus **p**erfekten, d.h. theoretischen Protospinsystemen besteht. Jedes Protospinsystem ist **a**llen Aminosäuren zugeordnet. Die Aminosäureerkennung beschränkt die Positionen nicht.
oP	Ein Pool, der nur aus **p**erfekten, d.h. theoretischen Protospinsystemen besteht. Die Protospinsysteme sind nur einzelnen Aminosäuren zugeordnet. Die Aminosäureerkennung beschränkt die zulässigen Positionen.
oMa	Ein Pool, der auch Protospinsysteme mit Defekten enthält. Jedes Protospinsystem ist **a**llen Aminosäuren zugeordnet. Die Aminosäureerkennung beschränkt die Positionen nicht.
oM	Ein Pool, der auch Protospinsysteme mit Defekten enthält. Die Protospinsysteme sind nur einzelnen Aminosäuren zugeordnet. Die Aminosäureerkennung beschränkt die zulässigen Positionen.

Tabelle 2−2 Kurzbezeichnungen der Protospinsystempoolvarianten, wie sie in den Kapiteln 3 bis 5 benutzt wird.

2.2.2 Relationen zwischen Protospinsystemen

In den folgenden Unterkapiteln werden verschiedene Relationsfunktionen und −operationen zwischen den Protospinsystemen verwendet. Diese werden hier eingeführt und definiert.

Definition: Die *Differenz D zweier Protospinsysteme* a, b ist die Menge aller (Peak,Bedeutungs)−Paare, die nur in einem der beiden Protospinsysteme vorkommen.

Die Differenz zweier Protospinsysteme kann ein Protospinsystem sein, wenn es die Bedingungen eines Protospinsystems erfüllt. Die meisten Differenzen enthalten aber mehrere (Peak,Bedeutungs)−Paare zur gleichen Bedeutung.

Definition: *resourcengleich*: Zwei Protospinsysteme a und b sind resourcengleich, wenn sie die gleichen Resourcen verwenden. Die Bedeutung der Resourcen in a und b kann unterschiedlich sein.

Definition: *funktional gleich*: Zwei Protospinsysteme a und b sind funktional gleich, wenn sie zu den gleichen backbone Frequenzen führen.

Definition: *subfunktional*: Protospinsystem a ist subfunktional zu b, wenn a weniger backbone Frequenzen als b erklärt und alle Frequenzen aus a in b vorhanden sind. b ist dann *superfunktional* zu a. Die Frequenzen werden mit einer toleranten Vergleichsfunktion verglichen (Gl.B−1).

Definition: *strukturgleich*: Zwei Protospinsysteme a und b sind strukturgleich, wenn sie resourcengleich und funktional gleich sind und identische Peaks die gleiche Bedeutung haben. Die Peaks sind dann gleich verknüpft. Strukturgleiche Protospinsysteme sind gleich, wenn sie die gleiche Bewertungsfunktion durchlaufen haben.

Definition: S*ubstruktur, Superstruktur*: Protospinsystem a ist eine Substruktur (subCX) von b, wenn man a durch Entfernen von Peaks aus b erzeugen kann. b ist dann eine Superstruktur von a.

Definition: *konkurrenzneutral*: Ein Peak ist konkurrenzneutral, wenn er nur zu einem Basispeak gehört. Die Substrukturbeziehung ist konkurrenzneutral, wenn die Resourcen der Differenz der Protospinsysteme konkurrenzneutral sind. Ein Protospinsystem ist konkurrenzneutral, wenn es aus konkurrenzneutralen Resourcen besteht. Zwei Protospinsysteme sind zueinander konkurrenzneutral, wenn sie gleichzeitig erworben werden können.

2.3 AS−Protospinsysteme

Definition: *AS−Protospinsystem (***ACX***)*: Ein Protospinsystem, das alle Resourcen eines, ein bis zwei Aminosäuren langen, backbone− und Seitenketten−Abschnitts aufgrund eines einfachen Ordnungskriteriums zusammenfaßt. Es muß den AS−Strukturtest des verwendeten Experimentsystems bestehen. Alle AS−Protospinsysteme eines Proteins können gleichzeitig erworben werden.

Das Ordnungskriterium im Bax Experimentset ist das NH,N−Frequenzpaar. Die NH,N−Frequenzen jedes Peaks eines Protospinsystems muß mit dem NH,N−Frequenzpaar des HNCO Peaks übereinstimmen. Dieser Peak ist der Basispeak des Protospinsystems.

2.3.1 Die AS−Protospinsystemmuster für das Bax Experimentset

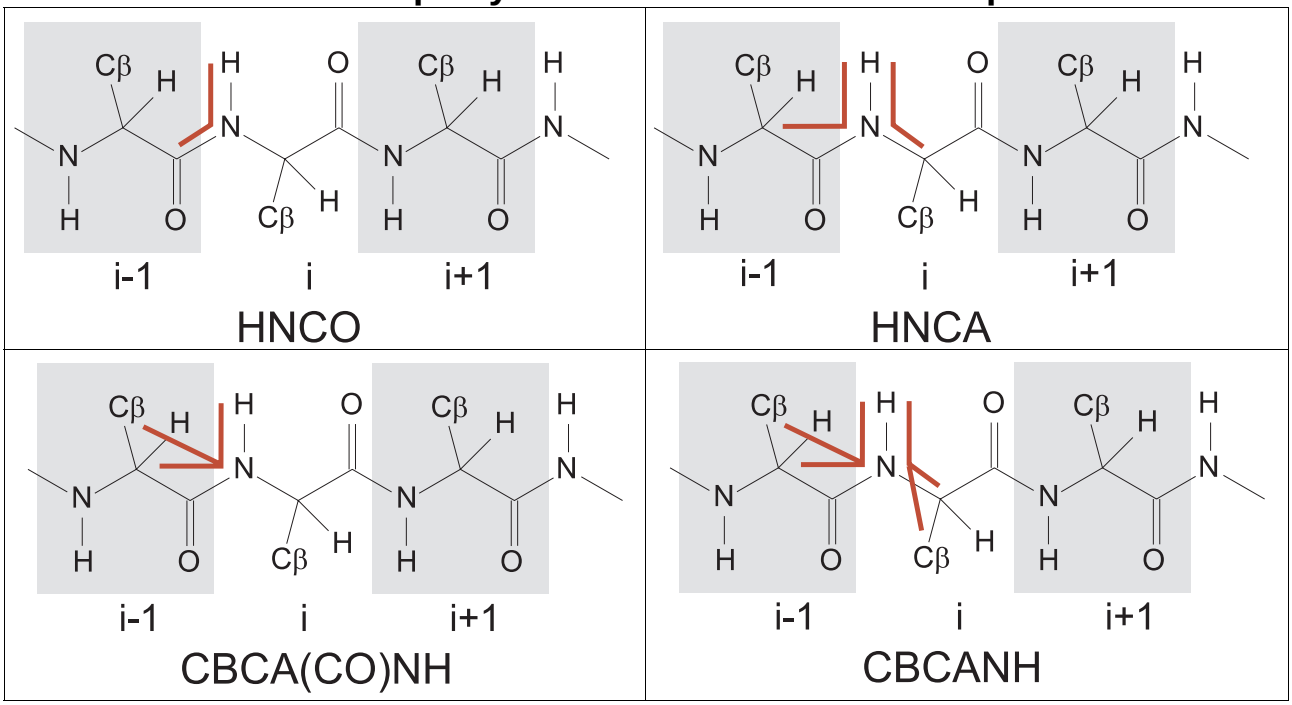


Abbildung 2−1 Ausschnitt des backbones mit den ^{1}J−Magnetisierungstransferpfaden, die für die Zuordnungsstrategie verwendet werden.

Das Bax Experimentset (siehe auch Kap. 1.2.1) besteht aus 4 Experimenten vom Typ HNCO, HNCA, CBCANH und CBCA(CO)NH. Ein Protospinsystem enthält Peaks mit Freuqenzen aus jeweils zwei benachbarte Aminosäuren ((i−1) und i). Alle Peaks eines Protospinsystems enthalten die ^{1}HN und ^{15}N Frequenz der Aminosäure i. Die Peaks mit übereinstimmenden ^{13}C Frequenzen sind in Tabelle 2−3 aufgeführt.

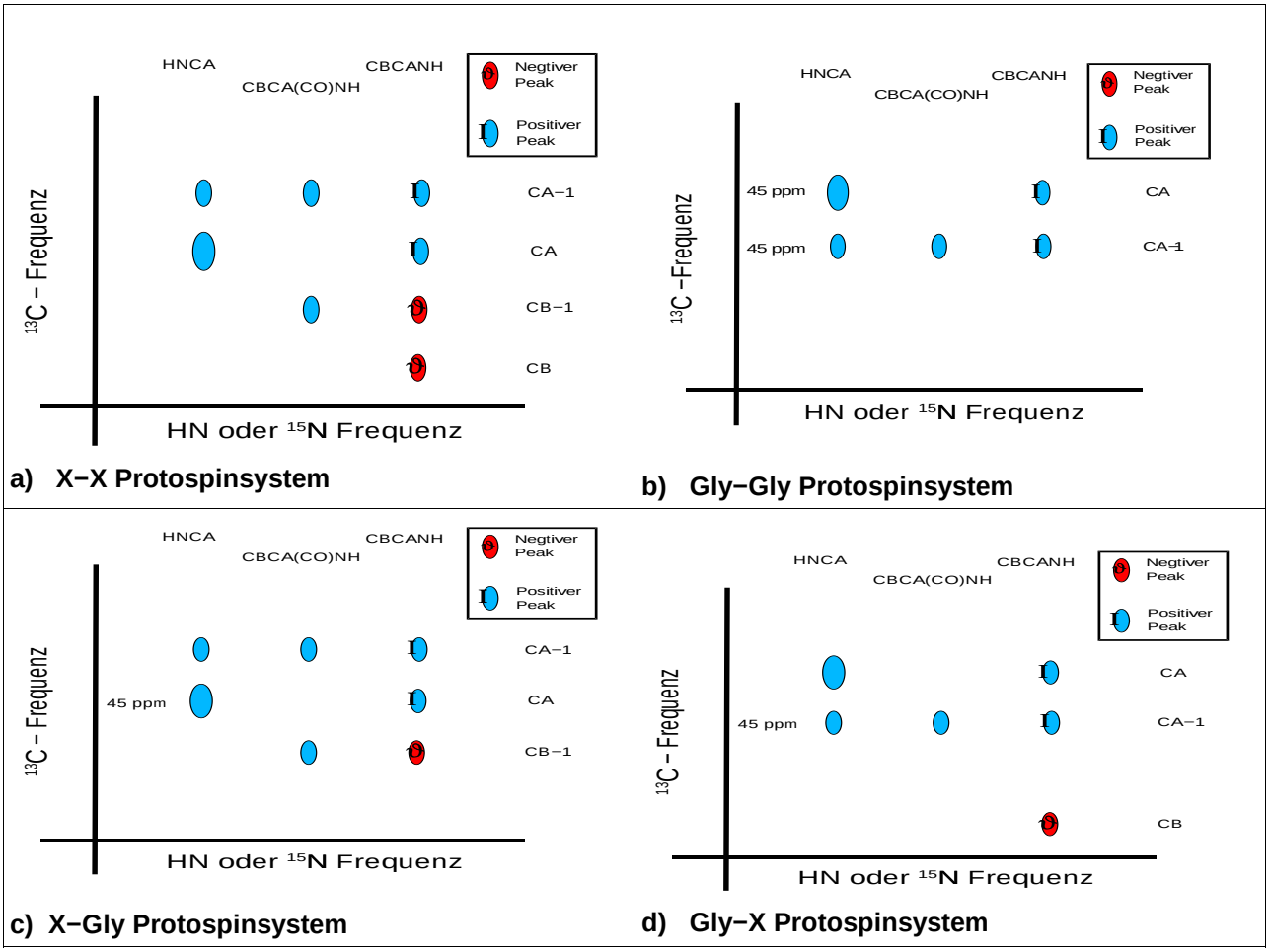


Abbildung 2−2 Die zu erwartenden Peakmuster für die in Tabelle 2−3 aufgeführten Protospinsystemtypen mit den Experimenten des Bax Experimentsets. Die Protospinsysteme wurden in der HN− oder N15−Dimension gedehnt, da sich sonst alle Peaks mit der gleichen Frequenz überlagern würden.

Peak	H_i	N_i	CO_{i-1}	CA_{i-1}	CB_{i-1}	CA_i	CB_i	Volumen
HNCO	x	x	x					
HNCA CA−1	x	x		x				<=
HNCA CA	x	x				x		>=
CBCA(CO)NH CA−1	x	x		x				
CBCA(CO)NH CB−1	x	x			x			
CBCANH CA−1	x	x		x				+
CBCANH CB−1	x	x			x			−
CBCANH CA	x	x				x		+
CBCANH CB	x	x					x	−

Tabelle 2−3 Die Peaks im Bax Experimentset. Frequenzen: x kennzeichnet vorhandene Frequenzen. Volumen: + und − bezeichnen das relative Vorzeichen des Peaks. >= und <= bezeichnen den größeren und kleineren HNCA Peak.

Außerdem gelten für die Peaks die folgenden Nebenbedingungen:

- Jedes Aminosäureprotospinsystem (kurz: ACX) enthält genau einen HNCO Peak.
- CBCA(CO)NH CB−1, CBCANH CB und CBCANH CB−1 haben ein negatives Volumen.
- CBCA(CO)NH CA−1, CBCANH CA und CBCANH CA−1 haben ein positives Volumen.
- Alle CA Frequenzen entsprechen der Funktion [Abb: "CA Freq"im Anh. C.1].
- Alle CB Frequenzen entsprechen der Funktion [Abb: "CB Freq weit" im Anh. C.1].
- HNCA CA−1 hat ein kleineres Volumen als HNCA CA. Diese Bedingung wird nicht immer eingehalten (weiche Bedingung).

Für die Implementierung wurden noch zusätzliche Bedingungen eingeführt, die den Lösungsraum begrenzen und Lösungen ausschließen, die *suboptimal* sind.

- Mindestens zwei der vier CA/CB−Frequenzen müssen im Protospinsystem enthalten sein.
- Aminosäuretyp: Wenn auf der i oder i−1 Seite eine Frequenz vorhanden ist, müssen die Frequenzen mindestens einem Aminosäuretyp entsprechen [Abbildungen: "AS Typen" im Anh. C.2].
- Nutzungsgrad *n*: Die Protospinsysteme müssen mindestens *n*% der vorhandenen Peaks oder, falls, durch Überlappung oder Fehlpeaks, mehr als *maxPeaksForPattern* Peaks vorhanden sind, *n*% von *max−PeaksForPattern* enthalten. Typische Werte für *n* sind 75−95%, je nach Qualität und Überlagerung der Spektren.

Aminosäuren können aufgrund der Strukturunterschiede in mehrere Gruppen eingeteilt werden. Die größte Gruppe (X) bilden die Aminosäuren, die im Protein sowohl ein CA−, ein CB− und ein NH−Atom enthalten. Die zweite Gruppe (G) besteht aus der Amionsäure Glycin, das kein CB−Atom enthält. Die dritte Gruppe (P) enthält nur Prolin, das kein NH Proton im Protein enthält.

Da die Protospinsysteme des Bax Experimentsets die Frequenzen zweier benachbarter Aminosäuren enthalten, müssen die Protospinsystemmuster die verschiedenen Nachbarschaftspaare berücksichtigen.

Das einfachste Nachbarschaftspaar ist XX, d.h. zwei Amionsäuren der Gruppe X. Diese Nachbarschaft wird vom Standardprotospinsystemmuster berücksichtigt.

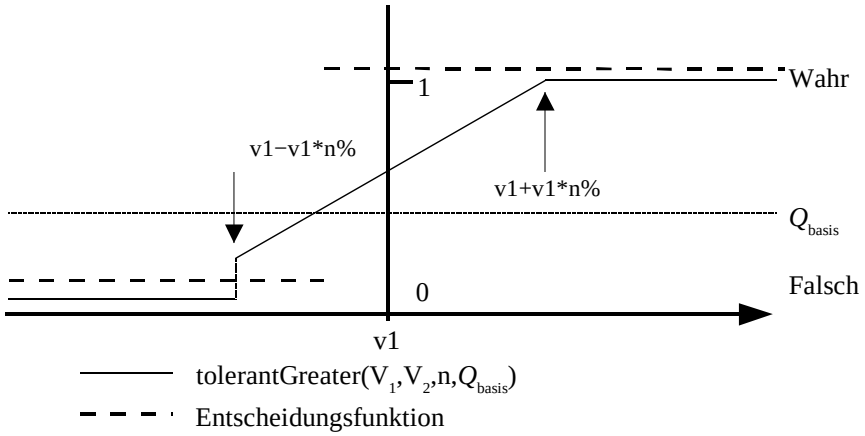
Die Glycin Protospinsysteme enthalten kein CB Atom und daher auch keine Peaks mit CB Frequenzen. Da die Protospinsystemmuster mit Glycin sich je nach der Position des Glycins unterscheiden, entstehen dadurch drei Arten von Glycin−Mustern.

Aminosäurepaare mit einem Prolin in i−Position (XP) erzeugen keine Protospinsysteme, da ihnen das ^{1}HN Proton fehlt. Aminosäurepaare mit einem Prolin in i−1 Position (PX) können Protospinsysteme erzeugen. Sie werden wie normale XX Protospinsysteme behandelt.

Abkürzung	XX	PX	GX	XG	GG
Nachbarschaft	X(i−1)−X(i)	Pro(i−1)−X(i)	Gly(i−1)−X(i)	X(i−1)−Gly(i)	Gly(i−1)−Gly(i)
maxPeaksForPattern	9	9	7	8	6

Tabelle 2−4 Benachbarte Aminosäure−Protospinsystemtypen, ihre Abkürzungen und die maximale Anzahl der Peaks für ein vollständiges Protospinsystem. X(n) bezeichnet eine beliebige Aminosäure außer Glycin und Prolin in Position i. maxPeaksForPattern *ist die Anzahl der Peaks, die für ein perfektes Spinsystem zu erwarten ist.* X *steht für eine beliebige Aminosäure, andere Buchstaben entsprechen den Aminosäuren gemäß dem 1−Buchstabenkode.*

2.3.2 Toleranter Vergleich des Volumens zweier Peaks



2Die Bewertung einer Zuweisung erfordert den Vergleich des Volumenverhältnisses zweier Peaks im glei−chen Spektrum. Das Volumen des einen Peaks ist theoretisch zwar größer als das des anderen, doch in der Praxis wird dieses Verhältnis, auf Grund der Fehlerbreite der Messung, nicht eingehalten. Statt dessen wird im manuellen Assignment ein toleranter Vergleich verwendet, der das Volumenverhältnis noch akzeptiert, wenn der theoretisch kleinere Peak den theoretisch größeren nur geringfügig überragt Die Funktion tole−rantGreater(V_1,V_2,n,Q_{basis}) implementiert diesen Vergleich zwischen den Volumen zweier Peaks.

Die Bedingung ist erfüllt, wenn das Volumen des zweiten Peaks (V_2) entweder kleiner oder nicht mehr als n% des Volumens des ersten Peaks größer als das Volumen des ersten Peaks (V_1) ist.

Im Bereich von $V_1 + n\%(V_1)$ bis $V_1 - n\%(V_1)$ steigt die Qualität linear an.

n ist der Prozentsatz des Volumens V_1 im Bereich [0 .. 1].

$$\text{tolerantGreater}(V_{1,}\, V_{2,}\, n, Q_{basis}) := \begin{cases} \text{wenn} \quad V_1 + d \leqslant V_2 & : \quad 0 \\ \text{wenn} \quad V_1 - d \geqslant V_2 & : \quad 1 \\ \text{dazwischen} & : \quad \dfrac{Q_{basis} - 1}{-2d} * (V_1 - V_2 - d) + Q_{basis} \end{cases} \qquad (2\text{−}2)$$

mit

$$\text{n}\,\%(V_1) := d := V_1 * n$$

2.3.3 Implementierung der Muster

Die Bedingungen für die Protospinsysteme wurden als Zustandsmaschinen implementiert, die die Protospinsysteme sowohl erzeugen wie auch testen können. Um schnell und ohne Neukompilieren der Programme unterschiedliche Muster testen zu können, wurde eine Klassenbibliothek geschaffen, die einen Regelparser und einen Bedingungsprozessor für eine problemspezifische Programmiersprache enthält. Diese Programmiersprache implementiert die Einzelbedingungen des Protospinsystemmusters. Die Bedingungen können zu Gruppen zusammengefaßt werden, die zusammen als Einzelbedingung in einer weiteren Gruppe fungieren können.

Jeder Aminosäure−Protospinsystemtyp wird von einer anderen Zustandsmaschine implementiert. Der Quellcode der Bedingungen des XX−Musters ist im Anhang D.1 dargestellt.

Da die HNCA Volumen−Bedingung nicht immer eingehalten wird (weiche Bedingung), wurde sie nicht in das Suchmuster aufgenommen, sondern wird erst in der Bewertung der Protospinsysteme berücksichtigt.

Das XX−Muster findet auch die Protospinsysteme der Glycin−Aminosäuren. Wenn die Experimente sehr unvollständige Spinsysteme enthalten, muß das XX−Muster so tolerant konfiguriert werden, daß auch Gly−Protospinsysteme als XX−Protospinsystem akzeptiert werden. Daher werden, auf jeden Basisknoten der mindestens einen Glycin Peak enthält, der der Funktion "istEinGly" im Anhang C.1 entspricht, die Protospinsystemmuster in D−2 bis D−4 angewendet, um den Typ des Knotens zu bestimmen.

Die Gly−Muster sind auf die Besonderheiten der Gly−Protospinsysteme zugeschnitten. Sie enthalten einen Frequenztest für den Gly−Frequenzbereich und berücksichtigen nur die Peaks, die in einem Gly−Protospinsystem enthalten sein können.

2.3.4 Implementierung der Suche

Auf jeden Basisknoten (HNCO Peak) wird das XX−Muster angewandt, bis der gesamte Lösungsraum des Knotens durchsucht ist. Die Protospinsysteme werden in einer Liste je BP gesammelt und die Liste mit verschiedenen Filtern behandelt.

2.3.5 Filter

Jeder Haltepunkt der Zustandsmaschine ergibt eine Lösung der Mustersuche. Da die Zustandsmaschine jeweils nur eine Lösung auf einmal betrachtet, kann sie prinzipbedingt keine Optimierungen aufgrund von Beziehungen zwischen den Protospinsystemen (Lösungen) vornehmen. Die Liste der Protospinsysteme wird daher mit einer Serie von Filterregeln nachbearbeitet. Die Filter nehmen dabei den Teil der nachfolgenden Optimierung vorweg, für den man das Ergebnis vorhersagen kann. Diese Filter finden Aminosäureprotospinsystemgruppen, die die Menge der Aminosäureprotospinsysteme (ACX) nur vergrößern, ohne neue Informationen zur Lösungsmenge der Optimierung beizutragen, d.h. Protospinsysteme, die während der Optimierung durch andere, höher bewertete Protospinsysteme verdeckt werden.

Die folgenden Filterregeln wurden implementiert:

a. Entferne doppelte ACX aus der Liste.

b. Entferne alle suboptimalen Interpretationsvarianten:

- bilde eine Gruppe aller strukturell gleichen ACX, die auch funktional gleiche oder substrukturelle ACX zum gleichen Super−ACX sind.
- bestimme den strukturellen Super−ACX der Gruppe.
- lösche alle anderen Gruppenmitglieder aus der Liste.

c. Entferne die ACX, die durch einen anderen ACX mit der gleichen Funktion ausgetauscht werden können, ohne das Optimierungsergebnis zu verändern:
   - bilde die Gruppe aller resourcengleichen CX, die nicht strukturgleich, aber funktional gleich sind.
   - suche den CX mit dem maximalen Qualitätswert der Gruppe.
   - lösche alle anderen Gruppenmitglieder aus der Liste.

d. Entferne ACX, die während der Optimierung durch einen äquivalenten ACX mit mehr Resourcen ersetzt werden:

   für jede strukturelle Subprotospinsystembeziehung mit den Protospinsystemen(subCX, superCX):
   - wenn die strukturelle Subprotospinsystembeziehung auf (einer) konkurrenzneutralen Resource(n) beruht und subCX auch funktionaler Subcontext von superCX ist, dann entferne subCX.

2.3.6 Suchmuster für mögliche Glycin Spinsysteme

Da Glycine kein CB Atom enthalten und die Frequenz ihres CA Peaks isoliert von den CA Frequenzen anderer Aminosäuren ist, kann man ihre Spinsysteme meistens gut von den Spinsystemen anderer Aminosäuren unterscheiden. Die Protospinsystemerkennung mittels des XX−Musters kann diese Protospinsysteme finden, obwohl sie nicht auf diese Eigenschaften speziell abgestimmt ist. Wenn die Qualität der Spinsysteme schlecht ist, kann man die Glycin Protospinsysteme gezielt mittels der GX− oder XG−Muster suchen.

Für diese Suche wird zunächst jeder HNCO Knoten, der mit einem Peak im Bereich der Bewertungsfunktion (~45 ppm) verknüpft ist, gesucht. Danach werden nur auf diese Knoten die Glycin−Muster GX, XG oder GG angewandt.

2.3.7 Konkurrierende Peakgruppen

In realen Spektren führen überlappende Spinsysteme zu mehrdeutigen Situationen. Eine reine Maximum−Strategie, die für jeden Basisknoten nur die maximal aufgefüllten ACX akzeptiert und alle Substruktur ACX entfernt, führt zum falschen Ergebnis, da dann die ACX der gesamten Gruppe nicht mehr gleichzeitig erworben werden können und so die Gruppenqualität nicht maximal ist. Die ACX Listen enthalten dann, je Basisknoten, nur noch die ACX, die auch Peaks beanspruchen, die eigentlich zu einem der anderen Spinsystem gehören. Ein ACX Gruppenfilter muß daher die Konkurrenzsituation der Spinsystemgruppe, d.h. die Spinsysteme die durch Überlappung zusammengefaßt werden, berücksichtigen.

Der Gruppenfilter betrachtet dazu die Spinsysteme aller konkurrierenden Basispeaks als eine Gruppe und versucht die Gruppe statt der einzelnen Spinsysteme zu optimieren. Dabei werden auch die Konkurrenten der Konkurrenten in die Gruppe der Konkurrenten eines Basispeaks aufgenommen.

Damit der Algorithmus eine Spinsystemgruppe auflösen kann, müssen alle Spinsysteme ungefähr gleich viele Peaks zur Gruppe beitragen. Außerdem darf die Gruppe keine überschüssigen (zusätzlichen) falschen Peaks enthalten, die die Mustersuche in die Irre führen. Die Gruppentrennung kann entweder mittels des automatischen Gruppenfilters oder manuell durchgeführt werden. Die Gruppen werden mit einem zweiten Programm gesucht.

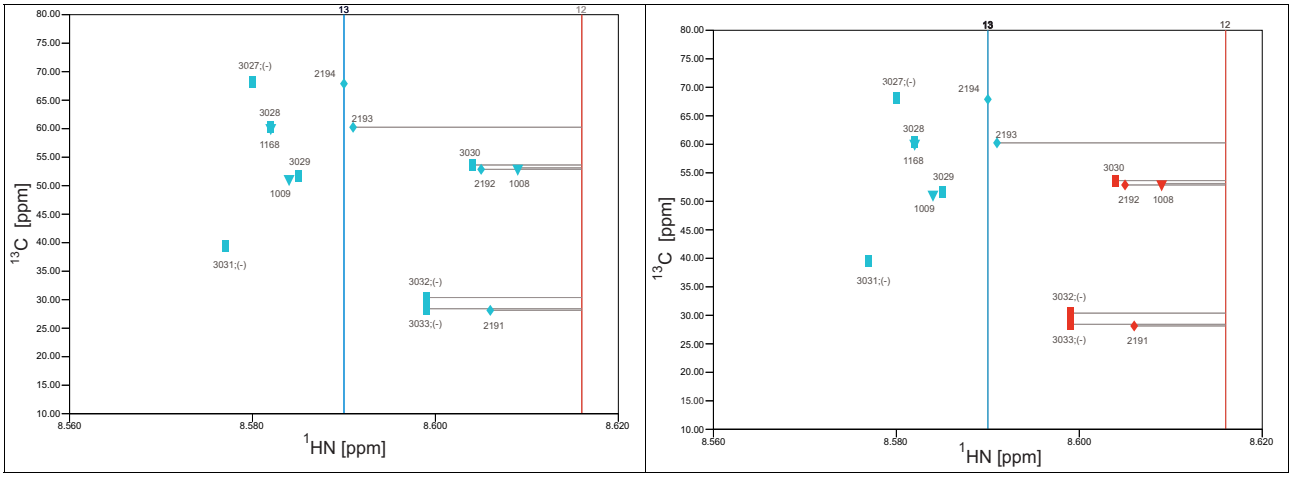


Abbildung 2−3 Separation der überlappenden Spinsysteme P12 und P13. Projektion der Peaks in die [HN,CACB]−Ebene, die im PeakNet mit den Basispeaks 12 und 13 verknüpft sind. Links: vor der Trennung. Rechts: Nach der Trennung.

Ein Beispiel für die automatische Separation, ist die erfolgreiche Trennung der Gruppe P13 + P12 in C21W/CAM (Abbildung 2−3). Ohne Trennung wurden für P13 und P12 je ein maximaler ACX im Cache abgelegt, von denen jeweils nur einer gleichzeitig erworben werden kann, da einige Peaks in beiden ACX enthalten sind. Durch die Trennung wurden ursprünglich suboptimale ACX, die weniger Peaks enthalten, wieder in die Liste aufgenommen, und die maximalen ACX aus den Listen entfernt. Die maximale Summe der Qualitätswerte der ACX aus beiden BP ist jetzt höher als vor der Trennung.

Die Ziffern neben den Peaks sind die Identifikationsnummern (ID) der Peaks. Ein Minuszeichen in Klammern hinter der ID bezeichnet ein negatives Vorzeichen des Peakvolumens. Die Form des Peaksymbols bezeichnet das Experiment aus dem der Peak stammt. Rechtecke stammen aus dem CBCANH, Romben aus dem CBCA(CO)NH und Dreiecke aus dem HNCA. Die HN Frequenz des HNCOs wird als senkrechte Linie markiert. Peaks, die nur dem Basispeak BP13 zugeordnet wurden, sind blau markiert, die nur zu BP12 gehören, sind rot.

In der linken Abbildung sind alle Peaks, die BP12 zugeordnet werden können, auch in der Entscheidungsmenge für BP13 enthalten und daher blau. Die Peaks auf der rechten Seite der gleichen Abbildung (von 8.595 bis 8.62ppm) gehören aber auch zu BP12 und sind daher mit der HN−Linie des BP12 verbunden. In der rechten Abbildung wurden die Spinsysteme für BP12 und BP13 getrennt, daher sind die Peaks für BP12 rot, da sie nicht zu BP13 gehören.

Das Protospinsystem um BP13 wurde als Thr79,Asp80 identifiziert. BP12 enthält das Protospinsystem für Glu114,Lys115.

2.4 Manuelle Bearbeitung

Für extrem überlagerte Spinsystemgruppen reichen die bisher besprochenen (automatischen) Filterregeln nicht aus, um die Anzahl der resultierenden ACX auf eine berechenbare Menge zu begrenzen. Daher wurde eine grafische Oberfläche entworfen, mit der man die automatisch erstellten ACX−Listen mittels manuell erstellter Strukturbedingungen weiter reduzieren kann. Diese Implementierung ermöglicht eine iterative Verfeinerung der Protospinsystempools. Dieser Filter wird von den sogenannten knotenlokalen Bedingungen implementiert.

2.4.1 Knotenlokale Bedingungen

Knotenlokale Bedingungen sind Bedingungen, die für jeden Basisknoten speziell angepaßt werden und nur auf die AS−Protospinsysteme dieses Knotens angewendet werden. Für jeden Basisknoten wird eine Liste von Bedingungen definiert, die sich aus den Bedingungstypen in Tabelle 2−5 zusammensetzen. Jede Bedingung kann auch in ihrer negierten Form verwendet werden. Außerdem kann eine Bedingung aktiviert oder deaktiviert sein. Die AS−Protospinsysteme, die allen aktiven Bedingungen gleichzeitig genügen, bilden dann die reduzierte ACX−Liste für diesen Basisknoten.

Typ	**Parameter**	**Beschreibung**
containsAll	Liste von (Name,Peak)−Paaren	Sind alle beschriebenen Bedeutungen enthalten?
maybeContains	−"−	Sind entweder alle beschriebenen Bedeutungen enthalten oder keine?
setMember	Liste von Peaks	Sind alle beschriebenen Resourcen enthalten?
inRange	Liste aus (Name,Frequenz, maxDist)−Gruppen	Sind alle angegebenen Frequenzen, innerhalb einer Toleranz, vorhanden?

Tabelle 2−5 Implementierte knotenlokale Bedingungstypen. Von jeder der Bedingungen kann auch die negierte Form bewertet werden.

2.4.2 Automatisch erstellte knotenlokale Bedingungen

Um die manuelle Bearbeitung zu vereinfachen, wird zu Anfang für jeden Knoten eine Liste mit Bedingungen aus den Mitgliedern des Basisknotens generiert. Die Bedingungen (Tabelle 2−6) beschreiben alle möglichen perfekten Teilspinsysteme des Spinsystems einer Aminosäure, zB. alle Peaks des HNCA Experiments oder alle Peaks, die die Frequenz CA−1 tragen. Sie sollen in einem Basisknoten mit Defekten die perfekten Teilspinsysteme isolieren.

Für perfekte Basisknoten wird dadurch automatisch das theoretische Protospinsystem isoliert, da sich die generierten Bedingungen nicht widersprechen. In einem Basisknoten, der mit einem Nachbarknoten

überlappt oder der überschüssige Peaks enthält, widersprechen sich die perfekten Bedingungen und müssen dann manuell abgeschaltet werden. Wenn Peaks fehlen, können einzelne perfekte Bedingungen nicht erfüllt werden, da diese perfekte, d.h. vollständige Teilspinsysteme voraussetzen. Diese Bedingungen müssen dann manuell abgeschaltet werden.

Zusätzliche Bedingungen können manuell angelegt werden. Durch die lokal angepaßten Bedingungen vereinfacht sich die Berechnung der Lösungsmenge, da die Größe des Lösungsraumes verringert wird.

Lokale Bedingungen	**Beschreibung**	**Konfigurationsdatei**
Einzelexperimentbedingungen	faßt alle Peaks eines Experiments zusammen.	
HNCA Paar	Je Basispeak ein Paar von HNCA Peaks mit den Frequenzen CA−1 und CA. Das Volumen von CA muß grösser als das von CA−1 sein.	listHNCA.complete
CBCA(CO)NH Paar	Je Basispeak ein Paar von CBCA(CO)NH Peaks mit den Frequenzen CA−1 und CB−1. Die Frequenzen müssen den Fuzzy−Funktionen »istEinCA« bzw. »istEinCB« entsprechen. Das Paar (CA−1&CB−1) muß der 2D Fuzzy−Funktion nach Grieziek entsprechen.	listCBCA_CONH.complete
CBCANH Quartet	Je Basispeak sind vier Peaks vorhanden (CA−1,CB−,CA,CB). Die Frequenzen müssen den Fuzzy−Funktionen »istEinCA« bzw. »istEinCB« entsprechen.Die Paare (CA−1&CB−1 , CA&CB) müssen der 2D Fuzzy−Funktion nach Grieziek entsprechen. Die CB−Peaks haben außerdem ein negatives Volumen.	listCBCANH.complete
Einzelfrequenzbedingungen.	faßt alle Peaks mit der funktional gleichen Frequenz zusammen.	
CA	Intranode verknüpftes Paar der Experimente HNCA und CBCA(CO)NH.	listCA.complete
CA−1	Intranode verknüpftes Paar der Experimente HNCA, CBCANH und CBCA(CO)NH.	listCA_m1.complete
CB−1	intranode verknüpftes Paar der Experimente CBCANH und CBCA(CO)NH.	ListCB_m1.complete
CB	Nur ein Peak im CBCANH, nicht suchbar da nicht selektiv.	

Tabelle 2−6 Liste der automatisch erstellten knotenlokalen Bedingungen für das Bax Experimentset.

Die Suche nach den Einzelfrequenzbedingungen erfordert eine intranodale Verknüpfung der Peaks. Die mit den in Tabelle (2−6) definierten Suchbedingungen gefundenen ACX werden dann in lokale Bedingungen übersetzt und können in der graphischen Benutzeroberfläche verifiziert und manuell zugelassen werden.

Die intranodalen Verknüpfungen werden temporär in einer eigenen Instanz des Peaknetzes angelegt. Dabei werden nur Verknüpfungen zwischen Peaks aus Nichtbasisexperimenten angelegt, bei denen beide Peaks mit dem gleichen Basisknoten verknüpft sind. Verknüpft werden Peaks mit gleicher Frequenz aufgrund der Abstandsfunktion.

Startpunkt		**Endpunkt**
HNCA :Achse CA	<−>	CBCANH:Achse CAB
HNCA :Achse CA	<−>	CBCA(CO)NH:Achse CAB
CBCANH:Achse CAB	<−>	CBCA(CO)NH:Achse CAB

Tabelle 2−7 Angelegte intranodale Verknüpfungen für die Detektion knotenlokaler Bedingungen.

Die Anzahl der entstehenden Verknüpfungen ist dadurch wesentlich geringer als bei einer vollständigen Verknüpfung aller Peaks mit ähnlicher Frequenz in der CA/CB Achse.

2.5 Link−Protospinsysteme

Bisher wurden die AS−Protospinsysteme nur als isolierte Einheiten beschrieben, die um Resourcen konkurrieren und bestimmten Strukturbedingungen entsprechen müssen. Außer der AS−Typeninformation enthalten die bisherigen Bedingungen kein Kriterium, das eine Zuordnung eines AS−Protospinsystems zu einer bestimmten Aminosäure begründet. Die Nachbarschaftsbeziehung zwischen Aminosäuren erzwingt, daß die Frequenzen ihrer AS−Protospinsysteme übereinstimmen. Diese Information wurde bisher nicht genutzt.

Dazu wurde das **Link−**Kriterium eingeführt, das die Reihenfolge ($ACX_1 \xrightarrow{\text{LINK}} ACX_2$) testen und bewerten kann. Um das Link−Kriterium für zwei AS−Protospinsysteme zu überprüfen, wird das Link−Protospinsystem aus den benachbarten AS−Protospinsystemen gebilden und mit dem zugehörigen Link−Protospinsystem−muster bewertet.

Definition: *Link−Protospinsystem (**LCX**)*: Ein Protospinsystem, das zwei AS−Protospinsysteme ver−bindet. Es muß den Link−Strukturtest des verwendeten Experimentsets bestehen. Es kann nicht gleichzeitig mit einem der beiden AS−Protospinsysteme erworben werden, da es die sel−ben Resourcen verwendet, wie die AS−Protospinsysteme. Die Strukturbedingungen werden vom *Link−Muster* implementiert. Das Link−Muster ist nicht kommutativ.

2.5.1 Das Link−Protospinsystemmuster für das Bax Experimentset

Das Link−Protospinsystemmuster für das Bax Experimentset besteht aus 8 Einzelbedingungen, die je einen Pfad vom i zum i+1 HNCO Peak beschreiben, an dem jeweils 4 Peaks beteiligt sind. Jeder Pfad beschreibt einen gerichteten Weg (Brücke) zwischen den benachbarten Spinsystemen.

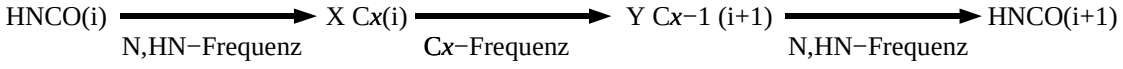


Abbildung 2−4[Cx,X,Y]−Brücke zwischen benachbarten AS−Protospinsystemen. Cx ist CA oder CB, X und Y sind die Namen eines Experiments mit einer ^{13}C −Frequenz aus dem Experimentset.

Die Einzelbedingung kann entweder *erfüllt*, *ignorierbar* oder *nicht erfüllt* sein.

Die Einzelbedingung ist erfüllt, wenn alle vier Peaks vorhanden sind und die korrespondierenden Frequenzen übereinstimmen. Die Bedingung ist nicht erfüllt, wenn alle Peaks vorhanden sind und mindestens eine Frequenz nicht übereinstimmt. Sie wird ignoriert, wenn wenn mindestens eine der Vorbedingungen nicht erfüllt ist, d.h. wenn mindestens einer der Peaks fehlt. Die HNCO Peaks sind zwangsläufig vorhanden, da sonst die AS−Protospinsysteme nicht existieren können.

Das gesamte Link−Protospinsystemmuster ist erfüllt, wenn mindestens eine Bedingung erfüllt ist. Die anderen einzelnen Bedingungen müssen dann erfüllbar oder ignorierbar sein.

2.5.2 Erzeugen der Link−Protospinsysteme

Peak im LINK	**i −Teil des i Spinsystems**	**i−1 −Teil des i+1 Spinsystems**	**Brücke**
HNCO i	HNCO		alle
HNCO i+1		HNCO	alle
HNCA CA−1		HNCA CA−1	1,4
HNCA CA	HNCA CA		1,2,3
CBCA(CO)NH CA−1		CBCA(CO)NH CA−1	2,5
CBCA(CO)NH CB−1		CBCA(CO)NH CB−1	7
CBCANH CA−1		CBCANH CA−1	3,6
CBCANH CB−1		CBCANH CB−1	8
CBCANH CA	CBCANH CA		4,5,6
CBCANH CB	CBCANH CB		7,8

Tabelle 2−8 Die einzelnen Resourcen des Link−Protospinsystems und ihr Ursprung.

NR	**i −Teil des i Spinsystems**	**i−1 −Teil des i+1 Spinsystems**
1	HNCA CA	HNCA CA−1
2	HNCA CA	CBCA(CO)NH CA−1
3	HNCA CA	CBCANH CA−1
4	CBCANH CA	HNCA CA−1
5	CBCANH CA	CBCA(CO)NH CA−1
6	CBCANH CA	CBCANH CA−1
7	CBCANH CB	CBCANH CB−1
8	CBCANH CB	CBCA(CO)NH CB−1

Tabelle 2−9 Brücken mit gleicher Frequenz im Link−Protospinsystem. Die HNCO Peaks von i und i+1 sind in allen Brücken vorhanden.

Das Link−Protospinsystem für das Bax Experimentset wird aus dem (i)−Teil des i Protospinsystems und dem (i−1)−Teil des i+1 Protospinsystems gebildet. Die Link−Protospinsysteme werden nach der Kombination der Teilspinsysteme mittels eines Protospinsystemmusters (Link−Muster) auf die Einhaltung der Strukturbedingungen überprüft. Das Link−Muster implementiert die Verknüpfungsbedingung und prüft die Übereinstimmung der Frequenzen der Peaks in den möglicherweise benachbarten AS−Protospinsystemen. Das Muster dient dabei als Strukturtest statt als Strukturgenerator. Die Konfiguration für das Link−Muster ist in Anhang D.1 aufgeführt.

2.6 Die Bewertungsfunktion

Für die Optimierung des N−ASS Problems wird eine Bewertungsfunktion gebraucht, die die Zustände im Lösungsraum relativ ordnet. Die folgenden Bewertungsfunktionen sind eine Implementierung der Kriterien in Tabelle 1−1.

2.6.1 Bewertung der AS−Protospinsysteme

Jedes Aminosäurepaar [i−1,i], außer dem ersten, bildet ein AS−Protospinsystem, wenn die i Aminosäure kein Prolin ist. Die theoretische Anzahl der AS−Protospinsysteme ist daher:

$$\begin{aligned} a_{AS,max} &= a_{amino} - 1 - a_{Pro} \qquad \text{die 1 berücksichtigt die AS am Anfang} \\ &\text{mit} \\ a_{amino} &= \text{Anzahl der Aminosäuren im Protein} \\ a_{Pro} &= \text{Anzahl der Proline im Bereich Aminosäure } 2 \ldots a_{amino} \end{aligned} \qquad (2-3)$$

Die relative Qualität eines ACX soll die Position innerhalb der Rangfolge einer Gruppe von konkurrierenden ACX festlegen. Dazu muß die Bewertung drei grundlegende Nebenbedingungen für jede Gruppe von ACX erfüllen.

- Vollständigkeit: Das ACX mit der größeren Anzahl Peaks wird höher bewertet.
- Frequenzübereinstimmung: Das ACX mit der besseren Überlappung der Frequenzen wird höher bewertet.
- Volumenverhältnis: Ein ACX, welches das theoretische Volumenverhältnis zwischen HNCA CA und HNCA CA−1 Peak einhält, erhält einen Bonus. Das Volumen wird mit der Funktion `tolerantComp` (Gl. 2−2) verglichen.

Für die relative Gewichtung der ersten beiden Ordungskriterien wurde keine theoretisch begründete Bedingung gefunden. Die Bedingungen werden gleichwertig behandelt. Die Protospinsysteme werden mittels Gl. 2−4 positionsunabhängig bewertet.

$$Q_{AS}(acx) = \frac{1}{2.5}\left(Q^{AS}_{Vollständigkeit}(acx) + Q^{AS}_{Verknüpfung}(acx) + 0.5\ Q^{AS}_{\text{HNCA Volumen}}(acx)\right)$$

mit

$$Q^{AS}_{Vollständigkeit}(acx) = \frac{1}{a_{ca,xx} + a_{cb,xx}}\left[a_{ca}(n_{ca}) + a_{cb}(n_{cb})\right]$$

$$Q^{AS}_{Verknüpfung}(acx) = \frac{1}{a_{ca,xx} + a_{cb,xx}}\left[\sum_{n_{ca}=0}^{a_{ca,xx}} Q_{kante_{ca}}(n_{ca}) + \sum_{n_{cb}=0}^{a_{cb,xx}} Q_{kante_{cb}}(n_{cb})\right] \qquad (2-4)$$

$$Q^{AS}_{\text{HNCA Volumen}}(acx) = \begin{cases} 1 & \text{wenn vol ist } wahr \\ 0 & \end{cases}$$

mit

$$\text{vol} = \text{tolerantComp}(Vol_{\text{HNCA CA}}, Vol_{\text{HNCA CA−1}}, tol_{HN}, base) > 0$$

$tol_{HN} = 0.05$ die isotopenspezifische Toleranz und $base = 0.5$

Die Anzahl und Art der Kanten hängt von den erwarteten AS Strukturtypen ab. Glycin gehört zum Strukturtyp G, alle anderen Aminosäuren zu Typ X. Dadurch ergeben sich 4 Nachbarschaftspaare mit je einem Kantenset (siehe Tabelle 2−4) und einer Bewertungsfunktion.

Nr	Peak		XX	XG	GX	GG
1	HNCA CA	CBCANH CA	✓	✓	✓	✓
2	HNCA CA−1	CBCANH CA−1	✓	✓	✓	✓
3	HNCA CA−1	CBCA(CO)NH CA−1	✓	✓	✓	✓
4	CBCA(CO)NH CA−1	CBCANH CA−1	✓	✓	✓	✓
5	CBCA(CO)NH CB−1	CBCANH CB−1	✓	−	−	−

Tabelle 2−10 Kanten entlang der CA, CB Frequenz, die zur Berechnung der Qualität eines AS−Protospin−systems verwendet werden.

Um den Aminosäuretyp eines ACX zu bestimmen, werden die CA und CB Frequenzen des ACX mit der Bewertungsfunktion `typeProb` nach Grzesiek (Anh: C) bewertet. Dadurch erhält man für jedes Ami−nosäurepaar im Protein die Wahrscheinlichkeit $Q_{AS}(acx,i)$, daß das ACX von diesem Paar Aminosäuretypen erzeugt wurde.

$$Q_{\mathrm{ASmitTyp}}(acx,as_{i-1},as_i) = Q_{AS}(acx,i) + Q_{\mathrm{ASTyp}}(acx,as_{i-1},as_i)$$

mit

$$Q_{\mathrm{ASTyp}}(acx,as_{i-1},as_i) = \mathrm{typeProb}(acx.CA_{i-1},acx.CB_{i-1},as_{i-1}.Typ) + \mathrm{typeProb}(acx.CA_i,acx.CB_i,as_i.Typ) \qquad (2-5)$$

2.6.2 Bewertung der Link−Protospinsysteme

Kanten			Nachbarschaftspaar			
Nr	i Peak in AS i	i−1 Peak in AS i+1	XX	XG	GX	GG
1	HNCA CA	HNCA CA−1	✓	✓	✓	✓
2	HNCA CA	CBCANH CA−1	✓	✓	✓	✓
3	HNCA CA	CBCA(CO)NH CA−1	✓	✓	✓	✓
4	CBCANH CA	HNCA CA−1	✓	✓	✓	✓
5	CBCANH CA	CBCANH CA−1	✓	✓	✓	✓
6	CBCANH CA	CBCA(CO)NH CA−1	✓	✓	✓	✓
7	CBCANH CB	CBCANH CB−1	✓	−	−	−
8	CBCANH CB	CBCA(CO)NH CB−1	✓	−	−	−

Tabelle 2−11 Kanten im Link−Protospinsystem, die für die Bewertung der Nachbarschaftsbeziehung zwi−schen zwei AS−Protospinsystemen genutzt werden.

Wenn zwei AS−Protospinsysteme zu benachbarten Aminosäurepositionen gehören, gibt es ein verknüpfen−des Protospinsystem. Das Link−Protospinsystem (kurz: Link) wird je zur Hälfte aus den benachbarten AS−Protospinsystemen gebildet. Den Link von Aminosäure i nach i+1 bilden alle Peaks mit CA,CB−Frequenzen der Aminosäure i aus den AS−Protospinsystemen der i und i+1 Aminosäure.

Eine Ausnahme bilden der Anfang und das Ende des Proteins und die Nachbarschaft eines Prolins. Da Proline kein NH−Proton haben, existiert weder sein AS−Protospinsystem noch das linke oder rechte Link−Protospinsystem. Die theoretische Anzahl der Links ist daher

$$a_{Link,max} = a_{amino} - 1 - 1 - 2\,a_{Pro,innen} \qquad \text{je 1 für Anfang und Ende}$$

mit

a_{amino} = Anzahl der Aminosäuren im Protein

$a_{Pro,innen}$ = Anzahl der Proline im Bereich Aminosäure $2 \ldots (a_{amino} - 1)$

(2−6)

AS	AS Nr.	AS−Protospinsy−stem	AS−Typ	Link Nr.	Link−Protospinsystem	Linktyp
Gln	1	unvollständig	(leer)X	0*	kein Link da Anfang	
Ala	2	Gln−Ala	XX	1	Ala−Ala	X
Gly	3	Ala−Gly	XG	2	Gly−Gly	Gly
Cys	4	Gly−Cys	GX	3	Cys−Cys	X
His	5	Cys−His	XX	4	(His−His) *	X
Pro	6	(His−Pro)*	XP	5*	kein Link da Pro und Protei−nende	−

*Tabelle 2−12 Klassifizierung der Protospinsystemtypen für ein fiktives Protein in Abhängigkeit von der Ami−nosäuresequenz. Die mit * gekennzeichneten Protospinsysteme existiert nicht, da sie entweder am Rand des Proteins liegen oder zu einem Prolin führen.*

Die Qualität eines Links wird mit Gleichung 2−7 bewertet. In Gl. 2−7 wird die Qualität sowohl nach der Anzahl der vorhandenen Brücken zwischen den benachbarten AS−Protospinsystemen als auch nach der Qualität der Brücken bewertet. Zusätzlich haben alle CA−Brücken zusammengenommen das gleiche Gewicht wie alle CB−Brücken zusammen, da sonst ein CA und CB verknüpfter Link schlechter gewertet wird als ein Link, in dem alle CA−Links vorhanden sind. Da die Anzahl der CB−Brücken kleiner als die der CA−Brücken ist, ergibt sich eine Gewichtung von f_{ca}=1 und f_{cb}=3. Außerdem erhält ein Link der sowohl CA als auch CB enthält einen Bonus.

$$Q_{Link}(lcx) = 1/3 * (Q^{Link}_{vollständigkeit} + Q^{Link}_{verknüpfung} + Q^{Link}_{CACBPaar})$$

mit

$$Q^{Link}_{vollständigkeit}(lcx) = \frac{1}{a_{ca,xx} + a_{cb,xx}} \left[f_{ca}\, a_{ca}(n_{ca}) + f_{cb}\, a_{cb}(n_{cb}) \right]$$

$$Q^{Link}_{verknüpfung}(lcx) = \frac{1}{a_{ca,xx} + a_{cb,xx}} \left[f_{ca} \sum_{n_{ca}=0}^{a_{ca,xx}} Q_{kante_{ca}}(n_{ca}) + f_{cb} \sum_{n_{cb}=0}^{a_{cb,xx}} Q_{kante_{cb}}(n_{cb}) \right]$$

$$Q^{Link}_{CACBPaar}(lcx) = \begin{cases} 1 & \text{wenn } a_{ca}(lcx) > 0 \wedge a_{cb}(lcx) > 0 \\ 0 & \text{wenn } a_{ca}(lcx) = 0 \vee a_{cb}(lcx) = 0 \end{cases}$$

(2−7)

und der Anzahl der vorhanden Brücken

$$a_{ca}(lcx) = \sum_{n_{ca}=0}^{a_{ca,xx}} \exists\, \text{kante}_{ca}(n_{ca})$$

$$a_{cb}(lcx) = \sum_{n_{cb}=0}^{a_{cb,xx}} \exists\, \text{kante}_{cb}(n_{cb})$$

$$f_{ca} = 1 \quad f_{cb} = 3 \quad a_{ca,xx} = 6 \quad a_{cb,xx} = 2$$

2.6.3 Bewertungsfunktion für Individuen

Die Qualität eines Individuums $\boldsymbol{S_i}$ setzt sich aus den Qualitäten der einzelnen AS−Protospinsysteme und der Link−Protospinsysteme zusammen. Die Qualität der Link−Protospinsysteme wird vierfach gewertet. Die

Existenz eines Links führt zu einem Bonus (1). Jede Aminosäureposition wird außerdem mit einer Gewichtung f_i^{as} bewertet. Die Standardgewichtung ist für alle Aminosäurepositionen 1.

$$Q_{state}(\boldsymbol{s}_i)=\sum_{i=2}^{a_{as}} Q_A(i)+\sum_{i=1}^{a_{link}} Q_L(i)$$

mit

$$Q_A(i)=\begin{cases}0 & \text{wenn } not\exists\ AS_{i-1,i}\\ & \text{sonst } q_{as}(i)\end{cases}$$

$$Q_L(i)=\begin{cases}0 & \text{wenn } not\exists\ LINK_{i+1,i+2}\\ 0 & \text{wenn } \exists\ LINK_{i+1,i+2}\wedge 10^{-4}>q_{link}(i)\\ & \text{sonst } q_{link}(i)\end{cases} \qquad (2\text{-}8)$$

$$q_{as}(i) = f_i^{as} \cdot Q_{\text{ASmitTyp}}(acx_i,\ i-1,\ i)$$
$$q_{link}(i) = f_i^{link} \cdot (4\cdot Q_{Link}(lcx_i)+1)$$

$$f_i^{link}=(f_{i+1}^{as}+f_{i+2}^{as})/2$$

Die theoretische Qualität eines Zustandes mit perfekten Protospinsystemen und perfekten Aminosäuretypen (alle AS Typen werden mit 1 bewertet) ist dann:

$$max(Q_{state,theor})=3\,a_{as}+5\,a_{link} \qquad (2\text{-}9)$$

$max(Q_{state,theor})$ wird selbst für perfekte Datensätze, in denen nur vollständige Protospinsysteme auftreten und alle gemessenen Frequenzen eines Atoms gleich sind, nicht erreicht, da die Qualität der Aminosäuren niemals 1 ist. Die theoretische Qualität ohne die AS−Typeninformation für einen perfekten Datensatz ist daher:

$$min(Q_{state,theor}) = 1\,a_{as}+5\,a_{link} \qquad (2\text{-}10)$$

Protein	Länge	Anz. Proline	a_{as}	a_{link}	theor. Q_{state}	geschätzt Q_{state}	Theor. min	bester Q_{state}	% d. thr.
C21W	148	2	145	142	1145	888.2	571	512.6	44.77
CheY	128	3	123	117	954	741	474	808	84.7
Trigger mit His Tag	113	2	110	107	865	671.2	431	465	53.76
Trigger ohne His Tag	103	2	100	97	785	609.2	391	465	59.24

Tabelle 2−13 Theoretische maximale Qualität für die untersuchten Proteine nach Gl. 2−9. Geschätzte Qualität nach Gl.2−11. Die Differenz zwischen theoretischer und tatsächlicher Qualität beruht auf den niedrigeren AS−Typ−Werten und der schlechteren Überlappung der Frequenzen. "bester" ist die höchste Qualität, die tatsächlich in einer Optimierung gefunden wurde.

2.6.3.1 Näherungsformel für die Qualität einer realen Zuordnung

Diese Näherungsformel berechnet die Qualität für einen Datensatz mit hoher Qualität, bei dem die AS−Typeninformation und die Streuung der Frequenzen der Praxis entsprechen. Die Näherungsformel wurde aus den Optimierungen von Trigger und C21W mit unterschiedlichen Cache−Konfigurationen gewonnen. Die Formel kann nur als grobe Näherung für den Endpunkt der Optimierung verwendet werden.

$$Q_{geschätzt} = a_{AS} * 2{,}06 + a_{Links} * 3{,}6 \qquad (2-11)$$

2.7 Datenfluß

Die vorhergehenden Kapitel beschreiben die Vorbereitung der Daten von den Peaklisten bis zu den Eingabedaten für die Optimierer. Abbildung 2−5 stellt diesen Datenfluß im Zusammenhang dar. Zuerst werden die Peaklisten in den PeakPool importiert. Die Peaks des PeakPools werden danach im PeakNet mit ihren Nachbarn verknüpft [Kap. 2.1.2] und die Verknüpfung mit den Konnektivitätsregeln überprüft.

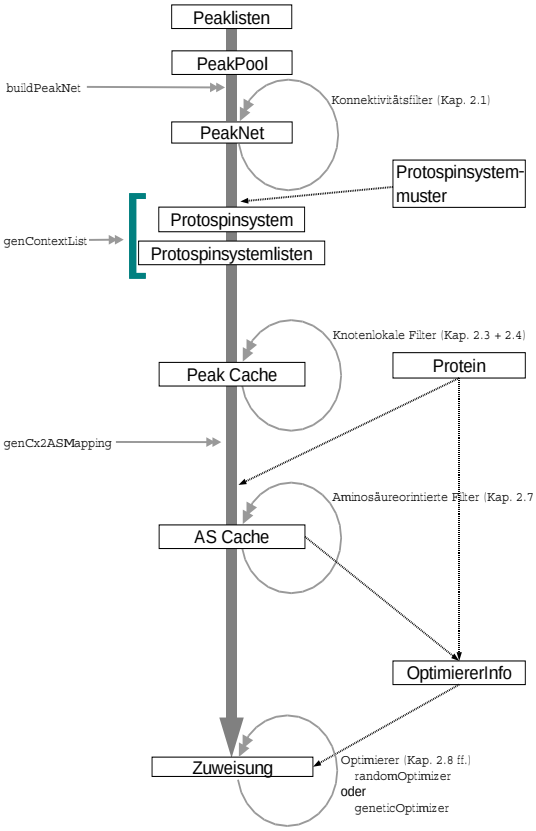


Abbildung 2−5 Übersicht über den Datenfluß vom Peakpicking zur Zuweisung.

In der Mitte der Abbildung ist die Zweiteilung des Protospin−systemcaches in den Peak−Cache und den Aminosäure−Cache (AS−Cache) zu sehen. Die Caches erhalten ihre Namen von dem Ordnungskriterium nach dem sie die Daten gruppieren. Jeder Cache besteht aus einem Verzeichnis−baum mit parallelen Verzeichnissen. Im Peak−Cache enthält jedes Verzeichnis die Daten für einen Basispeak. Der AS−Cache faßt die Protospinsystemlisten für eine Aminosäure zusammen.

Die Protospinsystemsuche [Kap. 2.3] stellt die Pro−tospinsysteme als Protospinsystemliste in den Peak−Cache. Innerhalb des Peak−Caches werden diese Protospinsy−stemlisten mittels der knotenlokalen Bedingungen [Kap. 2.4] reduziert. Dadurch entstehen neue Protospinsystemlisten, die auch im Peak−Cache gespeichert werden. Mit Hilfe der Aminosäuretypeninformation werden die Pro−tospinsysteme aus dem Peak−Cache auf die möglichen Aminosäuren im AS−Cache verteilt. Jeder Ami−nosäure können mehrere Protospinsystemlisten zugeordnet sein.

Die Konfiguration des Optimierers definiert den Ausschnitt des AS−Caches, der für den Optimierer rele−vant ist. Dadurch kann man den gleichen AS−Cache auf unterschiedliche Weise für eine Optimierung nut−

zen, indem man den zu optimierenden Aminosäuren im Protein unterschiedliche Protospinsystemlisten im AS−Cache zuweist. Dazu wird in der OptimiererInfo−Datei für jede Aminosäure eine Liste benannt, die für diese Aminosäure geladen werden soll.

Verzeichnis	**Erklärung**
`..\Trigger`	Hauptverzeichnis
`..\Trigger\bin`	Link zu den Programmen
`..\Trigger\cache`	Cache Verzeichnis

`..\Trigger\cache`
`\p0001`
`\p0002`
`\...`
`\p0203`
	Peak Cache
	je ein Verzeichnis pro Basispeak

`\as001`
`\as001`
`\...`
`\as113`
	Aminosäure Cache
	je ein Verzeichnis pro Aminosäure

`..\Trigger\pattern`	die Protospinsystemmuster
`..\Trigger\orginale`	die Peaklisten
`..\Trigger\sequences`	die Optimierungsergebnisse
`..\Trigger\sequences\macros`	die Konfigurationsmakros
`..\Trigger\sequences\<<optimierungsstufe>>`	die Optimierungsergebnisse einer Stufe

Tabelle 2−14 Dateisystemstruktur eines Projektes anhand von "Trigger"

Jeder Position im Protein können nur die Protospinsysteme zugewiesen werden, die sich in der zugeordneten Liste befinden. Die Konfiguration entscheidet so darüber, ob eine Aminosäure mit einer Protospinsystemliste mit oder ohne AS−Typeninformation geladen wird. Auf die gleiche Weise können auch neue Filter für den AS Cache entworfen werden. Ein Positionsverbot für einzelne Protospinsysteme und Peaks kann man zum Beispiel implementieren, indem man einer Aminosäure nur ein Protospinsystem zuweist oder aus den Listen einer Aminosäure alle Protospinsysteme mit einem bestimmten Peak entfernt. Zusätzlich gibt es noch eine globale Protospinsystemliste, die alle geladenen Protospinsysteme vereinigt. Diese Liste, die alle Protospinsysteme des aktuell konfigurierten Caches enthält, wird automatisch erstellt. Dadurch kann ein Cache ohne Positionsverbot konfiguriert werden, indem man diese Liste allen Aminosäuren zuweist.

Wie viele Protospinsysteme die Liste einer Aminosäureposition enthält, hängt von der Qualität der Spektren, dem Protein und den verwendeten Filtern ab. Spektren mit einem hohen Anteil von überlappenden Spinsystemen oder einem niedrigen Signal zum Rausch−Verhältnis erzeugen eine längere Liste als gut aufgelöste Spektren, da in die Liste immer alle gleich wahrscheinlichen Kandidaten aufgenommen werden müssen. Außerdem hängt die Länge der Listen von den Filtern ab, die auf die Listen im Peak− und AS−Cache wirken. Ohne AS−Typeninformationen sind die Listen um einen Faktor 10 länger. Ein Vergleich zwischen den Konfigurationen "mit Positionsverboten durch die Typenerkennung" und "ohne Nebenbedingungen" für die untersuchten Proteine befindet sich in Tabelle 1−3.

Von den Optimierungsalgorithmen wird nur der Ausschnitt des AS−Caches gelesen, der durch die Konfiguration bestimmt wird. Die Ein− und Ausgabe der Optimierer ist eine Zuordnung. Die Optimierer können daher eine beliebige Zuordnung lesen und verfeinern. Dabei kann die Konfiguration des AS−Caches und der

Optimierer gewechselt werden. Wenn die Zuordnung beim Lesen Protospinsysteme enthält, die in der neuen Cachekonfiguration nicht enthalten sind, wird der entsprechende Abschnitt geleert.

2.8 Der Random Algorithmus

Der RANDOM Algorithmus implementiert einen statistischen Algorithmus, der dem *threshold accepting* und dem *Lin−Kernighan* Verfahren ähnlich ist.

```
Algorithmus: RandomOptimize
parameter:
   S_best,        der beste bisher gefundene Zustand.
   S,             der Arbeitszustand.
return:
variablen:
   l,       Schleifenzähler
   l_last,  Schleifenzähler bei der letzten Verbeserung.
   TTL,     Lebenszeit, die S schon hinter sich hat,
               startet bei 0, wird in jedem Schleifendurchlauf heraufgesetzt.
               Bei TTL == maxTTL wird der Zustand durch S_best ersetzt.
               und TTL auf 0 gesetzt
   minTTL   wenn TTL < minTTL ist, darf die q(S) ≤ q(S_best)*f sein
   maxTTL   endpunkt für TTL,
   qTh      Abk. für qThreshold
   qThD     Abk. für qThresholdDecayRate
   t        Qualitätsschwelle
begin
   loop: solange das Abbruchkriterium noch nicht erfüllt ist:
      l ++;
      wähle eine Aktion A aus k Aktionen mit der Wahrscheinlichkeit p aus.
      führe A auf S aus.
      fülle S mit Zufallselementen auf wenn weniger als 75% Elemente vorhanden sind.
      wenn q(S) > q(S_best)
         kopiere S nach S_best und aktualisiere l_last = l
      berechne die Schwelle t = q(S_best) * qTh
      wenn
         die TTL über maxTTL liegt
        oder
         die TTL > minTTL ist und q(S) unterhalb der Schwelle t liegen.
      dann
         kopiere S von S_best,
         setzte TTL = 0

      TTL ++;
      wenn l_last < maxIdelLoops
         qTh := qTh + qThD * (1-qTh)
   end loop
end
```

Der Algorithmus unterscheidet sich vom *threshold accepting* durch die Einführung zusätzlicher Kontroll−mechanismen.Bei reinem *threshold accepting* kann das Individuum solange verändert werden, bis seine Qualität die Qualitätsschwelle unterschreitet. Unterschreitet es die Mindestqualität wird die letzte Aktion wie−der zurückgenommen und nach einer neuen Aktion gesucht. Die Mindestqualität berechnet sich nach Gl.2−12 aus der Qualität des aktuell besten Individuums und `qThreshold`. `qThreshold` wird mit Gl.2−13 erhöht, wenn die letzte Verbesserung von $\boldsymbol{S}_{best}$ mindestens `maxIdleLoops` Zyklen zurückliegt.

$$Q_{Schwelle} = Q(\boldsymbol{S}_{best}) * qThreshold \qquad (2-12)$$

$$qThreshold = qThreshold + qThresholdDecayRate * (1 - qThreshold) \qquad (2-13)$$

```
Algorithmus: RandomOptimize
parameter:
   S_best,        der beste bisher gefundene Zustand.
   S,             der Arbeitszustand.
return:
variablen:
   l,       Schleifenzähler
   l_last,  Schleifenzähler bei der letzten Verbeserung.
   TTL,     Lebenszeit, die S schon hinter sich hat,
               startet bei 0, wird in jedem Schleifendurchlauf heraufgesetzt.
               Bei TTL == maxTTL wird der Zustand durch S_best ersetzt.
               und TTL auf 0 gesetzt
   minTTL   wenn TTL < minTTL ist, darf die q(S) ≤ q(S_best)*f sein
   maxTTL   endpunkt für TTL,
   qTh      Abk. für qThreshold
   qThD     Abk. für qThresholdDecayRate
   t        Qualitätsschwelle
begin
   loop: solange das Abbruchkriterium noch nicht erfüllt ist:
      l ++;
      wähle eine Aktion A aus k Aktionen mit der Wahrscheinlichkeit p aus.
      führe A auf S aus.
      fülle S mit Zufallselementen auf wenn weniger als 75% Elemente vorhanden sind.
      wenn q(S) > q(S_best)
         kopiere S nach S_best und aktualisiere l_last = l
      berechne die Schwelle t = q(S_best) * qTh
      wenn
         die TTL über maxTTL liegt
        oder
         die TTL > minTTL ist und q(S) unterhalb der Schwelle t liegen.
      dann
         kopiere S von S_best,
         setzte TTL = 0

      TTL ++;
      wenn l_last < maxIdelLoops
         qTh := qTh + qThD * (1-qTh)
   end loop
end
```

Im *Lin−Kernighan* Verfahren darf der Startzustand S_{best} während der ersten *n* Züge verschlechtert wer−den, muß sich danach aber kontinuierlich verbessern. Nach einer vorher festgelegten Anzahl Züge wird die Qualität des Zustands mit der des Startzustands verglichen und der Endzustand nur behalten, wenn er bes−ser als der Startzustand ist.

Beide Verfahren erfordern, daß der zeitlich nächste Zustand kostengünstig bewertet werden kann und daß die Rückkehr zum vorherigen Zustand billig[2] ist. Dieser "Schritt zurück" wird in beiden Verfahren sehr häufig (mindestens bei 50% der Schleifendurchgänge) ausgeführt. Für Zustandsobjekte, wie sie im Assignment verwendet werden, sind diese Operationen sehr teuer, da man den gesamten Zustand neu berechnen muß, wenn man zu einem vorhergehenden Zustand zurückkehren will.

Im RANDOM Algorithmus wurde daher auf die spekulative Bewertung und die Rückkehr zum vorherigen Zustand verzichtet. Statt dessen darf der Startzustand S_i solange verändert werden, bis er die Qualitäts−schwelle unterschreitet. Unterschreitet er die Mindestqualität, startet der Algorithmus wieder mit einer Kopie des besten bekannten Zustandes S_{best} .

Außerdem darf sich das Individuum nur *maxTTL* Schritte von S_{best} entfernen. Findet es innerhalb der *maxTTL* Schritte keinen besseren Zustand als S_{best}, startet das Individuum wieder bei S_{best}. Dafür darf das Individuum sich innerhalb der ersten *minTTL* Schritte unterhalb der Qualitätsschwelle aufhalten ohne ersetzt zu werden. Erst danach gilt die Qualitätsbeschränkung wie im *threshold accepting*. Daher kann es anfangs durch einen im *threshold accepting* verbotenen Bereich tunneln und kann Aktionspfade durchlaufen, die beim *threshold accepting* verboten sind. Die Begrenzung der maximalen Länge des Aktionspfades durch

2 Teuer und billig im Sinne von verbrauchter CPU Zeit.

maxTTL führt zu einer intensiveren lokalen Suche um S_{best} . Die Anzahl der Aktionen ist dabei nicht die Anzahl der Elementaroperationen, sondern die Anzahl der Aktionen wie sie in den folgenden Unterkapiteln definiert werden. Ein Individuum kann sich daher maximal *minTTL* * 32 Elementaroperationen weit bewegen, wenn es *minTTL* unabhängige, nicht überlappende **`moveRange`** Operationen ausführt.

maxTTL muß immer größer oder gleich *minTTL* sein. Außerdem muß *maxTTL* größer 0 sein. Wenn *maxTTL* auf $+\infty$ und *minTTL* auf 0 gesetzt werden, entspricht der Algorithmus weitgehend dem *threshold accepting* mit dem Unterschied, das der aktuelle Zustand bei einer Kollision mit der Qualitätsschwelle durch S_{best} ersetzt wird.

Dieser Kontrollmechanismus entspricht entfernt der Qualitäts− und Pfadbeschränkung im Lin−Kernig−han Verfahren wie es für das TSP eingesetzt wird. [Reinelt94, S.124, Lin,Kernighan73]

2.8.1 Aktionsprofile

Die innere Schleife des RANDOM Algorithmus besteht aus der zufallsgesteuerten Auswahl und Ausführung einer Mutationsaktion. RANDOM stellt 6 verschiedene Operationen zur Verfügung. Die Wahrscheinlichkeit, mit der eine Aktion ausgewählt wird, hängt vom sogenannten Aktionsprofil ab. Die Aktionsprofile definieren die prozentualen Anteile der einzelnen Aktionen am Gesamtprozeß.

Jedes Profil kann aus mehreren Einzelprofilen bestehen, die jeweils nur innerhalb eines *qThreshold* −Bereiches gelten. Der RANDOM Algorithmus aktiviert beim Überschreiten der *qThreshold*−Grenze jeweils das nächste höhere Profil.

Der RANDOM Algorithmus kann mit verschiedenen Aktionsprofilen geladen werden. Die folgende Tabelle führt die in dieser Untersuchung verwendeten Profile und ihre Kurzbezeichnungen auf. Es wurden keine Profile mit Profilveränderung untersucht.

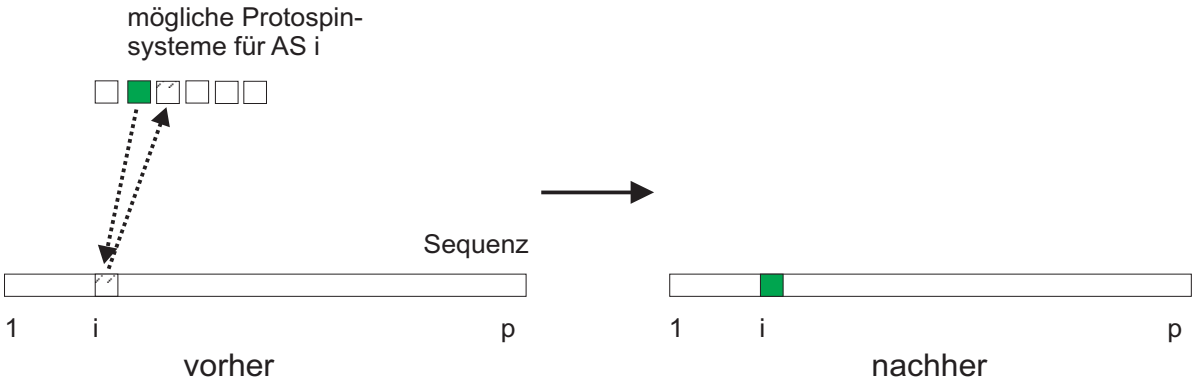
Name	Kurz/Alias	exchange	swap	moveRange	Crosso−ver	upgrade	moveSingle
single_only	exchange_only	100	0	0	0	0	0
swap80	mS80x20	20	80	0	0	0	0
swap60	mS60x40	40	60	0	0	0	0
swap50	mS50x50	50	50	0	0	0	0
swap40	mS40x60	60	40	0	0	0	0
swap20	mS20x80	80	20	0	0	0	0
swap10	mS10x90	90	10	0	0	0	0
swap05	mS05x95	95	5	0	0	0	0
mix1	m1	40	40	20	0	0	0
mix1_2	**m1v2**	**30**	**30**	**20**	**0**	**0**	**20**
mix1_3	m1v3	40	40	5	0	0	20
mix2	m2	20	60	20	0	0	0
mix3	m3	60	20	20	0	0	0
mix4	m4	20	50	20	10	5	0

Tabelle 2−15 Vordefinierte Aktionsprofile ohne Variation der Aktionsanteile: Die einzelnen Zeilen müssen sich nicht zu 100% addieren, da sie intern auf 100% normiert werden. Das Standardprofil ist hervorgehoben. Zur Umrechnung in Prozentanteile gilt $\mathrm{pct}(aktion_i) = 100 * anteil_i \,/\, \sum anteil_n$.

Die einzelnen Aktionen sind in den Pseudocodes (`exchange`) bis (`crossOver`) dargestellt. Jede Aktion kann fehlschlagen, wenn die Voraussetzungen in *maxVersuche* Versuchen nicht getroffen wurden. Für dieses Individuum wird dann in dieser Runde keine Ersatzaktion ausgeführt. Fehlgeschlagene Aktionen werden trotzdem mitgezählt.

Die Konvergenzgeschwindigkeit des RANDOM Algorithmus hängt stark von der Wahrscheinlichkeitsverteilung der Aktionen ab.

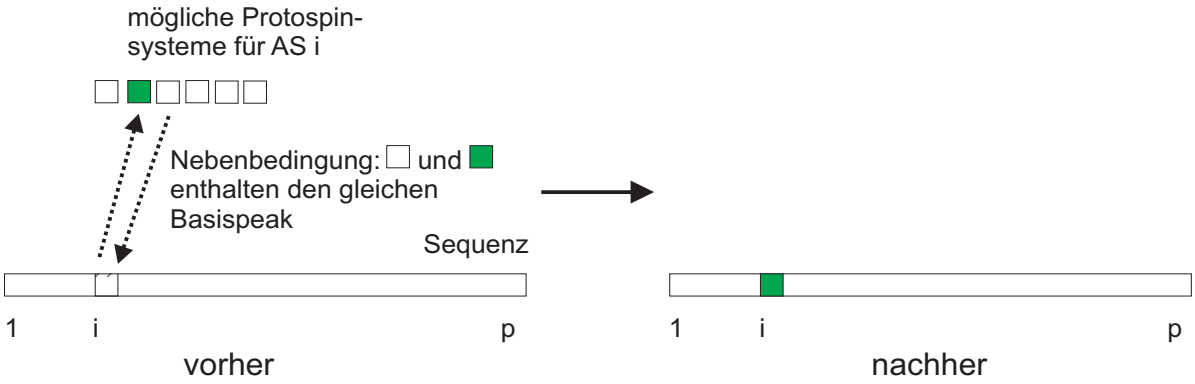
```
Algorithmus: exchange (S   das Individuum)
variablen:
   a           Aminosäureposition
   proto       ein Protospinsystem.
begin
   proto = leer
   loop: solange proto leer ist und versuche ≤ maxVersuche .
      versuche = versuche + 1
      wähle eine belegbare Aminosäureposition a in S1 aus.
      wähle ein Protospinsystem proto aus der liste der möglichen
            Protospinsysteme für die Aminosäure a aus.
      wenn proto nicht konfliktfrei erworben werden kann
            proto = leer
   end loop
   ersetze das Protospinsystem an der Position a durch proto.
end
```



exchange tauscht das Protospinsystem an einer einzelnen Aminosäureposition gegen ein zufällig gewähltes neues Protospinsystem aus. Wenn das neue Protospinsystem nicht konfliktfrei erworben werden kann, werden bis zu 8 neue Versuche gestartet, bis ein passendes Protospinsystem gefunden wurde, oder die Zuweisung der Aminosäure wird nicht verändert.

Die Zustandsdifferenz durch diese Aktion ist $\left\{T^{-}_{E_1,i},\ T^{+}_{E_2,i}\right\}$.

```
Algorithmus: upgrade (S das Individuum)
variablen:
   a           Aminosäureposition
   proto       ein Protospinsystem.
begin
   proto = leer
   loop: bis proto nicht leer ist oder versuche >= maxVersuche.
      versuche = versuche + 1
      wähle eine belegbare, nicht leere Aminosäureposition a in S aus.
      wähle ein Protospinsystem proto aus der Liste der möglichen
         Protospinsysteme für die Aminosäure a aus das den gleichen
         HNCO Peak verwendet, wie das aktuelle Protospinsystem bei a
      wenn proto nicht konfliktfrei erworben werden kann
         proto = leer
   end loop
   wenn  proto nicht leer ist
      ersetze das Protospinsystem an der Position a durch proto.
end
```

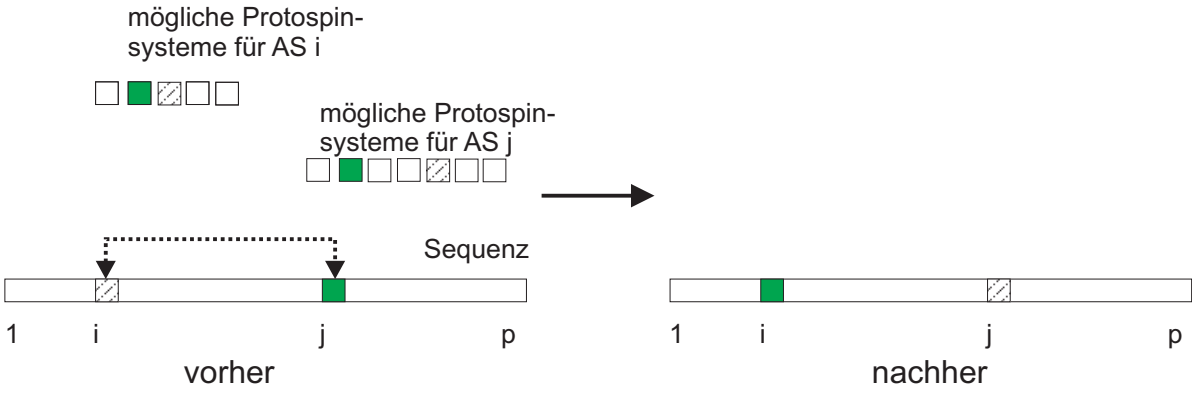


upgrade entspricht exchange, indem es das Protospinsystem einer einzelnen Aminosäure aus−tauscht. Im Gegensatz zu exchange tauscht upgrade nur gegen Protospinsysteme, die dem gleichen Basispeak angehören. upgrade hinterläßt keine leere Position.

Die Zustandsdifferenz durch diese Aktion ist $\left\{T^{-}_{E_1,i},\ T^{+}_{E_2,i}\right\}$ mit $E_1,\ E_2 \in S_{BP_x}$.

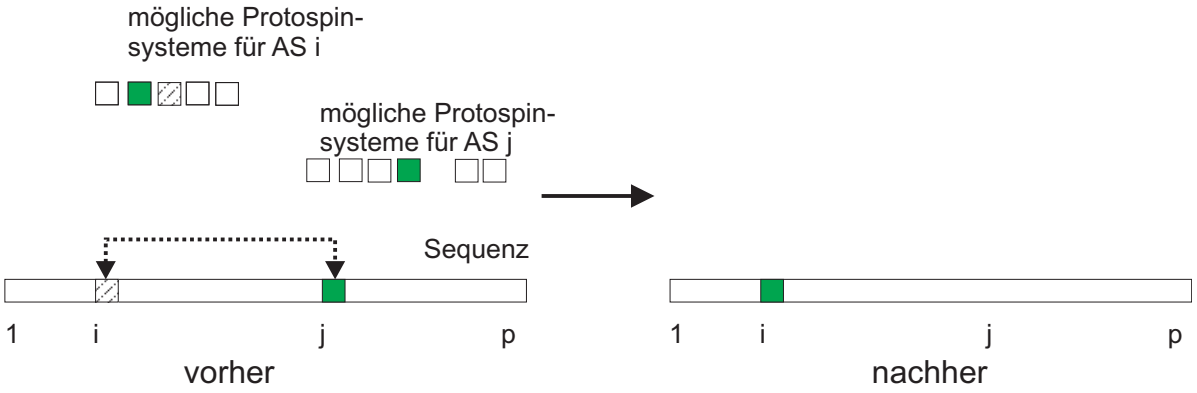
```
Algorithmus: swap (S das Individuum)
variablen:
   a1,a2             Aminosäurepositionen
   proto1,proto2     Protospinsysteme.
begin
   loop: solange versuche ≤ maxVersuche
      suche zwei belegbare Aminosäurepositionen a1 und a2, die entweder ein leeres
            Protospinsystem oder ein Protospinsystem das
            an der jeweils anderen Position erlaubt ist, enthalten .
      versuche += 1
   end loop
   wenn kein Aminosäurepaar gefunden wurde return
   ersetze das Protospinsystem an der Position a1 durch proto2.
   ersetze das Protospinsystem an der Position a2 durch proto1.
end
```

Swap tauscht die Protospinsysteme zwischen zwei Aminosäuren aus, sofern die Protospinsysteme wechselweise kompatibel sind. Die Aktion sucht 8 mal nach einem neuen Tauschpartner. Wenn sie keinen findet, bricht die Aktion ab. Swap hinterläßt keine leeren Zuweisungen.



Die Zustandsdifferenz durch diese Aktion ist $\left\{T^{-}_{E_1,i},\ T^{-}_{E_2,j},\ T^{+}_{E_2,i},\ T^{+}_{E_1,j}\right\}$.

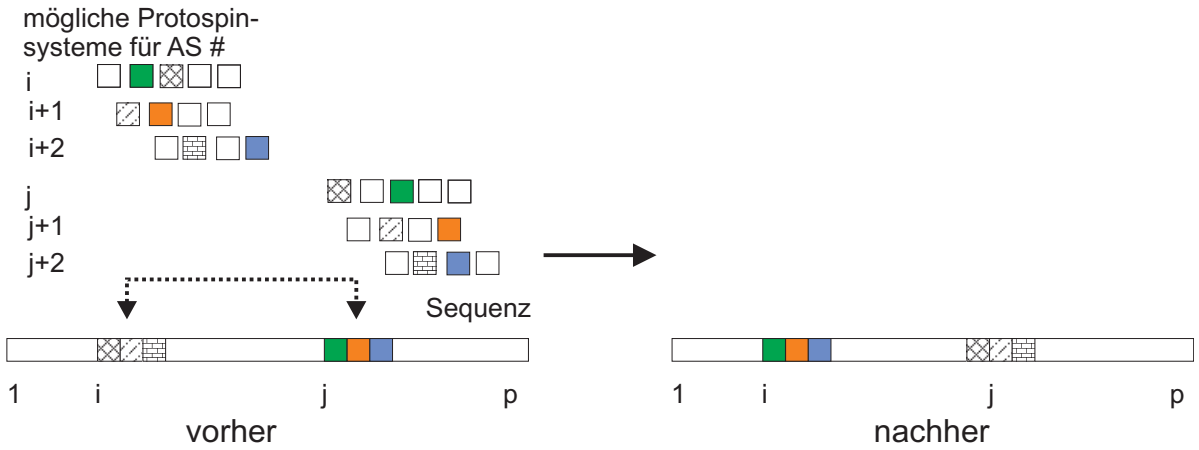
```
Algorithmus: moveSingle (S das Individuum)
variablen:
   a1,a2          Aminosäureposition
   proto1,proto2  die Protospinsysteme an den Positionen a1 und a2.
begin
   proto1 = proto2 = leer
   loop: bis proto1 nicht leer ist oder versuche >= maxVersuche.
      versuche = versuche + 1
      wähle eine belegbare, belegte Aminosäureposition a1 in S aus.
      wähle eine Aminosäureposition a2 in S aus die proto1 aus a1 aufnehmen kann.
      wenn proto1 nicht konfliktfrei in a2 erworben werden kann und versuche < maxVersuche-1.
           proto1 = leer
   end loop
   wenn  proto1 nicht leer ist,
      ersetze das Protospinsystem an der Position a2 durch proto1.
   wenn  proto2 nicht leer ist,
      ersetze das Protospinsystem an der Position a1 durch proto2.
end
```



MoveSingle entspricht `swap`, indem es die Protospinsysteme zwischen zwei Aminosäurepositionen austauscht. Im Gegensatz zu `swap` kann es eine leere Zuweisung an der Startposition hinterlassen, wenn das Protospinsystem der Zielposition nicht mit der Startposition kompatibel ist.

Die Zustandsdifferenz durch diese Aktion ist $\left\{T^{-}_{E_1,i},\ T^{-}_{E_2,j},\ T^{+}_{E_2,i},\ T^{+}_{E_1,j}\right\}$ oder $\left\{T^{-}_{E_1,i},\ T^{-}_{E_2,j},\ T^{+}_{E_1,j}\right\}$.

```
Algorithmus: moveRange (S  das Individuum)
variablen:
   len               Länge der Protospinsystemketten
   a1,a2             Aminosäurepositionen, Startpositionen für proto1 und proto2.
   proto1,proto2     gleich lange Protospinsystemketten.
begin
   len = 0
   loop: solange len == 0 und versuche ≤  maxVersuche
      versuche = versuche + 1
      suche zwei belegbare  Aminosäurepositionen a1 und a2, die entweder ein leeres
            Protospinsystem oder ein Protospinsystem das an der jeweils
            anderen Position erlaubt ist, enthalten. a1 und a2 sind der Kettenstart.
      verlängere beide Ketten am Ende um maximal maxLen Aminosäuren,
            bis das folgende Protospinsystempaar nicht mehr vertauscht werden kann
            oder das Ende des Proteins erreicht wurde.
      wenn die kette < 2 und versuche < maxVersuche
            len = 0
   end loop
   ersetze len Protospinsysteme ab der Position a1 durch proto2.
   ersetze len Protospinsysteme ab der Position a2 durch proto1.
end
```



moveRange bewegt einen Bereich von maximal 8 verketteten Protospinsystemen zu einem anderen kompatiblen Aminosäurebereich im Protein. Wenn der Zielbereich selbst Protospinsysteme enthält, versucht die Aktion, den ab der Startposition wechselweise kompatiblen Abschnitt zwischen den Bereichen auszutauschen. Ein Aminosäurebereich ist kompatibel mit einer Kette von Protospinsystemen, wenn alle kopierten Protospinsysteme in der Liste des Zielbereichs vorhanden sind.

Die Zustandsdifferenz durch diese Aktion entspricht 2 bis 8 swap Operationen, d.h. 8 bis 32 Elementaroperationen. Ein Individuum kann sich maximal *minTTL* * 32 Elementaroperationen weit vom Ausgangszustand entfernen, wenn es *minTTL* unabhängige, nicht überlappende **moveRange** Operationen ausführt. Außerdem können sich zwei Individuen um maximal 2 * a_{AS} − 2. Elementaroperationen unterscheiden (Gl. 2−16). Für "Trigger" ist diese Anzahl nach 8 **moveRange** Operationen erreicht.

```
Algorithmus: Crossover
parameter:
return:
variablen:
begin
   führe eine Crossover Aktion entsprechend den Aktionen im genetischen Optimierer aus.
      (siehe Crossover in Kapitel 2-9 ff.)
end
```

Crossover führt eine der Crossover Aktionen des Genetischen Optimierers innerhalb des RANDOM Operators aus. Dabei wird ein Zielzustand durch das Crossover Produkt zweier Zustände ersetzt.

Die Zustandsdifferenz durch diese Aktion entspricht 2 bis $a_{AS}/2$ swap Operationen, d.h. 8 bis $4 * a_{AS}/2$ Elementaroperationen und hängt vom gewählten Crossover−Operator ab.

Auf das N−ASS bezogen implementiert jede Aktion eine Wahlmöglichkeit oder Entscheidung bei der Zuweisung von Protospinsystemen zu den Aminosäurepositionen.

`exchange` und `upgradeSingle` sind notwendig, um neue Protospinsysteme in die Zuweisung einzuführen. Außerdem sind sie die direkte Implementierung der Operationen $T^{\pm}$ und müssen daher vorhanden sein, um jeden Punkt im Lösungsraum erreichen zu können. `exchange` ermöglicht die Auswahl zwischen verschiedenen Interpretationen einer Peakgruppe mit dem gleichen Basispeak als Protospinsystem, die nur global, d.h. im Kontext mit den restlichen zugewiesenen Protospinsystemen, erfolgen kann. Nur durch `exchange` und `upgradeSingle` kann die lokale Entscheidung über die richtige Interpretation einer Peakgruppe bis in die globale Optimierungsphase verschoben werden. Ohne diese Aktionen müßte diese Entscheidung vor der Optimierung bei der Auswahl der Protospinsysteme getroffen werden, so wie es z.B. für PASTA notwendig ist.

`swap`, `moveSingle` und `moveRange` vertauschen unterschiedlich große Abschnitte der Zuweisung und sorgen für eine Verschiebung der Protospinsysteme zwischen gleichartigen Abschnitten des Proteins. Gleichartig sind dabei alle Aminosäuren, die, mit den Entscheidungsfunktionen in Anh. C, als mögliche Interpretation der CA/CB Frequenzen des verschobenen Protospinsystems in Frage kommen z.B. Glu Gln. `swap` entspricht der Auswahl zwischen verschiedenen Positionen in der Sequenz, die gleichartige Aminosäuretypen tragen. Für zwei Sequenzen von Protospinsystemen wird die Entscheidung zwischen verschiedenen Domänen des Proteins mit gleichartigen Aminosäuren durch die `moveRange` Operation ermöglicht.

2.9 Der Genetische Algorithmus.

Der genetische Optimierer (GENETIC) basiert auf dem allgemeinen Genetischen Algorithmus. Allerdings wurden einige Teilschritte gegen spezifischere ausgetauscht.

```
Algorithmus: GENETIC
parameter:
   arenaSize,     Anzahl der Individuen.
   S(arenaSize)   die Individuen (Zustände)
   S_best         das Individuum mit der bisher höchsten Qualität.
Return:
   S_best         das Individuum mit der höchsten Qualität.
variablen:
begin
   initialisiere die Individuen
   loop: bis Abbruchbedingung erfüllt
      mutiere die Individuen
      führe Rekombination und Evolution durch
   end loop
end
```

Die in der Literatur getrennten Teilalgorithmen Rekombination und Evolution wurden zu einem einzigen Algorithmus zusammengefaßt. Die Evolution wird dadurch realisiert, daß die schlechtesten Individuen in der Rekombination als Zielzustände dienen, die von den neu erzeugten Individuen überschrieben werden. Die ersetzten Individuen werden unter den **n** schlechtesten ausgewählt, die gleichzeitig schlechter als `genCrossoverThreshold`% der Qualität des aktuell besten Individuums sind.

2.9.1 Der Mutationsalgorithmus

Der Mutationsalgorithmus wurde durch den RANDOM Algorithmus implementiert. RANDOM wird über einen veränderten Parametersatz an die Aufgabe angepaßt und erfordert keine Veränderung des RANDOM Algorithmus. Dazu werden die Parameter auf niedrigere oder allgemeinere Werte gesetzt, um die Laufzeit der einzelnen Mutation zu begrenzen.

Die Parameter wurden ursprünglich so gewählt, daß eine Mutation weniger als eine Minute pro Individuum braucht. Insbesondere wird die maximale Schleifenzahl durch den Parameter `subMaxLoops` so begrenzt, das RANDOM nach kurzer Zeit abgebrochen wird. `subMaxLoops` wurde mit Werten im Bereich 100 bis 16000 getestet. Außerdem wird die Zeit für eine Mutation direkt auf wenige Minuten begrenzt.

Das einzelne Individuum kann sich daher während einer Mutation weitgehend verändern. Dadurch entspricht das Verhältnis zwischen Mutation und Rekombination nicht mehr dem im klassischen Genetischen Algorithmus, bei dem die Mutationsrate klein gegenüber der Crossover−Rate ist. Die Anzahl der Mutationen pro Individuum zwischen den Crossover−Ereignissen wird in der Literatur sehr niedrig angegeben [Reinelt94, S.158]. Die Hauptlast der Optimierung und der Generierung von neuen Individuen soll theoretisch auf dem Crossover liegen. Die Individuen im N−ASS Problem erfahren aber durch die Rekombination einen Schock, der ihnen, aufgrund von Resourcenkonflikten, eine sehr niedrige Qualität aufzwingt. Die Individuen verlieren durch das Crossover die Protospinsysteme, die mit den elterlichen Genen im Konflikt stehen. Sie starten daher mit einer niedrigeren Qualität als ihre Eltern. Diese Resourcenkonflikte müssen durch einige RANDOM−Zyklen repariert werden, in denen die freien Positionen mit passenden Protospinsystemen aufgefüllt werden können.

```
Algorithmus: Rekombination und Evolution
parameter:
variablen:
   arenaSize,  Anzahl der Individuen.
   S(arenaSize)   die Individuen (Zustände)
   S_best         das Individuum mit der bisher höchsten Qualität.
   e,          Anzahl der Elternzustände.
   ü,          Anzahl der garantiert überlebenden Zustände
   n_max,      Anzahl der maximal ersetzten Zustände
   n,          Anzahl der tatsächlich ersetzten Zustände
   P1(arenaSize),P2(arenaSize),  Partnertabelle
   genCrossoverThreshold,        Faktor [0 - 1], zu Berechnung des Schwellwertes.
   konstante aufbauRate    = 0.3
   konstante zerfallsRate  = 0.27
begin
   sortiere die Arena absteigend, das Individuum mit der höchsten Qualität q(S) kommt an die
      erste Position.
   kopiere das beste Individuum nach S_best, wenn q(S(1)) > q(S_best)
   bestimme den Index i_grenze ab dem gilt:
       q(S(i)) ≤ q(S_best) * genCrossoverThreshold für alle i>i_grenze
   n := i_grenze
   begrenze die Crossover Anzahl n auf n_max
   if(slidingCross < minimum CrossoverRate){
         erzwinge ein Crossover und reduziere genCrossoverThreshold.
         genCrossoverThreshold := genCrossoverThreshold + genCrossoverThreshold_decay_rate
                                    * (1-genCrossoverThreshold)
         n := 1
   }
   if(n < arenaSize+1){
      erzeuge die Partnertabellen P1 und P2
      i := arenaSize
      für jedes Paar P1[i], P2[i] für das gilt P1[i] ungleich P2[i]
         wenn i gerade ist.
            führe ein Crossover mit zwei Zielzuständen durch,speichere in S[i] und S[i-1]
            i = i - 2
         wenn i ungerade ist.
            führe ein Crossover mit einem Zielzustand durch,speichere in S[i]
            i = i - 1
   }
   crossCount += n
   slidingCross += crossCount * aufbauRate - slidingCross * zerfallsRate
end
```

2.9.2 Die Qualitätsschwelle

Wenn innerhalb einer Periode von `genMaxIdleLoops` genetischen Zyklen keine Verbesserung des besten bekannten Zustandes S_{best} zustande kommt, wird die Qualitätsschwelle `genQThreshold` mittels Gl. 2−14 heraufgesetzt. Dadurch wird nach einigen Zyklen ohne Verbesserung der Arena Crossover−Aktionen erzwungen, da die Qualitätsschwelle über die Qualität der schlechtesten Individuen steigt.

$$genQThreshold = genQThreshold + genQThresholdDecayRate * (1 - genQThreshold) \qquad (2-14)$$

2.9.3 Zwangscrossover

Gegen Ende der Optimierung kann die Qualität aller Individuen oberhalb der Qualitätsschwelle liegen, obwohl das globale Optimum noch nicht erreicht wurde. Der Genetische Optimierer wird dann lange nicht konvergieren, da Veränderungen an den Individuen nur noch durch die Mutationsoperation eintreten, weil keine Crossover−Aktionen zugelassen werden. Durch eine geeignete Konfiguration der minimalen gleitenden Crossover−Rate `genCrossoverThreshold` kann eine Mindestrate an Crossover−Aktionen pro genetischem Zyklus erzwungen werden. Die zusätzlichen Crossover−Aktionen werden erzwungen, wenn `genCrossoverThreshold` über `slidingCross` liegt.

Die Länge der Periode ohne Crossover wird durch eine gleitende Mittelung der Crossover Raten kontrolliert. Die Crossover Rate (`slidingCross`) berechnet sich aus der Summe aller Crossover und der alten Crossover Rate wie in Gl. 2−15 dargestellt.

$$slidingCross = slidingCross + crossCount * aufbauRate - slidingCross * zerfallsRate$$
$$\text{mit} \quad aufbauRate = 0.3, \quad zerfallsRate = 0.27 \tag{2−15}$$

$$crossCount = crossCount + \text{Anzahl der Crossover in diesem genetischen Zyklus}$$

2.9.4 Die Wahl der Eltern

Die potentiellen Eltern werden positionsproportional ausgewählt. D.h. die Eltern werden nach ihrer Qualität sortiert und danach proportional ihrer Position ausgewählt. Die ersten *seedPoolSize* Eltern werden als potentielle Eltern in der Partnerwahl verwendet. Ob sie tatsächlich an Kindszuständen beteiligt sind, hängt auch von der Anzahl der zu erzeugenden Kindszustände und der Position der Eltern in der Partnertabelle ab.

2.9.5 Die Partnerauswahl

```
Algorithmus: Erzeuge die Partnertabelle
parameter:
   P1(arenaSize),P2(arenaSize),  Partnertabelle
   n,          Anzahl der tatsächlich ersetzten Zustände
   arenaSize,  Anzahl der Individuen.
   e,          Anzahl der Elternzustände.
   ü,          Anzahl der garantiert überlebenden Zustände
return:
variablen:
   x1, x2, i, p,     index.
begin
   for(i=0; i<e; i++){
         for(p=i+1; x<arenaSize && p<e; p++){
            x1 = x++;
            x2 = x++;
            if(x2<arenaSize)
                  P1[x2] = p; P2[x2] = i;
            if(x1<arenaSize)
              P1[x1] = i; P2[x1] = p;
         }
   }
end
```

Die Auswahl der Partner für die Erzeugung eines neuen Individuums erfolgt, analog zur Auswahl der Eltern, nach einem positionsproportionalen elitären Verfahren. Die Aggressivität der elitären Auswahl kann über die Parameter *arenaSize, seedPoolSize und survivingSeeds* in weiten Grenzen kontrolliert werden. Den beiden besten Individuen wird durch den Algorithmus zu Erzeugung der Partnertabelle die Reproduktion garantiert. Mit abnehmender Qualität werden die restlichen Individuen immer seltener in die Tabelle aufgenommen. Außerdem werden die schlechteren Individuen zuerst von der Rekombination ausgeschlossen, wenn die Anzahl der Crossover pro Zyklus sinkt. Die Parametergruppe *arenaSize, seedPoolSize* und *survivingSeeds* wird im folgenden häufig mit *Arena(A,E,Ü)* abgekürzt (Arena,Eltern,Überlebende).

Position	**m**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Parameter (A,E,Ü)											
10,4,4		[1:1]	[2:2]	[3:3]	[4:4]	[1:2]	[2:1]	[1:3]	[3:1]	[1:4]	[4:1]
10,4,4	4	[1:1]	[2:2]	[3:3]	[4:4]	[5:5]	[6:6]	[1:2]	[2:1]	[1:3]	[3:1]
10,6,2		[1:1]	[2:2]	[1:2]	[2:1]	[1:3]	[3:1]	[1:4]	[4:1]	[1:5]	[5:1]

*Tabelle 2−16 Beispiel für die Partnertabelle mit den Parametern (AEÜ)=(arenaSize, seedPoolSize,survi−vingSeeds), wenn die Anzahl der Crossover gleich **m** ist. Ohne Angabe für* m *ist m = arenaSize − survi−vingSeeds. [A:B] bedeutet, daß Individuum A die Funktion des ersten Elternteils und B die des zweiten im Crossover übernimmt, [A:A] bedeutet, daß A unverändert auf diese Position kopiert wird.*

2.9.6 Die Zustandsdifferenz

```
Algorithmus: Bilde die Zustandsdifferenz
parameter:
   S1 ,S2      zwei Zustände (Individuen)
   D           Differenz, Tabelle mit den Elementaroperationen die S1 in S2 verwandeln.
return:
variablen:
begin
   für jede Position p in S1 und S2
      wenn S1[p] ungleich S2[p]
         wenn S1[p] unbelegt und S2[p] belegt
            push(D, addiere S2[p] an Position p)
         sonst wenn S2[p] unbelegt und S1[p] belegt
            push(D, entferne S1[p] an Position p)
         sonst
            push(D, addiere S2[p] an Position p)
            push(D, entferne S1[p] an Position p)
end
```

Die Differenz diff(***S₁*** ,***S₂***) zwischen zwei Individuen ist der Aktionspfad, der ihre Zustände verbindet. Die Differenz besteht aus den $T^{\pm}$ Operationen, die notwendig sind, um den einen Zustand in den anderen zu verwandeln. Dabei werden jeweils die T^{+} und $a(T_E^{+})$ Operation für jede unterschiedliche Aminosäurepo−sition bestimmt. Die Distanz zwischen zwei Individuen dist(***S₁*** ,***S₂***) ist die Länge der Differenz, d.h. die Anzahl ihrer Aktionen.

$$\mathrm{dist}(S_1,S_2) \;=\; \mathrm{len}(\mathrm{diff}(S_1,S_2)) \qquad (2-16)$$

Die maximale Länge der Differenz für ein Protein ist daher

$$\mathrm{dist}_{max}(S_1,S_2) \;=\; 2\,a_{as} \;-\; 2 \qquad (2-17)$$

2.9.7 Der Reparaturalgorithmus

Wenn Teile der Differenz auf ein drittes Individuum angewendet werden, kann das neu entstehende Indivi−duum in einem illegalen Zustand sein. Das Individuum ist dann illegal, wenn die Protospinsysteme des alten Individuums und des neuen Individuums die gleichen Peaks verwenden. Da die Peaks nur entsprechend ihrer Entartung verwendet werden dürfen, muß, entsprechend der Argumentation in Kap. 1.4.2, eines der kollidierenden Protospinsysteme entfernt werden.

Falls zwischen den Genen des kopierten und injizierten Teils ein Resourcenkonflikt vorliegt, werden die neu injizierten Gene mit Priorität behandelt und die konkurrierenden Gene des kopierten alten Teils entfernt,

auch wenn sie außerhalb des injizierten Bereichs liegen. Diese Konfliktauflösung wird für alle Crossover−Operatoren verwendet. Dadurch hat das neue Individuum meist weniger Gene als jeder der Elternteile und meist auch weniger als das ersetzte Individuum.

Durch einige Zyklen im RANDOM−Algorithmus werden diese Leerstellen aber sofort gefüllt. Die neuen Individuen erreichen erst danach eine Qualität, die mit der Qualität der ungestörten Individuen vergleichbar ist. Das Individuum befindet sich im Zeitraum vom Crossover, bis es wieder Anschluß an die restliche ungestörte Gruppe erhält, im Crossover−Schock. Während des Crossover−Schocks haben die Individuen eine Qualität weit außerhalb des Qualitätsbereiches der ungestörten Individuen.

Die erwünschte Qualitätsverbesserung durch das Crossover tritt erst nach Überwinden des Crossover−Schocks ein.

2.10 Implementierung der Crossover Operatoren

2.10.1 Die Elemente der Crossover Operation

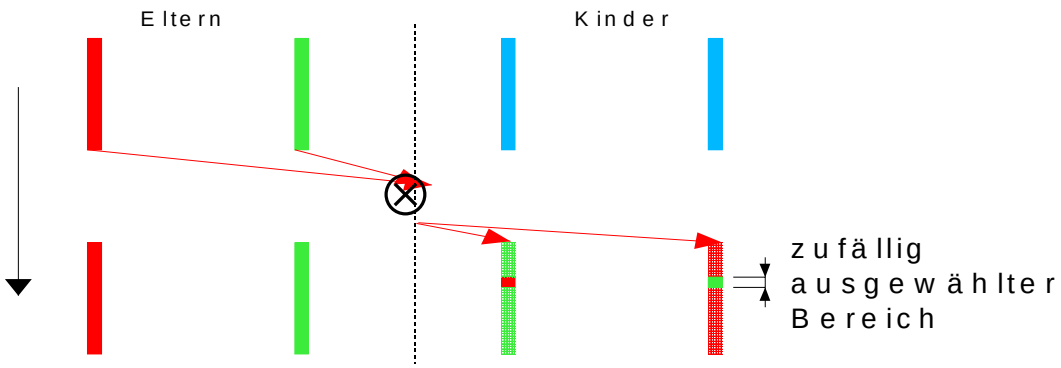


Abbildung 2−6 Crossover Operation des GENETISCHEN Algorithmus.

Klassische Implementierungen [Michalewicz92,S.105] des genetischen Algorithmus verlangen, daß der Crossover Operator ohne Kenntnis des Optimierungsproblems arbeiten soll. Dazu wird die Repräsentation des Individuums in einen Bitvektor konvertiert. Die eigentliche Crossover Operation vertauscht dann Teile der Bitvektoren und anschließend wird der Bitvektor wieder in die Repräsentation des Individuums konvertiert.

Diese Implementierung kann schon für kleine Probleme, die weniger als 30 Variablen besitzen, ungeeignet sein [Michalewicz92, S.100 +105; Goldberg89]. Statt dessen wurden von ihnen die natürlichen Einheiten der Repräsentation (Fließkommazahlen oder Städte) eines Individuums als tauschbare Objekte im Crossover verwendet. Für eine numerische Gleichung sind dies Fließkommazahlen oder für das TSP−Problem Städte oder die Wege zwischen den Städten. Diese Implementierung ist in den meisten Fällen sowohl schneller als auch besser zu implementieren und zu analysieren als die Variante mit Bitvektoren.

In dieser Implementierung werden daher die Protospinsysteme oder die Operationen, die sie bewegen als vertauschbare Elemente für den Crossover−Operator verwendet. Dies trifft sich auch in natürlicher Weise mit den Anforderungen des RANDOM Algorithmus.

2.10.2 Aufteilung der Arena

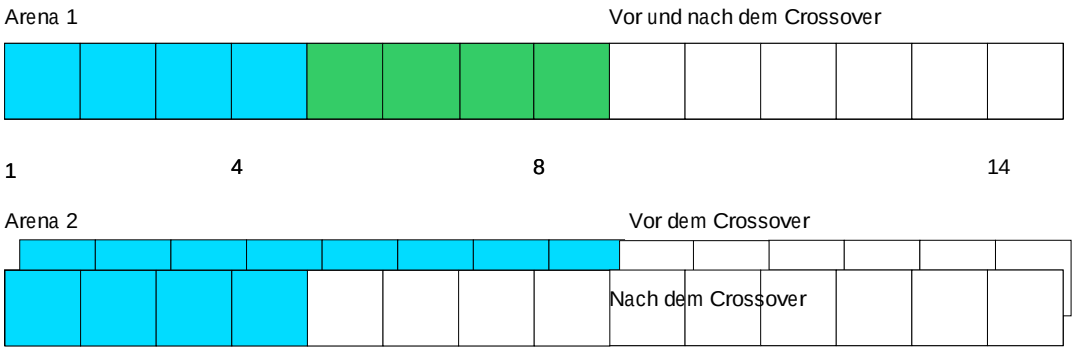


Abbildung 2−7 :Bereiche der Arena bei verschiedenen Konfigurationen (A,E,Ü) vor und nach einem genetischen Zyklus. oben: Arena 1=14,4,8 unten: Arena 2= 14,8,4. blau: Reproduktive (Eltern), grün: Überlebende, die nicht reproduktiv sind, weiß: Kindszustände.

In der unteren Darstellung ragt die Zone der Nachkommen in den Bereich der Eltern. Der blaue Bereich im Hintergrund ist daher vor dem Crossover größer als nach dem Crossover. Nur die ersten vier Eltern überdauern die Crossover−Phase!

Für das Crossover wird die Arena durch die Parameter (`genArenaSize`,`seedPoolSize`,`survivingSeeds`), abgekürzt (A,E,Ü)[3], in mehrere Zonen aufgeteilt. Diese Einteilung beeinflußt die Wirkung des Crossovers auf die Population und damit auf die genetische Zusammensetzung der Arena.

Da die Arena vor dem Crossover nach der Qualität sortiert wird, befinden sich die Individuen mit der niedrigsten Qualität am Ende der Arena während sich die zukünftigen Eltern auf den Positionen mit dem Index [1 ... `seedPoolSize`] befinden. Im Crossover werden die Zustandsobjekte im Bereich [`survivingSeeds+1`... `arenaSize`] durch die neuen Nachkommen der Eltern ersetzt. Dazwischen [`seedPoolSize+1` ... `survivingSeeds+1`] befindet sich die Zone der Individuen, die zwar nicht an der Reproduktion teilnehmen, aber auch nicht durch das Crossover ersetzt werden. Wenn `seedPoolSize` kleiner oder gleich `survivingSeeds` ist, stoßen die Bereiche der Eltern und der Nachkommen direkt aneinander. Außerdem darf `survivingSeeds` auch größer als `seedPoolSize` sein. Dann werden einige der Eltern durch die Crossover−Produkte überschrieben.

Die Implementierung sorgt dafür, daß die Zustände erst dann überschrieben werden, wenn sie an allen geplanten Crossover−Aktionen teilgenommen haben.

3 Abkürzung für (Größe der **A**rena, Anzahl der **E**ltern, Anzahl der **Ü**berlebende)

2.10.3 Der generalisierte Crossover−Operator

```
Funktion: Crossover
A) wähle n Elternzustände (S^E_1 .. S^E_n) aus. (Quellzustände)
B) wähle r zu ersetzende Zustände(Sz_1 .. Sz_r) aus. (Zielzustände)
   loop für jeden Zielzustand Sz_i
C)    wähle ein Elternpaar P(S^E_1,S^E_2) aus.
D)    initialisiere den Zielzustand Sz_i.
E)    wähle das zu übertragende Genmaterial d in den Eltern aus.
F)    kopiere die neuen Gene in den Zielzustand.
G)    repariere etwaige Fehler im Zielzustand.
   end loop
end
```

Algorithmus 2−1 Der verallgemeinerte Crossover−Operator

Um einen geeigneten Crossover−Operator zu finden, wurde die Crossover−Operation in die sieben formalen Schritte A−H auf geteilt.

- Schritt A. Auswahl der Elternzustände. S_i^E

  Die Eltern werden nach einem positionsproportional elitären Schema ausgewählt. Die besten *n* Zustände sind die Eltern der neuen Zustände.

- Schritt B. Auswahl der Zielzustände. S_i^Z

  Auch die Zielzustände werden nach einem positionsproportionalen elitären Schema ausgewählt. Die schlechtesten *m* = `arenaSize-survivingSeeds` Zustände werden ersetzt, wenn sie schlechter als $Q_{schwelle}$ = `genQThreshold*Q(`$\mathbf{s}_i$`)` des besten Individuums sind. Wenn keine Individuen unter dieser Schwelle liegen, erzwingt der Algorithmus ein Crossover und setzt `genQThreshold` herauf. Dadurch steigt die Wahrscheinlichkeit, daß im nächsten Zyklus ein normales Crossover stattfindet.

- Schritt C. Wähle das Elternpaar für jeden neuen Zielzustand aus:

  Die Elternpaare werden nach einem moderat elitären Schema aus der Menge der Elternzustände gewählt. Dabei gelten 3 Kriterien für die Paarbildung.

  1. Bessere Elternzustände werden häufiger ausgewählt.
  2. Schlechtere Elternzustände sollen trotzdem mindestens einmal ausgewählt werden, wenn dadurch nicht Punkt 1 verletzt wird.
  3. Das Schema soll etwaige Asymmetrien der Operatoren ausgleichen, falls die Operatoren nicht kommutativ sind.

Punkt 3 erfordert, daß jedes Elternpaar doppelt verwendet wird und in der zweiten Kombination die Rollen tauscht. Ein Crossover erzeugt daher immer zwei Kinder. Punkt 1 wird durch eine positionsproportionale Auswahl der Eltern erfüllt.

- Schritt D: Initialisiere den Zielzustand (lösche alle Spuren des vorherigen Zustandes).

  Der Zielzustand kann mit drei Zuständen initialisiert werden. Jedes der Elternteile oder der alte Zustand kann in den neuen Zustand kopiert werden. Auf die vierte Möglichkeit, die Erzeugung eines neuen Zustandes mittels Zufallsgenerators, wurde verzichtet, da auf diese Weise kein konkurrenzfähiger Zustand erzeugt werden kann, da die Wahrscheinlichkeit, eine Zuweisung durch "würfeln" zu finden, die eine vergleichbar gute Nachbarschaftsbeziehung zwischen den angrenzenden Protospinsystemen hat, wie einer der iterativ gewonnenen Elternteile, verschwindend gering ist. Diese Wahrscheinlichkeit sinkt

während der Optimierung immer weiter. Nahe dem globalen Optimum ist sie proportional des Kehrwertes der Größe des Lösungsraumes, für Trigger ist das 10^{-208} [siehe Tab. 1−4].

- Schritt E: Wähle das übertragene Genmaterial **d** aus.

  Das übertragene Genmaterial wird mittels Zufallsgenerator ausgewählt. Dieser Schritt ist vom verwendeten Operator abhängig.

- Schritt F: Kopiere die neuen Gene in den Zielzustand.

  Die neu kopierten Gene überschreiben die alten. Wenn die restlichen Gene des Zielzustandes mit den neuen Genen in Konflikt stehen, haben die neuen Gene Vorrang, d.h. die alten Gene werden durch Leerstellen ersetzt. Dieser Schritt ist vom verwendeten Operator abhängig.

- Schritt G: Repariere etwaige Fehler im Zielzustand.

  Wenn es Konflikte zwischen den alten und neuen Genen gab, werden die dabei entstanden Leerstellen mit Teilen der anderen Individuen aufgefüllt. Auch dieser Schritt ist vom verwendeten Operator abhängig.

2.10.4 Untersuchte Crossover Operatoren

Es wurden 30 verschiedene Operatoren untersucht. Die Operatoren lassen sich in 4 Gruppen aufteilen, die sich durch die verwendete Generationenfolge unterscheiden. Die ersten drei Gruppen bestehen aus der Implementierung eines einzigen Algorithmus. Die dritte Gruppe besteht aus einer Familie von Operatoren, bei der die Generationenfolge durch die Kombination von zwei Teilalgorithmen bestimmt wird. Diese Teilalgorithmen werden in allen Kombinationen untersucht.

- **Operator C** entspricht dem *klassischen* Crossover Operator, wie er in [Michalewicz92] definiert ist. Er erzeugt ein Kopie des einen Elternteils (S_1^E) und kopiert einen zusammenhängenden Abschnitt des anderen Elternteils (S_2^E) in diese Kopie. Dafür benötigt er kein Wissen über die Bedeutung oder die Verknüpfung des kopierten Bereichs. Der kopierte Bereich muß sich in den Eltern nicht unterscheiden. Die übertragenen Gene werden durch die blockartige Übertragung in ähnlicher Weise koordiniert wie in **s2**.

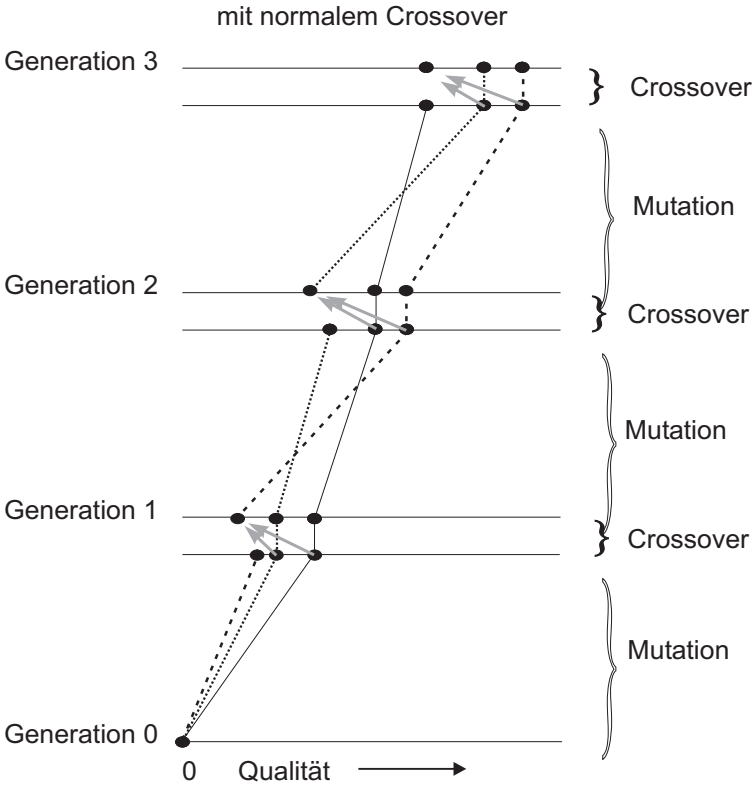


*Abbildung 2−8 Klassische Generationenfolge mit den Crosso− ver Operatoren **C** oder **G** mit einer Arena aus 2 Eltern und 1 Kind. Zeitliche Abfolge von unten nach oben.*

- Der **Maximal Elitäre Operator** (kurz **elitär**) kopiert immer nur das aktuell beste Individuum unverändert auf alle neuen Positionen. Zusammen mit einer geeigneten Konfiguration des Crossover−Planungsalgo− rithmus ersetzt er in jeder Runde alle anderen Zustände durch den besten bekannten Zustand $\mathbf{S}_{\mathbf{best}}$. Die neue Generation besteht daher nur aus Klonen des einen besten Individuums. Dieser Operator wird auch **greedy** genannt.
- Der **Statistische Operator** (oder **noCross**) implementiert das entgegengesetzte Konzept. Er ersetzt niemals einen Zustand. Die Optimierung läuft in den Individuen völlig unabhängig ab, so als würde man **n** unabhängige RANDOM Algorithmen gleichzeitig laufenlassen. Im Gegensatz dazu werden aber alle Läufe gleichzeitig abgebrochen, sobald ein Lauf das Abbruchkriterium erfüllt. Daher wird nur die Laufzeit für **n** Individuen bis zum Erreichen des ersten Optimums in einem der Individuen berechnet. Die Laufzeit entspricht im Mittel der kürzesten Laufzeit in ***n*** unabhängigen RANDOM Läufen multipliziert mit der Arenagröße **n**.
- **Operator G** ist ein konfigurierbarer Operatoralgorithmus, der die restlichen 27 Operatoren implementiert. **G** zerfällt in 2 verschiedene Teilalgorithmen, die Selektion **s** und die Produktion **p**. Jeder davon kann in drei verschiedenen Varianten konfiguriert werden, die mit den Ziffern 1 bis 3 gekennzeichnet werden. Es existieren außerdem drei verschiedene Varianten des **G** Operators, die sich in der Anwendung der selektierten Differenzobjekte unterscheiden.

- Variante **D** zerstört die differierende Region im Zielzustand. Alle Operationen werden als T^- ausgeführt.
- Variante **A** berücksichtigt bei der Differenzbildung nur T^+ / T^- Paare und einzelne T^-, für die es keine korrespondierende T^+ Operation mit gleicher Aminosäureposition gibt. Von jedem T^+ / T^- Paar wird aber nur die T^+ Operation ausgeführt.

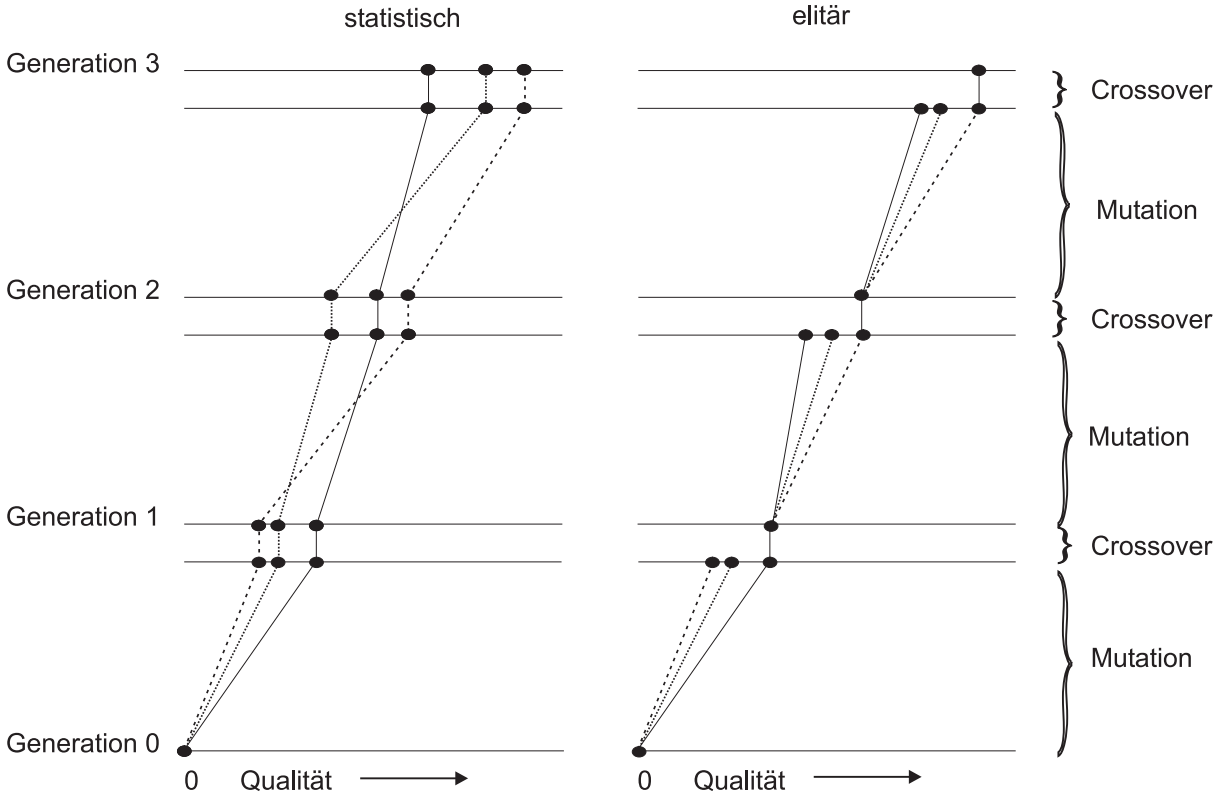


*Abbildung 2−9 Schematische Darstellung der Generationenfolge mit den Crossover Operatoren **Elitär** und **Statistisch**. Arena mit 3 Zuständen.*

- Variante **N** führt T^+ und T^- Operationen auf dem Zielzustand aus. Zusätzlich wird ein Teil (50%) der T^+ Operationen als T^- ausgeführt.

Variante **A** wurde letztendlich für die Auswertung des Trigger−Proteins verwendet und entspricht einem klassischen Crossover, wenn mehrere zusammenhängenden Abschnitte übertragen werden, die immer einem differierenden Bereich entstammen.

2.10.4.1 Die Selektionsalgorithmen **s** *des Operators* **G**

In allen **s**−Algorithmen wird der Kopiervorgang abgebrochen, wenn nach **4*n** Versuchen noch nicht **n** kopierbare Elemente gefunden wurden. Die Algorithmen kopieren dann nur die bisher gewählten Elemente. Die **s**−Algorithmen erzeugen aus den Differenzobjekten Δ die entsprechenden *reduzierten* Differenzobjekte **d**, indem sie einen Teil der Objektdifferenz zufällig auswählen.

- Algorithmus **s1**:

Kopiert **n** einzelne Elemente aus der Zustandsdifferenz Δ in ein neues Differenzobjekt **d**. Die kopierten Stellen werden zufällig ausgewählt und müssen keine Nebenbedingungen erfüllen.

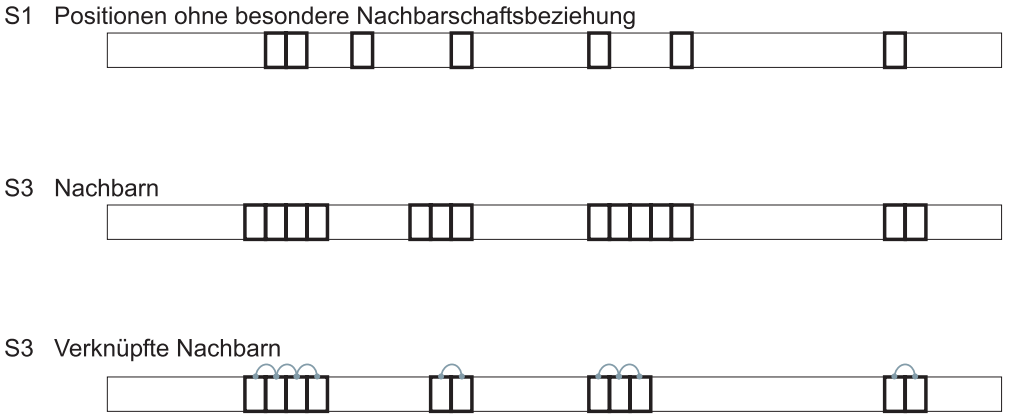


*Abbildung 2−10 Beispiele für die Auswahl der Positionen durch die Selektionsalgorithmen **s1** bis **s3** des Crossover Operators **G** anhand von drei Sequenzen. Die hervorgehobenen Rechtecke markieren die ausgewählten Positionen in der Sequenz für eine Zustands−sequenz Δ.*

- Algorithmus **s2**:

  Kopiert **n** einzelne Elemente aus der Zustandsdifferenz Δ in ein neues Differenzobjekt **d**. Dabei werden Strecken aus je vier aufeinander folgenden Elementen kopiert, bis insgesamt **n** Elemente übertragen wurden. Die Startpunkte der Stücke werden zufällig ausgewählt und die Strecke kann kürzer sein, wenn auf die gewählte Startposition nur weniger als 4 gefüllte Positionen folgen. Wenn nach der Auswahl einer Strecke noch nicht **n** Elemente ausgewählt wurden, wird ein neuer Startpunkt gewählt und die nächste Strecke kopiert.

- Algorithmus **s3**:

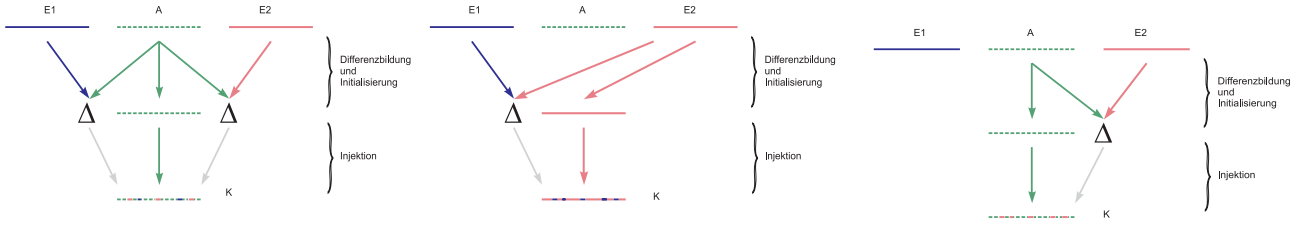
  Algorithmus **s3** unterscheidet sich von **s2,** indem die vier zusammenhängenden Elemente eine Strecke miteinander verknüpfter Spinsysteme bilden müssen. Jede Strecke besteht aus mindestens 2 und maxi−mal 4 Nachbarn. Die Strecke startet bei einer zufällig gewählten Position und wird dann mit dem i+1 Nachbarn verlängert, wenn eine Verknüpfung zur nachfolgenden Position existiert.

2.10.4.2 Die Produktionsalgorithmen p des Operators G

Die Produktionsalgorithmen wählen aus den Differenzobjekten **d** den Teil aus, der auf die Kindszustände angewendet werden soll. Die Wahl des Produktionsalgorithmus bestimmt, wie der neue Zustand initialisiert wird und welche Teile aus **d,** der reduzierten Differenz Δ, in den neugebildeten Zustand injiziert werden.

Jeder **p** Algorithmus zerfällt in einen Teil, der vor dem **s** Algorithmus und in einen zweiten, der danach ausgeführt wird.

Im ersten Teil werden die Elternzustände für das Differenzobjekt Δ ausgewählt. Teil zwei wählt die Initialisierung für den neuen Zustand aus. Außerdem wählt er die Mischung und die Übertragungsweise der Elementaroperationen aus den reduzierten Differenzobjekten **d**.



p1 **p2** **p3**

*Abbildung 2−11 Schematische Darstellung der Produktionsalgorithmen **p1** bis **p3**. Die Produktion startet mit den Elternzuständen E1 und E2 und dem zukünftigen Kindszustand A, der noch die alte Zuweisung enthält und durch K ersetzt wird. Am Ende des Crossovers enthält der Kindszustand K das neu produzierte S_i .*

- Algorithmus **p1**:

  P1 berechnet zwei Differenzobjekte mittels Δ_1 = **diff**(S^{alt} , S_1^E) und Δ_2 = **diff**(S^{alt} , S_2^E)

  Der alte Zustand wird *nicht* durch einen neuen ersetzt. In den Zielzustand wird dann je die Hälfte von **d1** und **d2** injiziert.

- Algorithmus **p2**:

  P2 berechnet ein Differenzobjekt mittels Δ = **diff**(S_1^E , S_2^E). Der alte Zustand wird durch S_1^E ersetzt und in den neuen Zielzustand wird dann **d** injiziert.

- Algorithmus **p3**:

  P3 berechnet ein Differenzobjekt mittels Δ = **diff**(S^{alt} , S_2^E). Der alte Zustand wird *nicht* durch einen neuen ersetzt. In den neuen Zielzustand wird dann **d** injiziert.

S = Selektion **P = Produktion**	**s1** einzelne Elemente.	**s2** je bis zu 4 aufeinander folgende Elemente.	**s3** je bis zu 4 Elemente, wenn sie verknüpft sind.
p1 $S^{neu} = S^{alt}$ + **diff**(S^{alt} , S_1^E) /2 + **diff**(S^{alt} , S_2^E)/2	Gs1p1	Gs2p1	Gs3p1
p2 $S^{neu} = S_1^E$ + **diff**(S_1^E , S_2^E)	Gs1p2	Gs2p2	Gs3p2
p3 $S^{neu} = S^{alt}$ + **diff**(S^{alt} , S_2^E)	Gs1p1	Gs2p3	Gs3p3

*Tabelle 2−17 Die Kurzbezeichnungen der Operator G Varianten. Aus den Kurzbezeichnungen werden mit den Subvarianten **D**,**N** und **A** die Langnamen der Operatoren. Beispiel: Gs1p1 wird zu Gs1p1A, Gs1p1D Gs1p1N.*

Theoretisch sollte der Operator die kürzeste Optimierungszeit ermöglichen, bei dem je Optimierungsschritt ΔQ maximal ist, d.h. bei dem sich die Individuen in jedem Optimierungsschritt am meisten verbessern.

Der Operator Gs3p2A verwendet nur die Gene der schon optimierten Individuen, die einen Qualitätsvorsprung gegenüber den anderen Individuen haben. Er initialisiert den Startzustand mit den Genen des einen Elternteils und injiziert in dieses die Gene, die es vom anderen Elternteil unterscheiden, d.h. die Gene durch die das eine Elternteil besser als das andere ist.

Theoretisch sollte daher Gs3p2A der optimale, schnellste Operator sein.

Kapitel 3 Parameterabschätzung

Um die einzelnen Algorithmen optimal zu betreiben, müssen die Parameter in ihren Wertebereichen auf ihre Empfindlichkeit getestet werden. Für die Testrechnungen wurden theoretische Spektren aus einer Frequenzliste aus der Literatur erzeugt. Die Bedeutung der einzelnen Parameter wurde im vorhergehenden Kapitel erklärt. Eine Zusammenfassung der Parameternamen und ihrer Bedeutung ist im Anhang aufgeführt.

3.1 Testverfahren

Im folgende Abschnitt wird der Einfluß der frei wählbaren Parameter des RANDOM Algorithmus auf die Geschwindigkeit der Optimierung untersucht. Der Parameterraum ist mindestens 12−dimensional und kann daher nicht vollständig untersucht werden. Unter der Voraussetzung, daß die einzelnen Parameter im praktisch nutzbaren Wertebereich voneinander unabhängig sind, sollte die folgende statistische Untersuchung gültig sein. Die Konvergenz zum globalen Maximum ist nur dann garantiert, wenn minTTL und maxTTL auf den Maximalwert, qThreshold auf 0.0 und maxIdleLoops auf unendlich gestellt werden. Selbst dann ist die Konvergenz nur für eine unendlich langsame Erhöhung der Qualitätsschwelle garantiert. Für jede andere Konfiguration ist die Konvergenzwahrscheinlichkeit immer kleiner 1.0. Für praktisch auftretende Probleme können die Parameter so eingestellt werden, daß die Optimierung innerhalb endlicher Zeit (<1d) konvergiert.

Tabelle 3−1 faßt die Standardparameter für die folgenden Untersuchungen zusammen. Die Parametrisierungen benutzten diese Standardkonfiguration und variieren nur die jeweils untersuchten Parameter.

Die Geschwindigkeit, mit der ein Optimierer konvergiert, ist ein direktes Maß für die Qualität seiner Konfiguration. Da die Laufzeit weitgehend linear von der Anzahl der RANDOM Zyklen abhängt, wird im folgenden die Laufzeit und die Zyklenzahl synonym gebraucht. Eine Ausnahme bildet der Abschnitt über den Berechnungscache. Da außerdem die Zyklenzahl, im Gegensatz zur Rechenzeit, unabhängig von der Art der CPU ist wird die Zyklenzahl als Maß für die Geschwindigkeit der Konfiguration verwendet.

Jede Konfiguration wurde auf ihre Laufzeit und die Konvergenz zum globalen Optimum untersucht.

Parameter	Wert	Parameter	Wert
Aufgabe:	**"Chey50"**		
Protein:	Chey50	Startzustand:	(leer)
Cache:	default	AS−Positionsstrafen:	keine
Net:	inner.outer	Positionsvorwahl durch AS−Typen	Ja
Distanz zum Nachbar (CA):	1.00		
Optimierer	**(RANDOM)**		
Profil:	mix1v2	qToleranz, decay	0.9 0.0005
maxLoops	10^6	TTL (min,max)	2, 3
maxIdleLoops	5000	maxTime [min]	20

Tabelle 3−1 Standardparameter für die statistische Untersuchung des RANDOM Algorithmus

3.2 Testdaten: Protein Chemotaxis Y

Chemotaxis Y (CheY) gehört zu einer Proteinfamilie in *eschericha coli*, die die Chemotaxis des Bakteriums reguliert. Die Funktionsweise des Proteins ist heute bis hinunter zur molekularen Ebene bekannt [Stock93, Volz93]. CheY wurde ausgewählt, da die CA und CB Frequenzen bekannt sind und das Protein nicht mit "Trigger" verwandt ist.

CheY ist Teil eines zweigeteilten Regulierungssystems mit einem Sender und Empfänger. Der Sender ist eine Histidyl−Kinase, die sich unter dem Einfluß eines chemischen Umweltreizes verändert. Durch den chemischen Reiz ändert sich der Phosphorylierungsgrad der sensitiven Region. Dies löst eine Veränderung der Expression des regulierten Gens oder eine andere physiologische Reaktion entsprechend der regulierten Signalkette aus.

CheY bindet sich in seiner phosphorylierten Form an den Motor der bakteriellen Geißeln (Flagellaten) und aktiviert sie dadurch. Damit erreicht es eine Veränderung der Schwimmrichtung, d.h. eine Reaktion auf die Konzentration der chemischen Komponente. Die dephosphorylierte Form ist inaktiv.

Die Aminosäuresequenz und die Frequenzliste für die theoretischen Spektren wurden aus [Franklin94] entnommen. Fehlende Frequenzen wurden mit Werten aufgefüllt, die den Typen der Aminosäuren entsprechen [Grzesiek93].

CheY hat 129 Aminosäuren. Die erste Aminosäure ist aufgrund von hydrolytischem Abbau nicht nachweisbar. Die Sequenz in 1 Buchstaben Notation ist:

1− 50	MADKE LKFLV VDDFS TMRRI VRNLL KELGF NNVEE AEDGV DALNK LQAGG
51−100	YGFVI SDWNM PNMDG LELLK TIRAD GAMSA LPVLM VTAEA KKENI IAAAQ
101−129	AGASG YVVKP FTAAT LEEKL NKIFE KLGM

Tabelle 3−2 Aminosäuresequenz von CheY

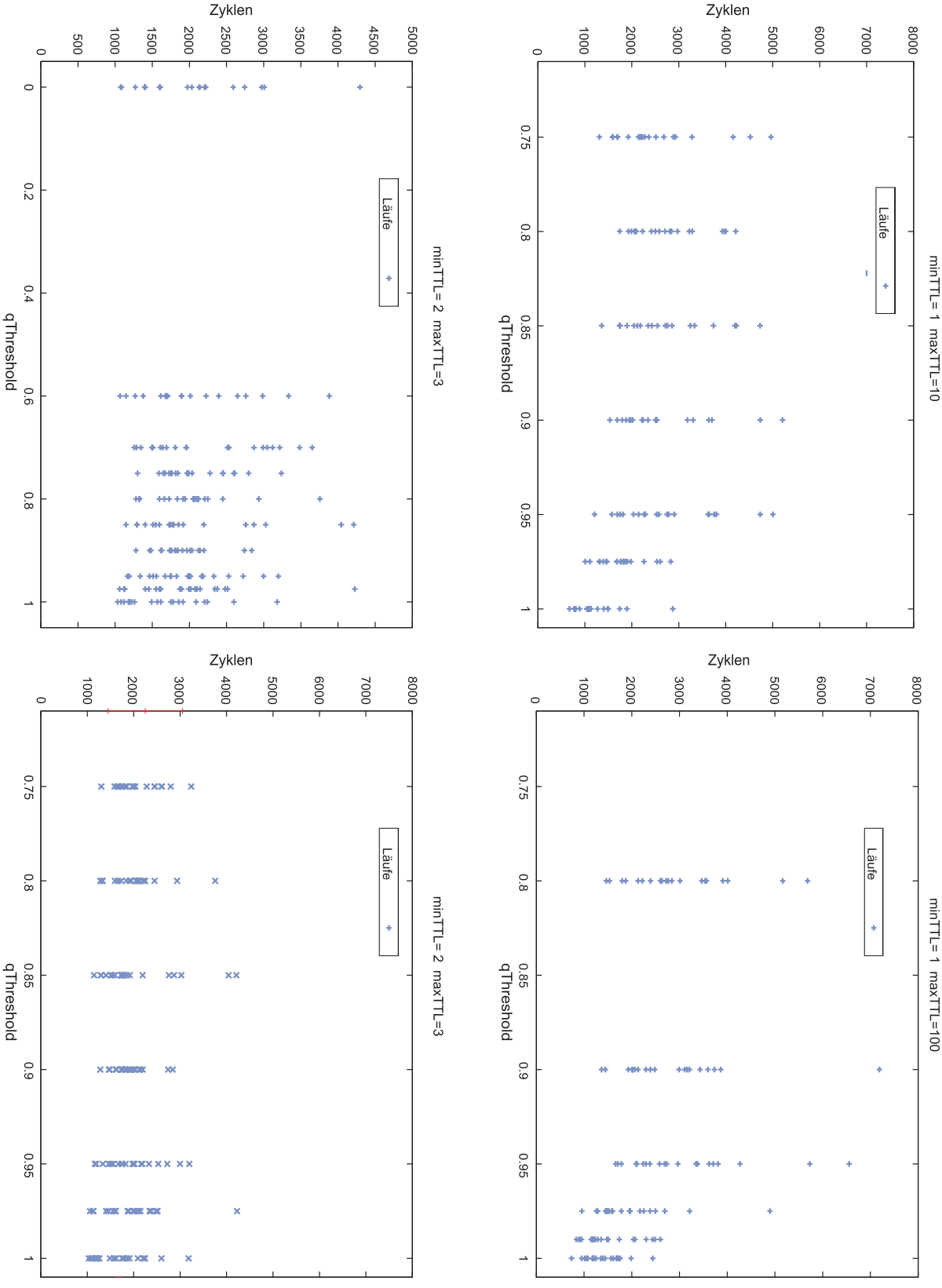


Abbildung 3−1 Abhängigkeit der Laufzeit von der Qualitätsschwelle (qThreshold) bei variierender minima− ler und maximaler Lebenszeit (minTTL und maxTTL).

3.3 Parameterabschätzung für RANDOM

3.3.1 maxLoops und maxTime

maxLoops und *maxTime* sind Teil des Abbruchkriteriums. Sie müssen so gewählt werden, daß der Optimierungslauf nicht vorzeitig beendet wird. Sie dürfen keinen Einfluß auf den Verlauf der Optimierung haben.

3.3.2 maxIdleLoops (Maximale Zyklen ohne Verbesserung)

maxIdleLoops muß auf das Optimierungsproblem abgestimmt werden. Der Parameter ist schwach mit *qThresholdDecayRate* und *maxLoops* verknüpft. Eine großes *maxIdleLoops* verlängert die Laufzeit direkt, da *qThreshold* erst herabgesetzt wird, wenn *maxIdleLoops* mindestens einmal abgelaufen ist. Diese Mindestlaufzeit beginnt nach jeder Verbesserung erneut. Andererseits verbessert ein höheres *maxIdleLoops* die Suchbreite und ermöglicht dem Optimierer, aus tieferen Suboptima zu entkommen. In gleicher Weise verbessert es die Wahrscheinlichkeit aus Suboptima, die mit dem globalen Optimum nur durch einen Aktionspfad mit niedriger Wahrscheinlichkeit verbunden sind, zu entkommen. Daher wird die Umgebung des einzelnen Zustandes mit höherem *maxIdleLoops* besser durchsucht.

maxIdleLoops hat daher direkten Einfluß auf die Zyklenzahl. In reinen *hill−climbing* Problemen kann es niedriger eingestellt werden als in Problemen mit rauher, trügerischer Hyperfläche. Für große Zustandsräume muß *maxIdleLoops* höher eingestellt werden, damit die Umgebung des aktuellen Zustandes hinreichend durchsucht werden kann. Der optimale Wert für *maxIdleLoops* muß daher für jedes Problem extra mit einem Zyklen/Qualitäts Graph bestimmt werden.

Für die Optimierungen in der Parameterabschätzung mit CheY wurde *maxIdleLoops* auf 200000 gesetzt, da *maxIdleLoops* sehr vom Optimierungsproblem abhängt und hier der Einfluß der anderen Parameter untersucht werden soll.

3.3.3 qThreshold

Die Wahl des Startwertes für *qThreshold* beeinflußt die Zyklenzahl nur mäßig, wenn der Wert innerhalb des Bereichs [0.75 − 0.95] liegt. In einer reinen *hill−climber* Umgebung ist 0.9999 optimal. Rauhe Umgebungen benötigen niedrigere Werte, z.B. 0.9, damit sie aus den lokalen Optima herausfinden. Siehe Abb. 3−1.

3.3.4 qThresholdDecayRate

qThresholdDecayRate muß wie maxIdleLoops auf das Optimierungsproblem abgestimmt werden. *hill−climber* Probleme können höhere *qThresholdDecayRate* Werte vertragen (bis 0.05) als rauhe Umgebungen mit vielen Nebenoptima. *qThresholdDecayRate* hat direkten Einfluß auf das Abbruchkriterium, aber nicht auf den Verlauf der Optimierung. Ungeeignete *qThresholdDecayRate*'en führen nur zu vorzeitigem Abbruch oder zu unnötig langer Laufzeit. Die Wahl von *qThresholdDecayRate* muß der Optimierung daher genügend Zeit lassen, um das Optimum zu erreichen.

Experimentell wurde der Wert auf 0.0005 festgelegt. Dieser Wert ist geeignet für alle hier untersuchten Optimierungsprobleme.

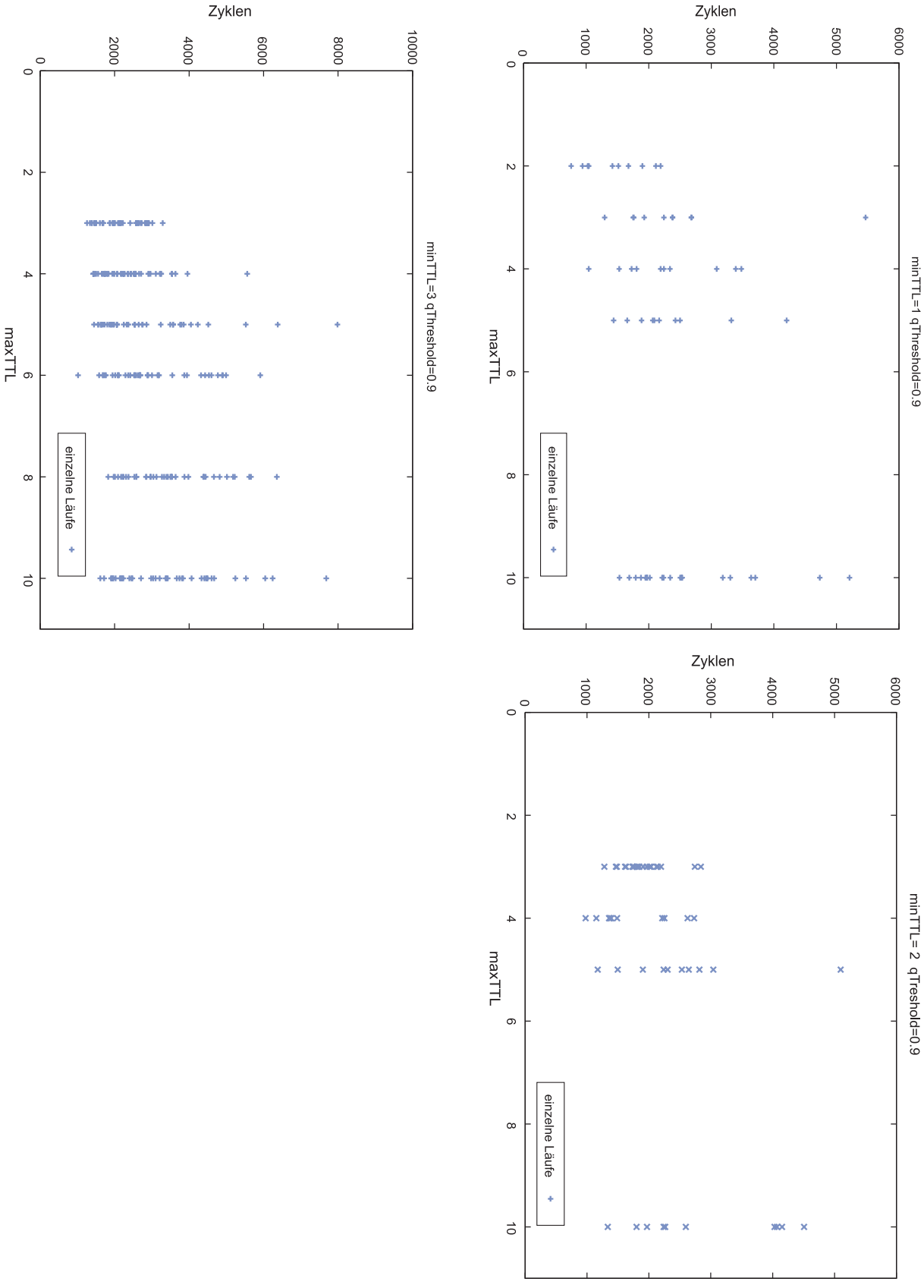


Abbildung 3−2 Abhängigkeit der Laufzeit von der maximalen Lebenszeit (maxTTL)

3.3.5 minTTL

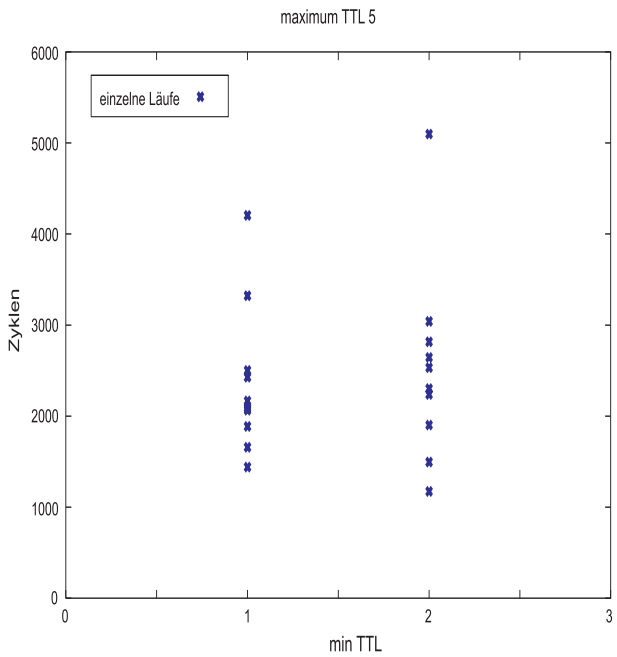


Abbildung 3−3 Abhängigkeit von der minimalen Lebenszeit (minTTL) bei maxTTL=5

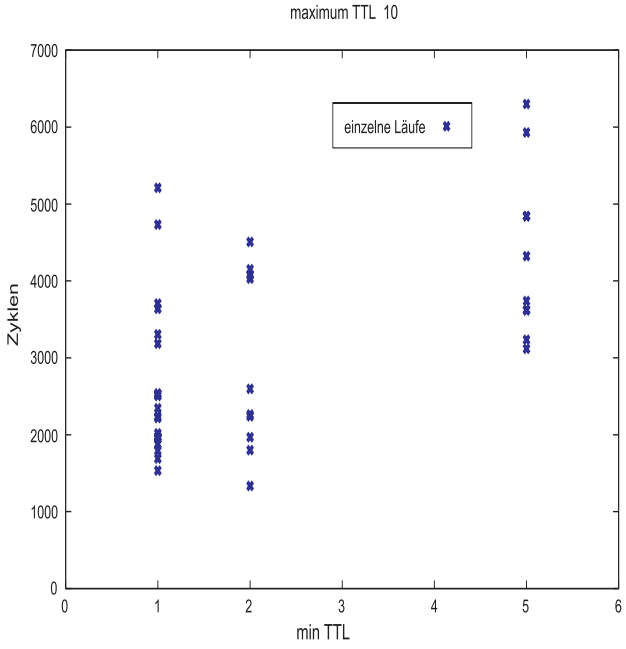


Abbildung 3−4 Abhängigkeit von der minimalen Lebenszeit (minTTL) bei maxTTL=10

Zu kleine Werte für *minTTL* behindern das Entkommen aus lokalen Optima, zu große verlängern die Zyklenzahl, da viele schlechte Suchpfade zunächst zugelassen und später verworfen werden. Daher dauert jeder Fehlversuch mindestens *minTTL* Zyklen. (siehe Abb. 3−3 und 3−4)

Optimale Werte sind 1 bis 3. Der Standardwert ist 2.

3.3.6 maxTTL

maxTTL muß größer gleich *minTTL* sein. Mit zunehmenden Werten streut die Zyklenzahl und die minimale Zyklenzahl nimmt zu. (siehe Abb. 3−2)

Optimale Werte liegen zwischen 2 und 5. Der Standardwert ist 3.

3.3.7 randArenaSize

Die Arenagröße im Random Optimierer[4] hat linearen Einfluß auf die Konvergenzgeschwindigkeit. Die Schleifenzahl ist linear antiproportional zu *randArenaSize*. Der Standardwert (4) wurde so gewählt, daß der RANDOM Algorithmus bei einem bestimmten Testproblem eine vergleichbare Schleifenzahl wie PASTA verbraucht. *randArenaSize* kann frei gewählt werden, da es keinen Einfluß auf die Laufzeit hat (<<1%).

4 Die von der Arenagröße im Genetischen Optimierer unabhängig ist!

3.3.8 Profile

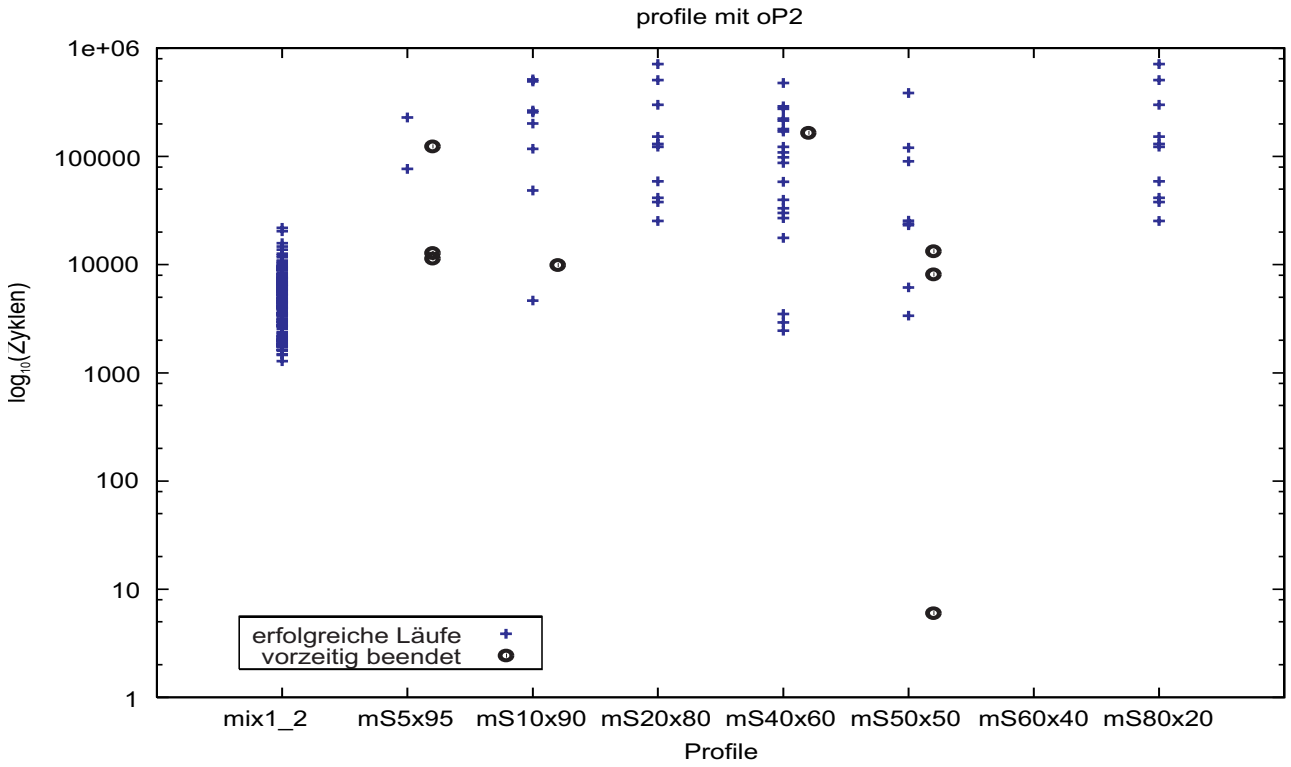


Abbildung 3−5 Abhängigkeit der Zykluszeit vom gewählten Aktionsprofil in logarithmischer Darstellung. Die Symbole für vorzeitig beendete Läufe wurden mit einer kleinen Verschiebung rechts neben die zugehörigen erfolgreichen Läufe projiziert.

Das gewählte Profil hat sehr großen Einfluß auf die Konvergenzgeschwindigkeit (siehe Abb. 3−5). Die einzelnen Profile sind in Tab [2−15] definiert. »`exchange_only`« (nicht dargestellt) konvergiert nur selten (<90%) innerhalb von endlicher Zeit (<10^8 Zyklen). Es bleibt meist in einem niedrigen lokalen Optimum stecken. Schon mit einem kleinen Anteil `swap` konvergieren die Läufe häufiger. Bis 50% nimmt, mit zunehmendem `swap`−Anteil, sowohl der Anteil fehlerhafter Läufe als auch die Laufzeit ab. Danach erhöht sich die Laufzeit wieder. Gleichzeitig streut sie über einen Bereich von bis zu 2 Größenordnungen.

Mit einem `moveRange`−Anteil von 5% bis 20% verringert sich die Laufzeit um bis zu 2 Größenordnungen und die Streuung wird auf eine Größenordnung eingeschränkt.

In den Experimenten mit C21W wurde eine geringfügige Verbesserung der Konvergenz und Laufzeit festgestellt, wenn das Profil einen Anteil `moveSingle` enthält. Diese Aktion wurde speziell zum Auflösen einer typischen `deadlock` Situation in diesen Experimenten zugeschnitten.

Das Standardprofil ist »`mix1_2`«, damit wurden keine fehlerhaften Läufe mehr erzeugt. Die anderen in Tabelle 2−15 aufgeführten Mischprofile führten zu keiner signifikanten Verbesserung oder die Verbesserungen traten nur in einzelnen Testkonfigurationen ein und konnten in anderen nicht wiederholt werden.

Die große Abhängigkeit der Laufzeit von der Profilzusammensetzung ist verständlich, wenn man berücksichtigt, daß jede neue Operation einen neuen, zusätzlichen Graphen im Zustandsraum aufspannt. Die neuen Kanten können Abkürzungen der alten sein, die zwei Zustände verknüpfen, die vorher nur durch

einen längeren Weg aus Elementaroperationen oder mehreren anderen alten Operationen verbunden waren. Wenn der alte Weg durch den verbotenen Bereich unterhalb der Qualitätsschwelle führte oder dort endete, konnte er daher vorher nicht genutzt werden. Die neuen Operatoren durchqueren den Bereich mit niedriger Qualität in einem Schritt und können daher weiter entfernte Bereiche des Lösungsraumes erreichen, als es mit den Elementaroperationen allein möglich ist. `Swap` verkürzt den Weg um 4 Elementaroperationen, `moveRange` verkürzt den Weg sogar noch stärker (siehe Kapitel 2).

3.3.9 Einfluß des Optimierungsproblems

Einige Eigenschaften, die durch das Optimierungsproblem vorgegeben sind, haben auf die Konvergenz und die Zyklenzahl der Optimierung großen Einfluß. Die eine ist die Proteinlänge. Die andere ist die akzeptierte maximale Distanz in der CA/CB−Dimension. Die Proteinlänge kann nicht beeinflußt werden, dagegen ist die maximale CA Distanz innerhalb bestimmter Grenzen wählbar, wenn man die Möglichkeit nutzt, die Spektren gut zu referenzieren. Notfalls kann man die Distanz sogar manuell editieren, d.h. explizit Kanten löschen, die nach externen Erkenntnissen nicht relevant sind.

3.3.10 Maximale Nachbarschaftsdistanz ΔCA

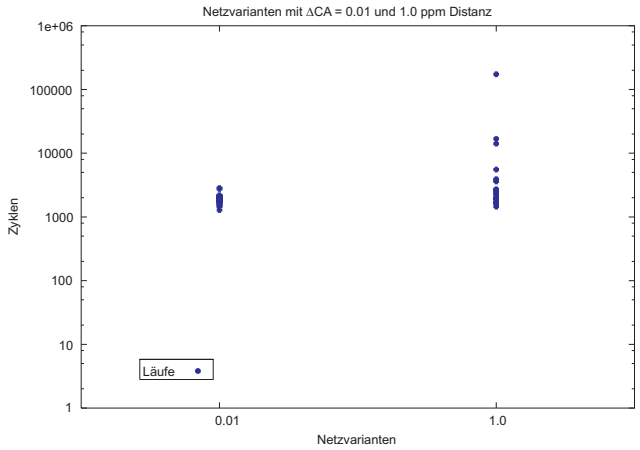


Abbildung 3−6 Abhängigkeit der Laufzeit vom Nachbarschaftsabstand in der CA Dimension (logarithmische Darstellung).

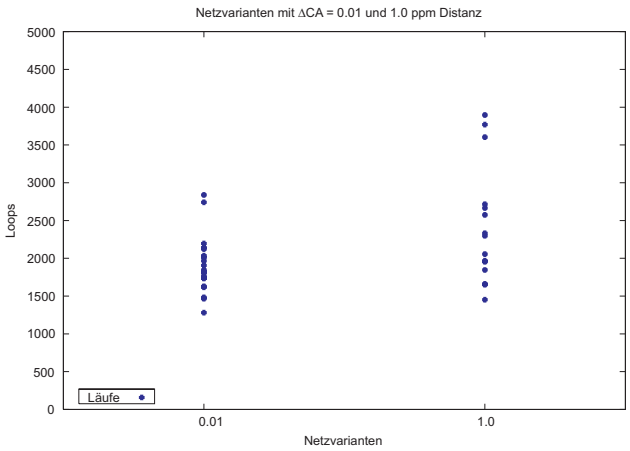


Abbildung 3−7 Abhängigkeit der Laufzeit vom Nachbarschaftsabstand in der CA Dimension (lineare Darstellung).

Die Zykluszahl hängt sehr stark von der maximalen Nachbarschaftsdistanz entlang der CA Dimension ab, da die Anzahl der benachbarten Zustände mit der maximalen Nachbarschaftsdistanz zunimmt (siehe Abb. 3−6 und 3−7). Gleichzeitig nimmt die Streuung der Zyklenzahl extrem zu. Während für 0.1 ppm der Faktor vom bestem zum schlechtestem Durchlauf 2 ist, erreichte er bei 1.0 ppm einen Faktor größer 10, mit Berücksichtigung des Extremwertes sogar einen Faktor von 100 ! Da die kleinste akzeptable Nachbarschaftsdistanz auch von der Frequenzverteilung im Protein und der Qualität des Datensatzes beeinflußt wird, hängt damit die Laufzeit direkt von der Qualität der Spektren und der Referenzierung ab. Eine schlechte Referenzierung verzehnfacht die Laufzeit.

3.3.11 Proteingröße

Entscheidend für die Laufzeit ist die Größe des Zustandsraums, die auch von der Länge des Proteins abhängt (Gl.1−9). Die Laufzeit ist proportional dem Logarithmus der Größe des Zustandsraumes. Die Größe des Zustandsraumes nimmt aber exponentiell mit der Proteingröße zu. Daher ist die Laufzeit ungefähr proportional der Länge des Proteins. (siehe Abb. 3−8 und 3−9)

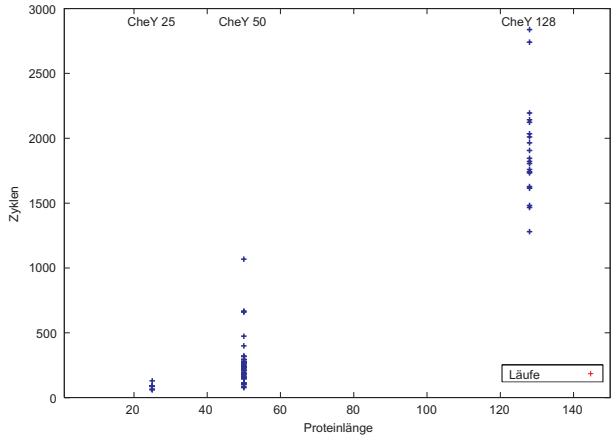


Abbildung 3−8 Abhängigkeit der Laufzeit von der Proteingröße

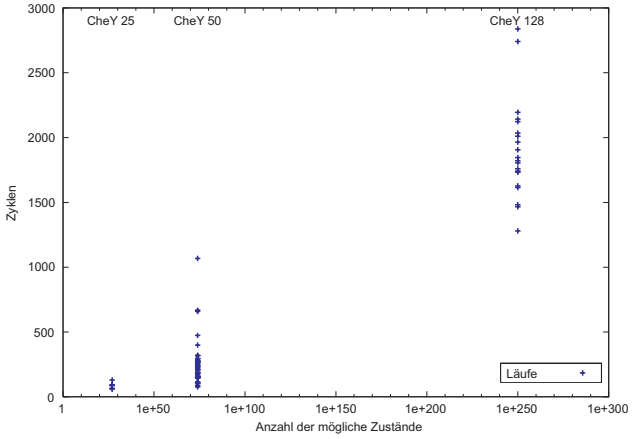


Abbildung 3−9 Abhängigkeit der Laufzeit von der Anzahl der Zustände im Zustandsraum.

Problem	**Länge**	**Konfiguration**	**Protospinsysteme**	**Maximale Anzahl der Zustände.**
				alle Positionen
CheY 25	24	perfekt	24	1e27
CheY 50	49	perfekt	57	1e74
CheY 128	127	perfekt	171	1e256
Trigger, mit His Tag	110	Stufe 1	158	1e219
Trigger, mit His Tag	110	Stufe 2 und 3	162	1e221
Trigger, ohne His Tag	103	Stufe 1	158	1e207
Trigger, ohne His Tag	103	Stufe 2 und 3	162	1e209

Tabelle 3−3 Abschätzung der Größe des Zustandsraums mittels Gl.1−9, wenn man alle Protospinsysteme für alle Aminosäurepositionen berücksichtigt muß.

3.3.12 random (Startwert für den Zufallszahlengenerator)

Die Wahl des Startwertes für den Zufallszahlengenerator ist unkritisch. Gleiche Startwerte führen bei gleicher Konfiguration zu identischen Programabläufen und daher zu gleichen Endzuständen. Der Startwert "time" wählt allerdings immer die aktuelle Systemzeit als Startwert und gilt daher nicht als identischer Startwert.

3.4 Wahrscheinlichkeitsverteilung der Zyklenzahlen für RANDOM.

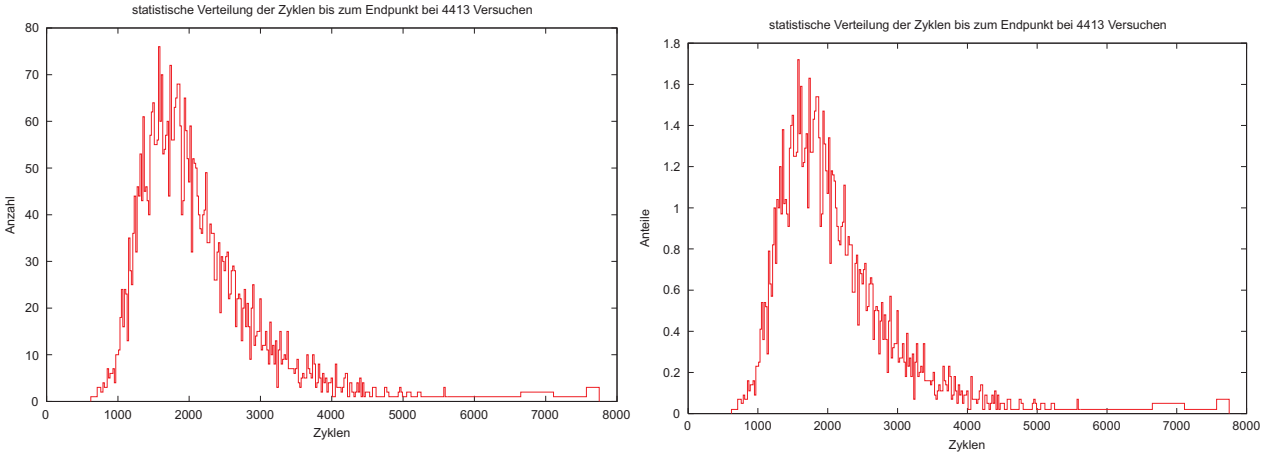


Abbildung 3−10 Statistische Verteilung der Zyklenzahlen für RANDOM in 4413 Einzelläufen.

Die Verteilung der Laufzeiten bis zur Konvergenz mit der Standardkonfiguration wurde mit 4413 Einzelläufen ermittelt. Die theoretische Verteilungskurve kann numerisch exakt nur mittels einer Modellierung der homogenen Markowschen Kette des gesamten Zustandsraumes ***S*** berechnet werden. Dazu ist eine Modellierung der Übergangswahrscheinlichkeiten notwendig. Für CheY erfordert das die Berechnung der Übergangsmatrix Π zwischen den Zuständen im Lösungsraum ***S*** und ist daher praktisch nicht durchführbar.

$$\Pi = \begin{Bmatrix} p_{1,1} & \cdots & p_{1,k} \\ \vdots & \ddots & \vdots \\ p_{k,1} & \cdots & p_{k,k} \end{Bmatrix} \qquad \text{mit} \quad k \approx 10^{208} \quad \text{für Trigger} \qquad (3-1)$$

Die Form der Verteilungskurve kann aber mittels eines sehr einfachen Modells aus verketteten Einzelereignissen erklärt werden. Für eine linear verknüpfte Menge *K* von *n* Zuständen (Kette) sei ein Vektor der Übergangswahrscheinlichkeiten *Q* definiert, in dem $q_{i,i+1}$ die Übergangswahrscheinlichkeit vom *i* zum *i+1* Nachbarn definiert. Die Zyklenzahl $z_{i,i+1}$ ist die Anzahl der Zufallszahlen *x* im Bereich [0 − 1], die man ziehen muß, um ein $x \geq q_{i,i+1}$ zu erhalten. *Z(K,Q)* ist die Anzahl der Zyklen, die für einen vollständigen Lauf vom ersten zum letzten Zustand in K mit den Übergangswahrscheinlichkeiten in *Q* benötigt wird.

Wenn *Q* mit Gl. 3−2 belegt wird, ergibt sich eine Wahrscheinlichkeitsverteilung für *Z(K,Q)*, die der Form der Verteilungskurve in Abb. 3−10 entspricht. Lineare oder konstante Funktionen zur Erzeugung der Übergangswahrscheinlichkeiten ergeben eine symmetrische Gauß'sche Verteilungskurve.

$$\begin{aligned} q_{0,1} &\ll 10^{-4} \\ q_{i,i+1} &= \log_{10}\left(1+\frac{9}{n}\cdot(i-1)\right)\cdot 0.1 \end{aligned} \qquad (3-2)$$

$$Z(K,Q) = \sum_{i=1}^{i=n} z_{i,i+1} \qquad (3-3)$$

3.5 Beschleunigung der Berechnung mittels eines Caches

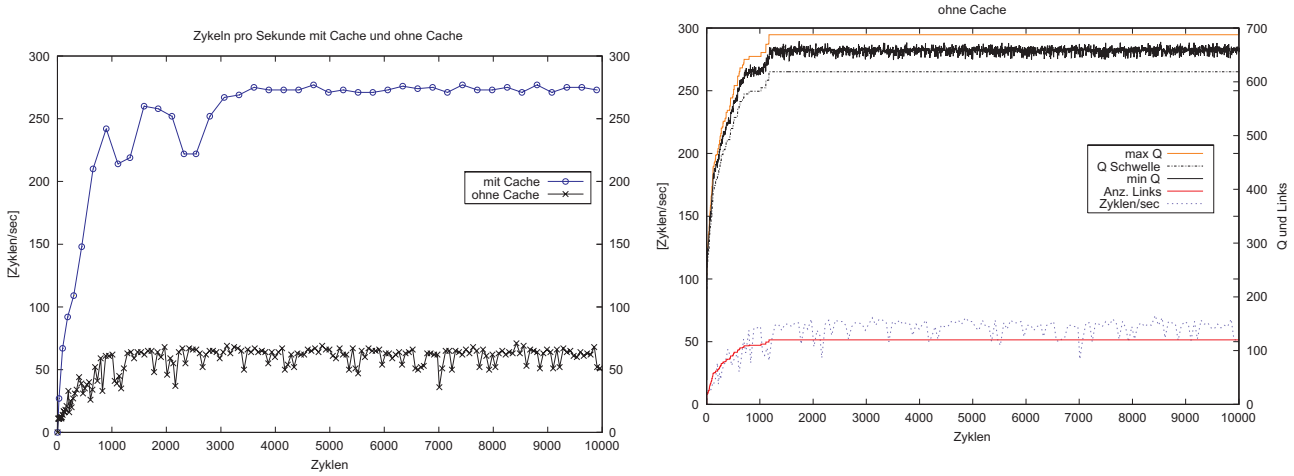


Abbildung 3−11 Beschleunigung der Berechnung durch den Berechnungscache. Links: der Vergleich zwischen der Berechnung mit und ohne Cache. Rechts: Verlauf der Rechnung ohne Cache.

Die Bildung der Link−Protospinsysteme und die Bewertung der AS− und Link−Protospinsysteme verbraucht viel Rechenzeit. Da häufig die gleiche Berechnungen mehrfach innerhalb derselben Optimierung gebraucht werden, werden ohne Cache bereits früher berechnete Werte mehrfach erneut berechnet.

Die Berechnung kann daher beschleunigt werden, wenn einmal berechnete Zwischenergebnisse in einer Tabelle, einem sogenannten Cache, gespeichert und nachgeschlagen werden können. Der Cache beschleunigt die Berechnung, d.h. reduziert die real verbrauchte CPU Zeit, um einen Faktor von mehr als 4.

3.6 Parameterabschätzung für den Genetischen Optimierer

Im Gegensatz zum RANDOM Optimierer sind für den Genetischen Optimierer drei Zyklenzahlen charakteristisch.

- *Mittelwert(Summe(subLoops))*

  Dies ist die gemittelte Summe über alle genetischen Zyklen, die von allen Individuen verbraucht wurde. Die Anzahl der sequentiell verbrauchten Zyklen entspricht der verbrauchten Rechenzeit.

- *Summe(mid(max(subLoops)))*

  Die Anzahl der parallelen Zyklen zeigt, wie schnell die entsprechende Konfiguration ist, wenn alle Individuen auf einem eigenen Prozessor ausgeführt werden. Dabei wird jeweils die Zyklenzahl des Individuums bewertet, das in diesem Mutationsschritt am meisten RANDOM Zyklen verbraucht hat, da dies der Zeit entspricht, die ein parallelisierter Algorithmus warten muß, um den nächsten genetischen Zyklus synchron zu starten. Die Anzahl der parallelen Zyklen ist die Summe der Mittelwerte aller längsten Mutationsschritte je genetischem Zyklus über alle genetischen Optimierungen.

- Die Anzahl der genetischen Zyklen

Die genetischen Zyklen sind für sich alleine genommen kein Kriterium, um die Qualität einer Optimierung zu bewerten. Sie wird daher nur betrachtet, wenn sie eine besondere Einsicht in die Konfiguration gibt. Zusätzlich ist die gemittelte genetische Zyklenzahl problematisch, da der Optimierer innerhalb eines genetischen Zyklus unterschiedlich viel Zeit in RANDOM Zyklen verbrauchen kann. Außerdem kann ein genetischer Optimierer nur am Ende eines genetischen Zyklus eine Optimierung beenden.

3.6.1 Die Ausgangskonfiguration für die Parametrisierung

Die folgende Basiskonfiguration wurde für alle folgenden Berechnungen, bis auf die Bewertung der Crosso−ver Operatoren, verwendet.

Parameter	**Wert**	**Parameter**	**Wert**
Aufgabe:	**"CheY"**		
Protein:	CheY	Startzustand:	(leer)
Cache:	Default	AS−Positionsstrafen:	keine
Net:	Inner.outer	Positionsvorwahl durch AS−Typen	Nein, alle Protospinsysteme können auf jede Position (=oPa)
Distanz zum Nachbar (CA)	1.00		
Genetischer Optimierer	**(GENETIC)**		
Crossover Operator	Gs2p2	crossOverPct	0.15
Arena (A, E, Ü)	32 A, 16 E , 8 Ü	genQToleranz, decay	0.999, 0.05
genCrossoverThreshold	0.9		
maxLoops:	1 000 000	maxTime [min]	3600
maxIdleLoops	15	maxEqualLoops	5
SubOptimierer	**(RANDOM)**		
Profil:	mix1v2	qTolerance, decay	0.9 0.005
maxLoops:	4000	TTL (min,max)	2, 5
maxIdleLoops:	10 000	maxTime [min]	10
Finaler Optimierer	**(RANDOM)**		
Profil:	mix1v2	qTolerance, decay	0.9 0.005
maxLoops	200 000	TTL (min,max)	2, 5
maxIdleLoops	5000	maxTime [min]	20

A,E,Ü = Arenagröße,Eltern,Überlebende

3.6.2 Die Qualitätsunterschiede der Operatoren

Die Beurteilung der Crossover Operatoren kann nicht unabhängig von der restlichen Parametrisierung vor−genommen werden. Gleichzeitig ist eine vollständige Exploration des Parameterraumes zusammen mit den Crossover Operatoren aus Zeitgründen nicht möglich, da eigentlich für jeden Operator eine vollständige Parametrisierung durchgeführt werden muß. Daher wurde zuerst eine akzeptable Standardkonfiguration für alle Operatoren gesucht. Mit dieser Konfiguration wurden dann die Crossover Operatoren beurteilt. Für die besten zwei Operatoren wurde danach die restliche Parametrisierung durchgeführt.

Die Bewertung der Crossover Operatoren erfordert eine andere Konfiguration als die Bewertung aller anderen Parameter. Da zur Bewertung der Crossover Operatoren besonders viele langsame Konfiguratio−nen bewertet werden mußten, wurde ein Problem ausgewählt, das mit RANDOM nur langsam konvergiert, aber mit GENETIC in kurzer Zeit lösbar ist. Außerdem sollte es viele Crossover−Ereignisse ermöglichen und innerhalb kurzer Zeit (~1/2 h) lösbar sein, um eine statistisch hinreichend große Anzahl Rechnungen zu erreichen. Daher wurde nicht das gesamte CheY Protein als Problem verwendet ,sondern nur CheY50, d.h. die ersten 50 Aminosäuren von CheY.

Außerdem wurden nur die Zyklen gemessen, die zum Erreichen einer *Qualitätsschwelle* (99.37% der theoretischen Qualität) verbraucht wurden. Oberhalb dieser Schwelle beginnt, außer bei sehr effizienten Crossover Operatoren, der rein statistische Abschnitt der Optimierung, in dem der Optimierer nach der letz−ten Aktion sucht, die das Individuum zum Qualitätsmaximum führt. Durch die niedrigere Schwelle werden die Messungen um ein Drittel bis zur Hälfte, gegenüber einer Rechnung bis zum theoretischen Endpunkt, beschleunigt, da die statistische Phase der Konvergenz nicht durchlaufen wird. Außerdem ermöglicht es eine gemeinsame Vergleichsmessung mit den schlechten Operatoren, die das theoretische Maximum nicht erreichen. Zum Vergleich zwischen der Messung mit Qualitätsschwelle und der Optimierung bis zum theo−retischen Maximum wurde zusätzlich Operator D in allen Varianten bis zum theoretischen Endpunkt berech−net.

3.6.2.1 Statistische Auswertung

- Konfiguration:

  random: maxLoop= 2k, Schwelle=0.9, minTTL=2, maxTTL=3

  genetisch: Arena (Individuen, Eltern, Überlebende) =8,4,4 crossOverPct=0.5

  Problem: ohne AS−Typen, alle Positionen erlaubt (=oPa) , CheY50

- **Optimierung bis zum theoretischen Endpunkt (Qualität = 261.65) mit Operator D**

	random, sequentiell	**genetisch, statistisch**	**genetisch, elitär**	**genetisch, Operator C**
Zyklenzahl	[5] (183.319 ± 67.000)	120.000	[6] *(63.000 ± 3.000)*	126.000

Operator D			**Selektionsmethode**	
Produktions methode		**s1**	**s2**	**s3**
	p1	124.000	138.000	134.000
	p2	138.000	118.000	106.000
	p3	136.000	144.000	126.000

*Tabelle 3−4 Einzelne Läufe mit dem Operator **D**. Im Gegensatz zu Tab.3−6 wurden die Rechnungen solange fortgesetzt, bis die automatische Endpunkterkennung die Optimierung beendete.*

5 (random, sequentiell) 2 von 5 Versuchen erreichten das Maximum nicht (erreichte Qualität ~ 238).

6 (genetisch, elitär) erreichte den theor. Endpunkt in allen drei Versuchen nicht. Der Optimierer blieb bei einer Quali−tät von ~248 stecken.

- **Optimierung bis zum Erreichen der Qualitätsschwelle mit allen Operatoren**

Die Optimierung wird durchgeführt bis die Qualitätsschwelle von 260, d.h. 99.37% des theoretischen Maximalwertes, erreicht ist.

	random, sequentiell	**genetisch, statistisch**	**genetisch, elitär**	**genetisch, Operator C**
Zyklenzahl	[7] (51.000 ± 19.000)	58.500 ± 38.310	[8] (37.000 ± 7.000)	42.000 ± 11.000

Tabelle 3−5 Gemittelte Laufzeiten der klassischen Crossover Operatoren bis zur Qualitätsschwelle.

Produktionsmethode			**Selektionsmethode**	
		s1 (punktförmig)	**s2 (Nachbarn)**	**s3 (verkn. Nachbarn)**
	p1 **alt + ½ Δ(alt,$E_{1,2}$)**	D 57.000 ± 4.200 N 43.333 ± 6110 A 42.000 ± 5.291	D 53.000 ± 1.400 *N 48.666 ± 6.110* A 56.000 ± 5.656	D 89.000 ± 26.000 N 45.000 ± 4.242 A 50.000 ± 8.000
	p2 **E + Δ(E)**	D **45.333 ± 12.000** N 39.500 ± 3.451 **A 36.000 ± 2.800**	D **44.000 ± 4.300** N 38.500 ± 5.000 **A 36.500 ± 4.434**	**D 41.333 ± 12.000** **N 34.500 ± 4.431** A 38.000 ± 6.928
	p3 **alt + Δ(alt,E)**	D 84.000 ± 19.800 N 42.666 ± 2.309 A 44.666 ± 11.547	D 54.000 ± 8.400 *N 48.000 ± 5.656* A 47.350 ± 2.309	*D 99.000 ± 72.000* N 39.333 ± 4.618 A 39.333 ± 1.157

*Tabelle 3−6 Gemittelte maximale Laufzeiten mit den einzelnen Operatoren aus Tabelle 2−17. Der **beste** und schlechteste Wert jedes Operators sind hervorgehoben.*

Die niedrige Effizienz der klassischen Operatoren in Tabelle 3−5 gegenüber den speziellen Operatoren A und N in Tabelle 3−6 ist offensichtlich. "random,sequentiell" und "genetisch,elitär" können nicht mal für die niedrigere Qualitätsschwelle eine stabile Konvergenz zum Optimum garantieren. "genetisch,statistisch" zeigt die Wirkung der parallelen Berechnung ohne genetischen Austausch zwischen den Individuen.

"genetisch,Operator C" zeigt die Wirkung eines blinden Crossovers. Durch die blockweise Übertragung der Gene erreicht Operator C eine Koordination der direkt benachbarten Protospinsysteme und kommt damit in den Bereich der G Operatoren.

Gs3p2A ist der Operator, der die qualifiziertesten Gene auf die Nachkommen überträgt, da sowohl die Initialisierung als auch die Differenzbildung aus den Elternteilen erfolgt und außerdem nur verknüpfte Berei−che übertragen werden. Daher wurde für Gs3p2A die kürzeste Laufzeit erwartet. Entgegen der Erwartung aus Kapitel 2.10.4 zeigen dagegen die Operatoren Gs2p2A und Gs3p2N die kürzeste Laufzeit.

Da Gs2p2A auch im schlechtesten Fall noch gute Laufzeiten erzeugt, wurde er für die restlichen Tests verwendet.

7 siehe 5.
8 (genetisch, elitär) erreichte die Schwelle 260 in vier von 6 Versuchen nicht. Der Optimierer blieb bei einer Qualität von 218 bis 248 stecken.

3.6.3 Übertragungsrate (`crossOverPct`)

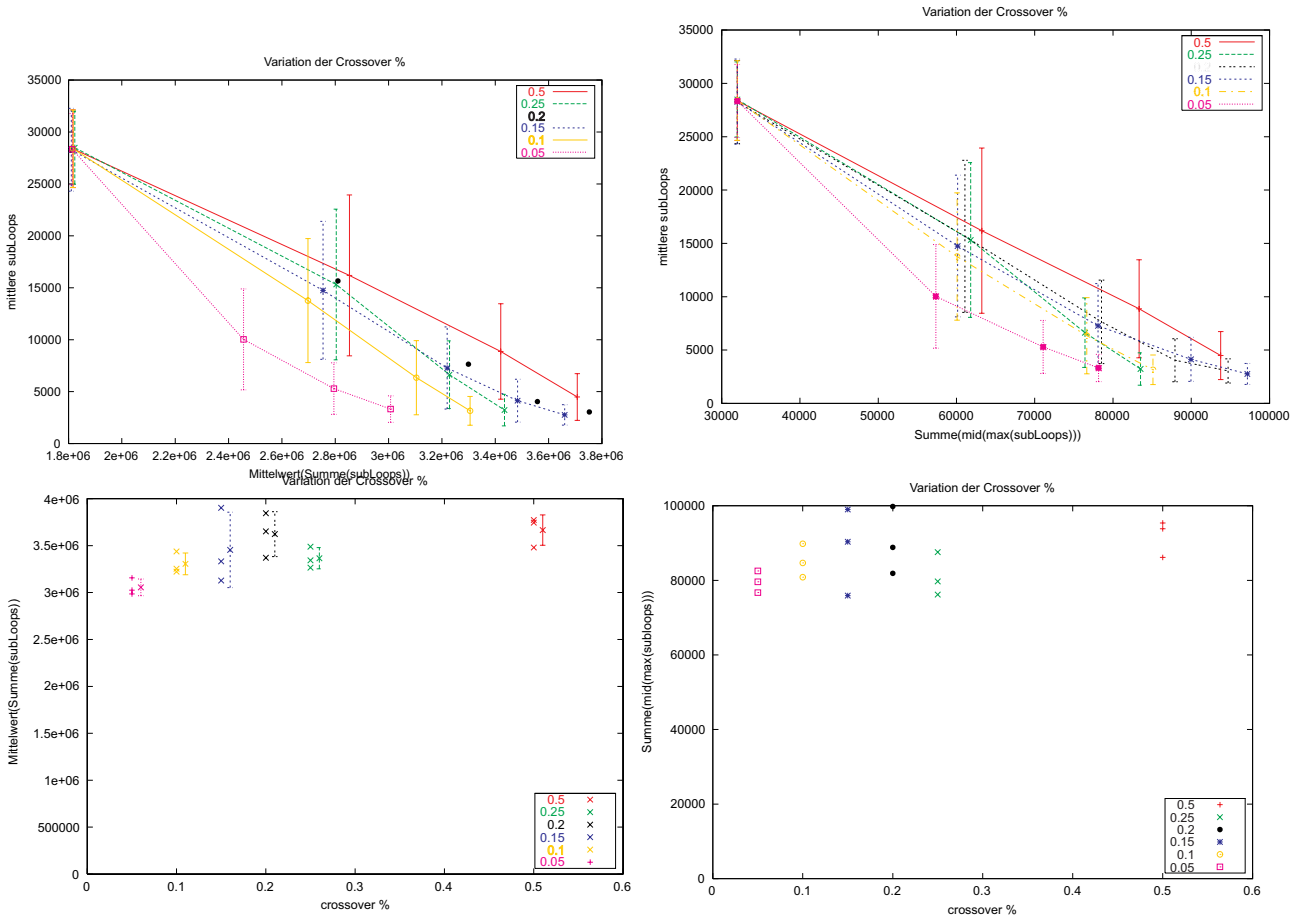


Abbildung 3−12 Parallele und sequentielle Zyklen und crossOverPct.

Die Wahl der Übertragungsrate im Crossover kann beim Gs2p2A Operator die Geschwindigkeit um bis zu 25% beeinflussen. Der Standardwert für `crossOverPct` ist 0.15. Kleinere `crossOverPct` Werte reduzieren die parallele und die sequentielle Zyklenzahl gleichermaßen, während große Werte die Zyklenzahl erhöhen.

Für das lokale Minimum im Bereich von 0.2 konnte keine Begründung gefunden werden.

3.6.4 Konfiguration der Arena `genArenaSize, seedPoolSize, survivingSeeds`

3.6.4.1 Anzahl der Individuen (`genArenaSize`)

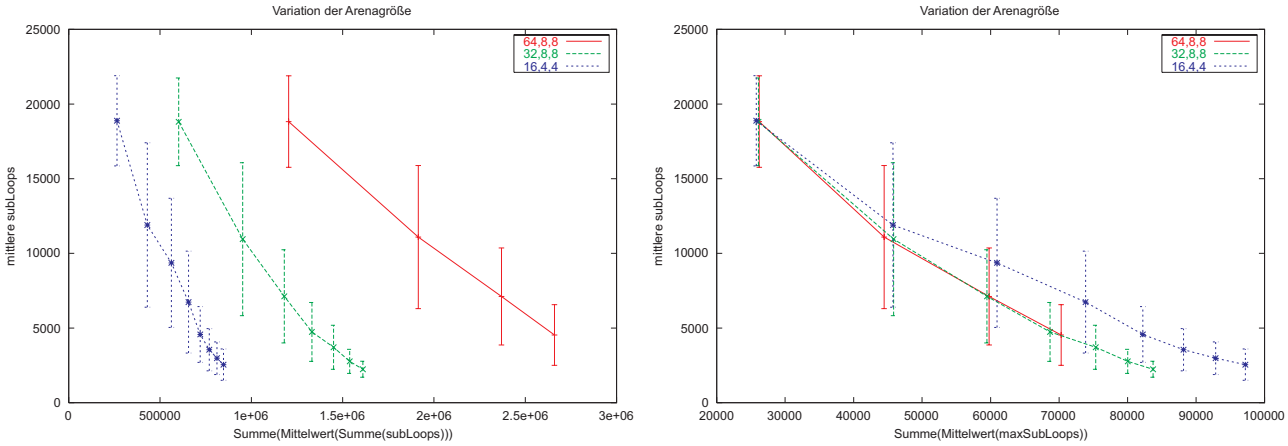


Abbildung 3−13 Parallele und sequentielle Zyklen und die Anzahl der Individuen

Parallele und sequentielle Zyklen zeigen eine entgegengesetzte Abhängigkeit von der Anzahl der Individuen. Während die sequentiell verbrauchten Zyklen linear mit der Anzahl der Individuen ansteigen, sinkt die Zahl der parallel verbrauchten Zyklen. Gleichzeitig sinkt die Anzahl der genetischen Zyklen mit der Anzahl der Individuen. Der Standardwert für die Arenagröße ist 64.

3.6.4.2 Anzahl der Eltern (`seedPoolSize`)

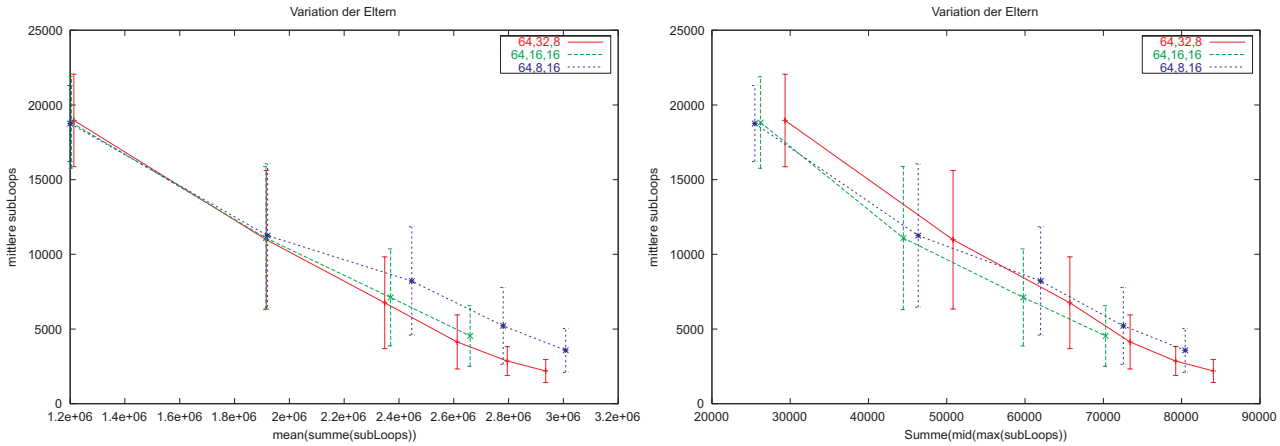


Abbildung 3−14 Parallele und sequentielle Zyklen und die Anzahl der Eltern

Beide Zyklenzahlen hängen nichtlinear von der Elternzahl ab. Sie sinken zuerst mit steigender Elternzahl, um dann wieder anzusteigen. Das Minimum liegt in beiden Fällen bei 16 Eltern. Der Standardwert ist 16.

Die Reihenfolge der absoluten Maxima ist zwischen den beiden Zyklenzahlen umgekehrt.

3.6.4.3 *Anzahl der garantiert überlebenden Individuen (`survivingSeeds`)*

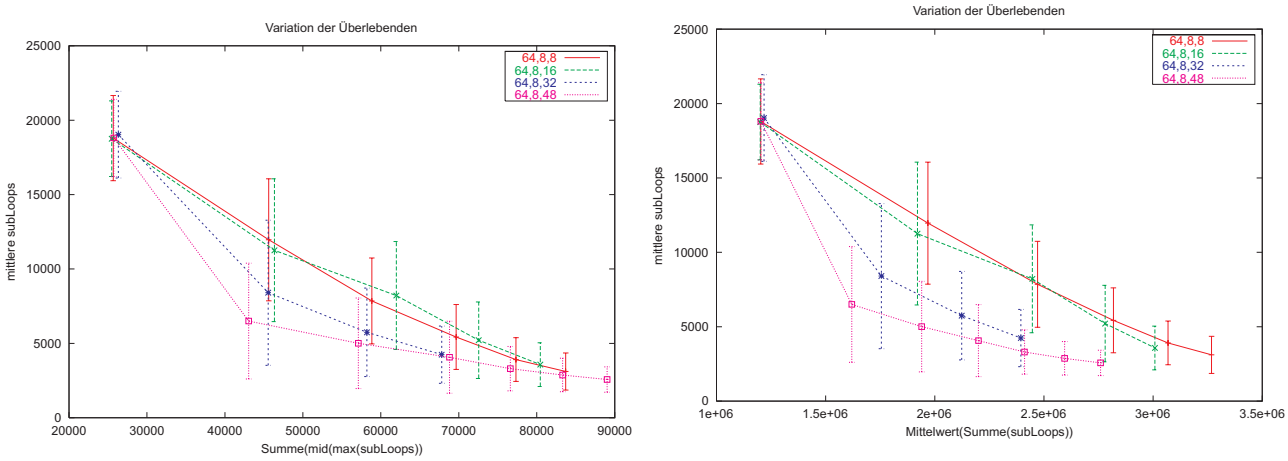


Abbildung 3−15 Parallele und sequentielle Zyklen und die Anzahl der garantiert überlebenden Individuen

Bis 32 `survivingSeeds` sinkt die parallele und sequentielle Zyklenzahl mit steigender Anzahl der überlebenden Individuen. Danach steigt sie unterschiedlich stark an. Für die sequentiellen Zyklen liegt das Maximum bei niedrigen `survivingSeeds` Werten, während das Maximum für die parallelen Zyklen bei hohen Werten liegt.

Der Standardwert für `survivingSeeds` ist 16.

3.6.5 Mindestqualität (`genQThreshold`) und die Zerfallsrate der Mindestqualität (`genQThresholdDecayRate`)

`genQThreshold` und `genQThresholdDecayRate` wurden analog zu `qThreshold` und `qThresholdDecayRate` konfiguriert. `genQThreshold` wurde so eingestellt, daß bis kurz vor Ende der Optimierung wenigstens ein Teil der Arena ersetzt wird.

3.6.6 Minimalwert für die gleitende Crossover−Rate. (`genCrossoverThreshold`)

`genCrossoverThreshold` wurde so eingestellt, daß mindestens 1 Crossover alle 4 Zyklen erfolgt. Die Programmläufe der Statistik halten die gleitende Crossover−Rate von selbst über dieser Schwelle. `GenCrossoverThreshold` wird daher nicht wirksam.

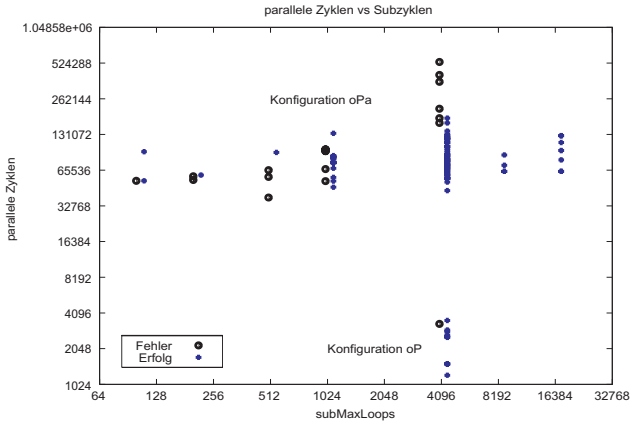
3.6.7 Die Parameter des Abbruchkriteriums. `genMaxEqualLoops` und `genMaxIdleLoops`

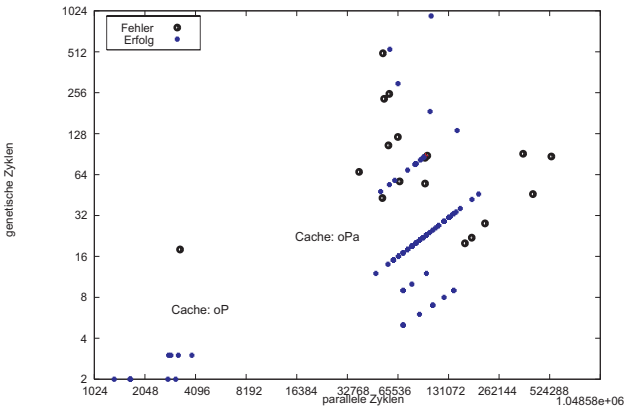
Die Parameter des Abbruchkriteriums sind für die Konvergenz des Algorithmus unkritisch. Sie wurden so eingestellt, daß sie erst dann zutreffen, wenn die Optimierungsphase schon abgeschlossen ist. Der Standardwert für `genMaxEqualLoops` ist 5, für `genMaxIdleLoops` ist er 15.

3.6.7.1 *Verschiedene Cachekonfigurationen*

Die Cachekonfiguration hat großen Einfluß auf die Laufzeit. Da keine geeignete Metrik bekannt ist, die die Schwierigkeit der Optimierung an jedem Knoten und im gesamten Netz bewertet, kann der Qualitätsun−

terschied nur durch die Laufzeit des Optimierers abgeschätzt werden. Die Zykluszahl entspricht der Größe des Zustandsraumes.





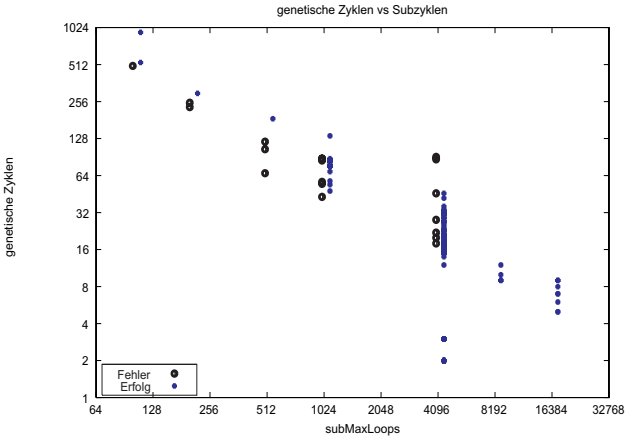


Abbildung 3−16 (link oben) Konvergenz in Abhängigkeit von der Anzahl der parallelen Zyklen und der maximalen Zyklenanzahl im Suboptimierer (alle in zyklenbegrenzter Konfiguration).

Abbildung 3−17 (rechts oben) Konvergenz in Abhängigkeit von der Anzahl der genetischen Zyklen und der maximalen Zyklenzahl im Suboptimierer

Abbildung 3−18 (links unten) Konvergenz in Abhängigkeit von der Anzahl der genetischen Zyklen und der Anzahl der parallelen Zyklen.

Alle Probleme wurden zyklenbegrenzt berechnet.

Die Abbildungen enthalten die Konfiguration oP und oPa (siehe Tabelle 2−2). Konfiguration oP enthält perfekte Protospinsysteme mit AS−Typeninformation, oPa sind Protospinsysteme ohne AS−Typeninformation (P=perfekte ACX, a=alle Positionen). Die Typeninformation vereinfacht die Berechnung mit CheY50 so weit, daß das Problem zu einer *hill−climber* Umgebung für den Genetischen Algorithmus wird. Die Zyklenzahl sinkt dadurch um einen Faktor von 16 bis 132, d.h. um bis zu zwei Größenordnungen.

3.6.8 Der SubOptimierer

Die Literaturkonfiguration des genetischen Optimierers mit kurzen Mutationszyklen, vielen Individuen und vielen Crossover Ereignissen führt beim N−ASS zu einer Stagnation der Entwicklung. Die Crossover Ereignisse sollten die Hauptverbesserungen erzeugen, statt dessen befinden sich die Individuen zuerst im Crossover−Schock. Um GENETIC in der klassischen Literaturkonfiguration zu betreiben, müßte die Arenagröße sehr groß (>16000) und die Mutationsrate klein sein (10 − 50 RANDOM Zyklen pro Individuum), damit jedes Protospinsystem im Cache schon zu Anfang vorhanden ist. Dies läßt die Laufzeit und der vorhandene Speicherplatz nicht zu. Für einen genetischen Zyklus benötigt GENETIC im Optimum mehr als 4h 38 min bei 16000 Individuen, sofern die Datenmengen keine veränderte Speicherlokalität gegenüber den Rechnungen mit weniger als 2000 Individuen erzwingen. Andernfalls wird die Rechenzeit noch höher. Die klassische Literaturkonfiguration ist daher nicht akzeptabel.

Daher muß eine neue Konfiguration für die Kontrolle des Verhältnisses zwischen Mutation und Crossover gefunden werden.

3.6.8.1 Konfigurationsvarianten des SubOptimierers

Da RANDOM als Unterroutine für die genetische Optimierung genutzt wird, kann er völlig anders konfiguriert werden, weil er nicht mehr innerhalb eines Mutationszyklus konvergieren muß. Daher können die Parameter, die zum Abbruchkriterium beitragen (`subMaxTime`, `subMaxLoops`, `subMaxIdleLoops`), neu gewählt werden. Durch drei Varianten der Parametrisierung des Abbruchkriteriums kann die Optimierung unterschiedlich eingestellt werden.

1. Zeitbegrenzt.

   Dabei begrenzt die Laufzeit (`subMaxTime`) die Optimierung während eines Laufs des Suboptimierers. Die Zyklenzahl wird so eingestellt, daß sie nicht zum begrenzenden Faktor werden kann.

2. Zyklenbegrenzt.

   Dabei begrenzt die Zyklenanzahl (`subMaxLoops`) die Optimierung während eines Laufs des Suboptimierers. Die Laufzeit wird so eingestellt, daß sie nicht zum begrenzenden Faktor werden kann.

Bei Zeit− und Zyklenbegrenzung laufen die Eltern den Crossover−Produkten (Kindern) davon, da sie den Crossover−Schock nicht mehr ausgleichen müssen. Sie können die Rechenzeit vollständig für die Verbesserung ihres Zustandes verwenden. Die Kinder müssen erst die Schäden ausgleichen, die durch die Kombination inkompatibler Teile der Elternzustände hervorgerufen wurden. Um produktiv zu sein, muß ein genetischer Prozeß möglichst viele neue reproduktive Individuen hervorbringen. Da neue Individuen nur dann reproduktiv werden können, wenn sie in ihrem ersten Mutationszyklus wenigstens den passiven Teil der Arena erreichen, wird die Produktivität des genetischen Prozesses reduziert, wenn die neuen Individuen nach dem ersten Zyklus weder eine Qualität oberhalb der Qualitätsschwelle haben noch besser als das schlechteste alte Individuum sind.

3. Veränderungsratenbegrenzt.

   Der Suboptimierer kann dabei so lange weiterlaufen, wie das optimierte Individuum eine Veränderungsrate (ΔQ positive Veränderungen pro Zyklen) überschreitet. `subMaxTime` und `subMaxLoops` und `subMaxIdleLoops` werden so eingestellt, daß der Suboptimierer auf die neuen Individuen (Crossover Produkte) zeitbegrenzt wirkt, während er auf die alten Individuen zyklenbegrenzt wirkt, wenn diese schon unter die Veränderungsrate gesunken sind. Die Zeitbegrenzung erlaubt dabei wesentlich mehr Zyklen als die Zyklenbegrenzung[9]. Die maximale Zyklenzahl ergibt sich für die alten Individuen aus `subMaxIdleLoops` und sub`ThresholdDecayRate`.

   Dadurch erhalten die neu erzeugten und die leicht optimierbaren Individuen die Möglichkeit, in ihrer Qualität zu den alten Individuen aufzuschließen. Die Zusammensetzung der Arena wird daher homogener, da die Rechenzeit hauptsächlich auf die Individuen mit der höchsten Veränderungsrate verteilt wird. Sobald die Veränderungsrate des jeweils veränderlichsten Individuums soweit sinkt, daß die theoretische Zyklenzahl zwischen zwei Verbesserungen größer `subMaxIdleLoops` wird, laufen alle

9 In den Programmläufen für die Statistik erhielten die neuen Individuen zwischen 4 bis 16 mal mehr Zyklen.

Individuen zyklenbegrenzt weiter. Außerdem verkürzt diese Konfiguration die Laufzeit zusätzlich, da ein Individuum, das in einem tiefen lokalen Optimum feststeckt, nur die Mindestlaufzeit statt der Maximallaufzeit verbraucht.

Tabelle 3−7 zeigt die zu erwartende maximale Zyklenzahl für unveränderliche Individuen in Abhängigkeit von `subMaxIdleLoops` und `subThresholdDecayRate`.

	subThresholdDecayRate					
subMaxIdleLoops	**0.5**	**0.05**	**.005**	**.0005**	**.00005**	**.000005**
0	10	135	1379	13813	138152	1.38E+006
1000	2001	2001	2379	14813	139152	1.38E+006

Tabelle 3−7 Maximale Zyklenzahl für die alten Individuen, die sich während der Optimierung nicht verändern, bei `qThreshold`*=0.9 in der ratenbegrenzten Konfiguration.*

3.6.8.1.1 `subMaxIdleLoops`

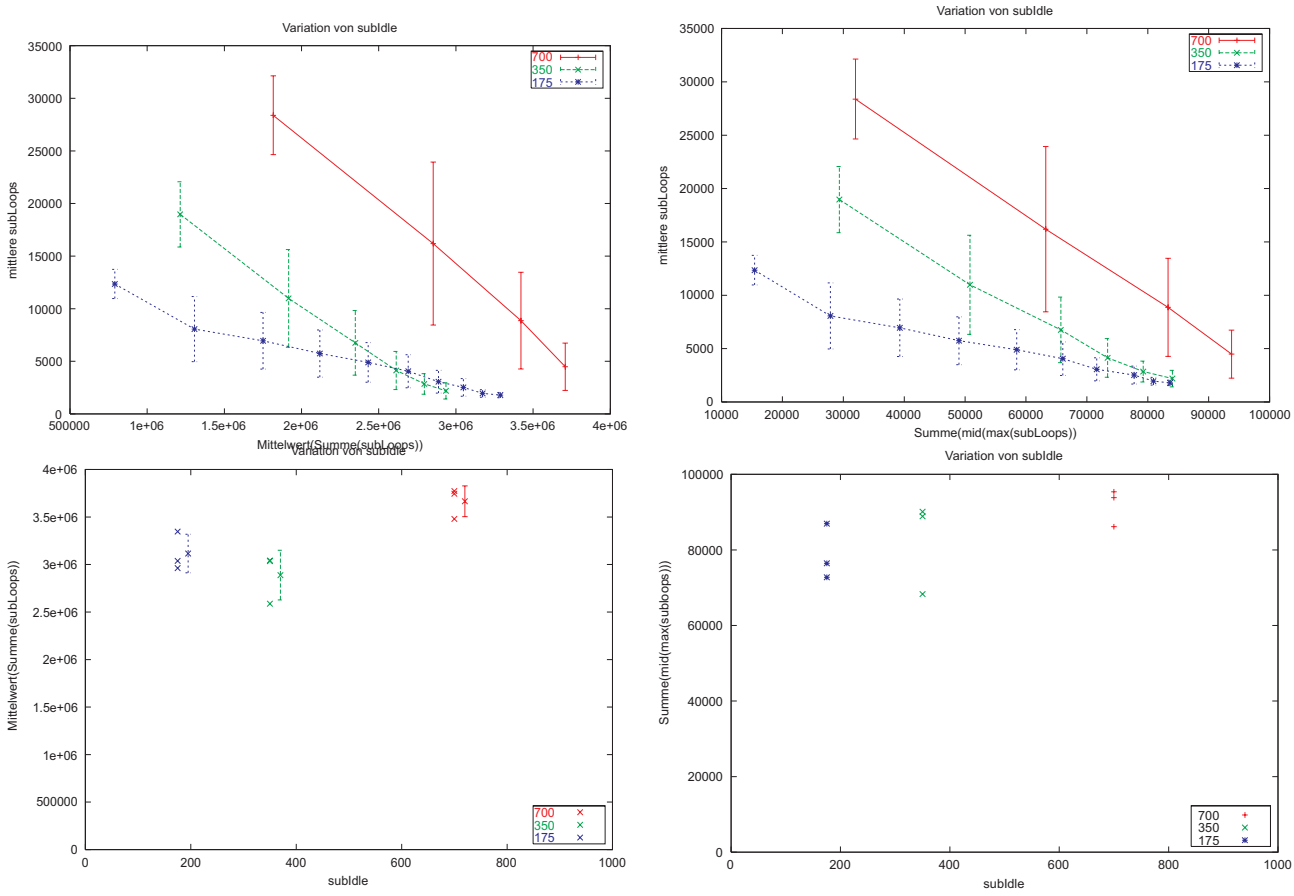


Abbildung 3−19 Variation der `subMaxIdleLoops` *in ratenbegrenzter Konfiguration.*

Der optimale Wert für subMaxIdleLoops liegt bei 350. Der Standardwert ist 750, da er eine intensivere lokale Suche ermöglicht.

3.6.8.1.2 Zerfallsrate `subThresholdDecayRate`

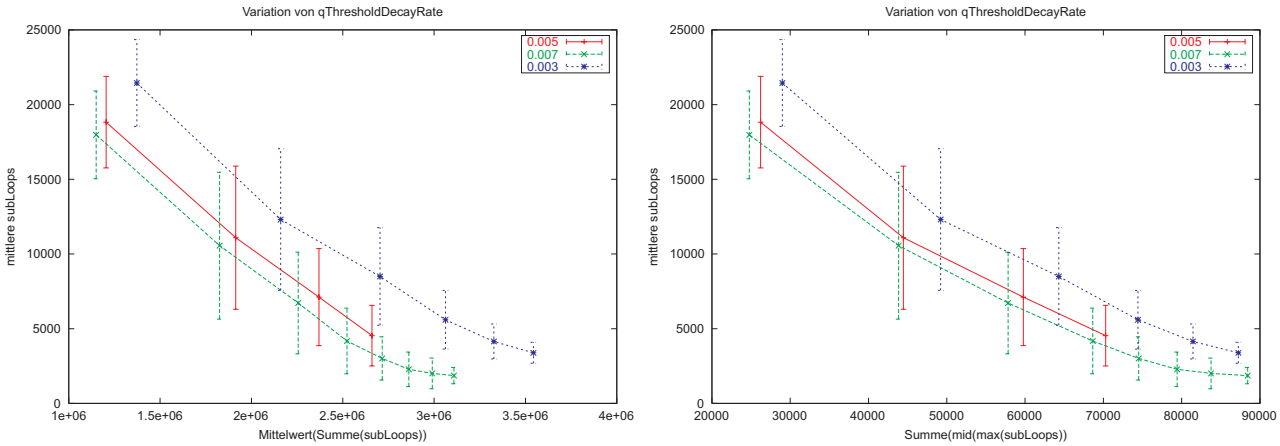


Abbildung 3−20 Variation der `subThresholdDecayRate` *in ratenbegrenzter Konfiguration.*

Der optimale Wert für subThresholdDecayRate liegt bei 0.005. Der Standardwert ist 0.005.

3.6.8.1.3 Vergleich der ratenbegrenzten Konfiguration mit anderen Konfiguratio−nen

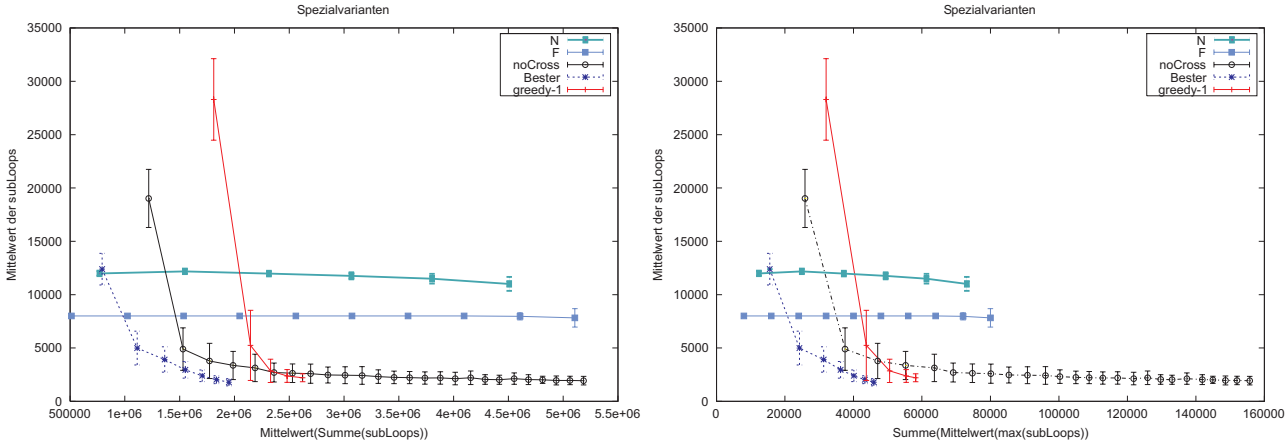


Abbildung 3−21 Vergleich verschiedener Konfigurationen.

***F**: ist zyklenbegrenzt mit* `subMaxLoops=8k`*,*

***N**: ist die ursprüngliche Standardkonfiguration,*

***Bester**: ist die schnellste Konfiguration, ratenbegrenzt.*

***noCross**: zyklenbegrenzte Konfiguration ohne genetischen Austausch. Kein Individuum wird ersetzt. Auch* statistisch *genannt.*

***Greedy−1**: ersetzt alle Individuen durch das beste, ratenbegrenzt.*

Abb.3−21 zeigt den Verlauf der Optimierung für verschiedene Konfigurationstypen. Die ratenbegrenzte Kon−figuration *best* ist allen anderen sowohl in den parallelen als auch in den sequentiellen Zyklen überlegen. Zusätzlich zeigt sie das stabilste Konvergenzverhalten.

Der Vergleich mit *noCross* verdeutlicht den Vorteil jeglichen genetischen Austausches gegenüber dem mehrfachen Ausführen eines reinen RANDOM Optimierers. Selbst wenn man eine gleiche Anzahl RANDOM Optimierer parallel ausführt und alle Optimierer stoppt, sobald der erste das Maximum erreicht, ist ein gene−tischer Optimierer um einen Faktor 3,3 in den parallelen Zyklen und um einen Faktor 2,7 gegenüber den

sequentiellen Zyklen besser. Die Konvergenz der *noCross* Konfiguration ist niedriger als in allen anderen Konfigurationen, außer den greedy Varianten.

Greedy implementiert eine Variante von noCross, allerdings werden, im Gegensatz zu *noCross*, alle n−1 Individuen durch das jeweils beste Individuum ersetzt. In der *greedy−1* Variante werden die SubOptimierer ratenbegrenzt betrieben. In zyklenbegrenzter Konfiguration (*greedy−2* nicht gezeigt) verläuft die Optimierung fast wie Konfiguration F. *Greedy−2* konvergiert aber zwei genetische Zyklen früher und ist damit wesentlich schlechter als *greedy−1*. Selbst die beste *greedy* Konfiguration ist noch um einen Faktor 1,3 schlechter als die beste genetische Konfiguration. Die Konvergenzwahrscheinlichkeit der *greedy* Varianten ist niedriger als in allen anderen Konfigurationen außer `noCross`.

3.6.8.2 Defekte in den Ausgangsdaten

Um die Robustheit des Algorithmus gegen fehlende Peaks zu testen, wurden verschiedene Konfigurationen erzeugt, in denen nur ein Teil der theoretischen Peaks in den Peaklisten vorhanden ist. Die Konfigurationen wurden mit 95 bis 55 % der theoretischen Peaks erzeugt. In der letzten Konfiguration (55%) führt von jedem Spinsystem zu seinem Nachbarn höchstens eine Brücke. Die Konfigurationen wurden aus der theoretischen Konfiguration erzeugt, indem so lange zufällig ausgewählte Peaks gelöscht wurden, bis der erforderliche Prozentsatz gerade unterschritten wurde. Nur die HNCO Peaks blieben immer erhalten.

Jeder Prozentsatz wurde mit mehreren Zufallslöschungen getestet. In allen Fällen wurde das Maximum der Optimierungsfunktion gefunden. In keinem Fall blieb die Optimierung wie bei RANDOM in einem Suboptimum stecken. Bis inklusive 65% war dies auch gleichzeitig die theoretische Zuweisung oder eines ihrer Permutationsäquivalente, die anhand der Frequenzen nicht unterschieden werden können und sich nur durch die Peakidentifikationsnummern unterscheiden. Bei 55% hängt die gefundene Lösung davon ab, ob die Aminosäuretypen noch erkannt werden können und ob die Nachbarschaftsbeziehung noch intakt ist. Andernfalls kann eine andere als die theoretische Zuweisung das Maximum der Optimierungsfunktion bilden. Die theoretische Lösung wird dann nicht mehr gefunden.

3.7 Crossover−Schock

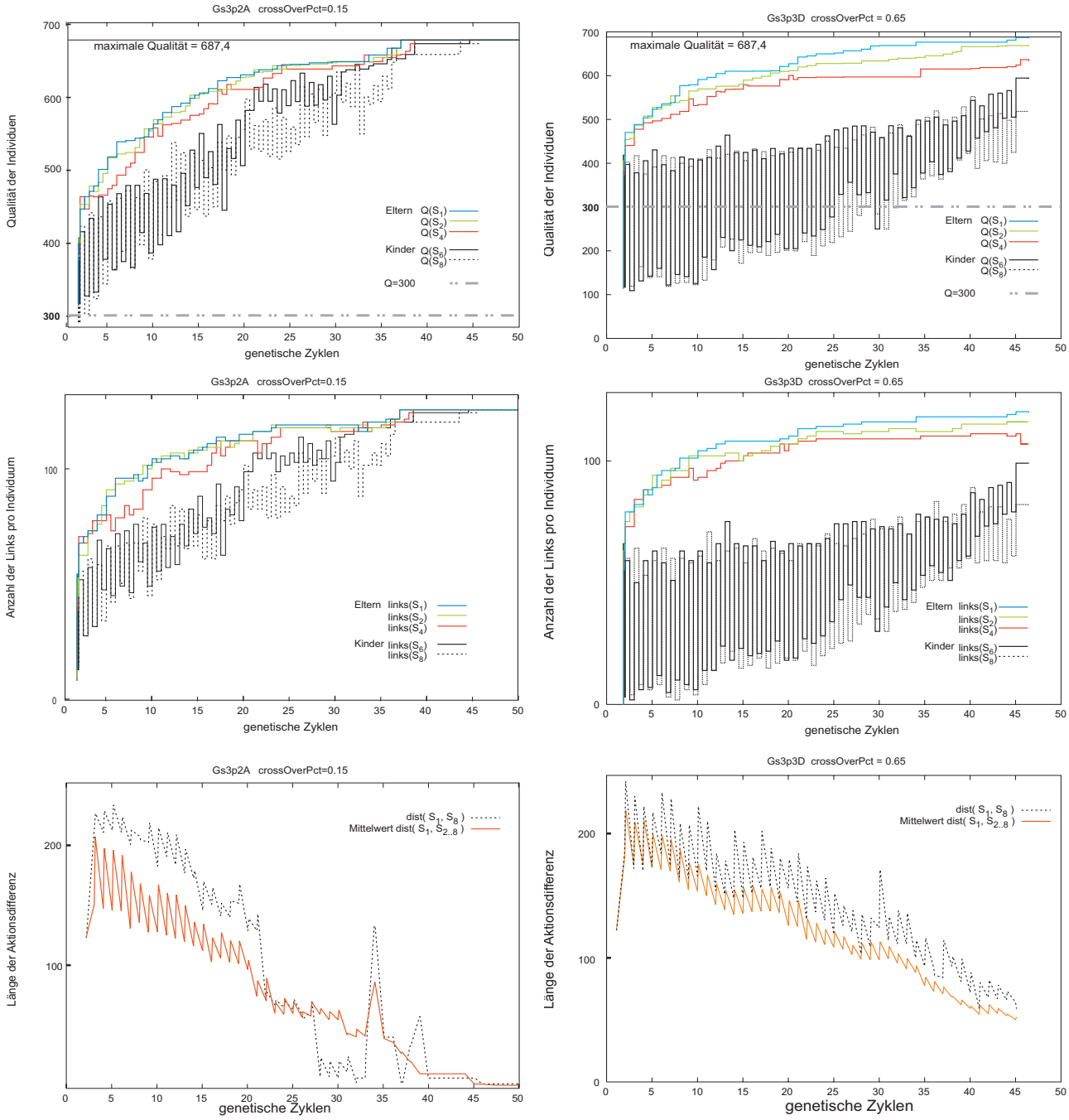


Abbildung 3−22Auswirkung des Crossover Schocks während zweier GENETIC Optimierungsläufe. Unterschiedlich starke Formen des Crossover−Schocks im zeitlichen Qualitätsverlauf anhand des Gs3p2A und Gs3p3D Operators bei unterschiedlichen crossOverPct Werten. Links: der Operator bei `crossOverPct=0.15`, *Rechts: bei* `crossOverPct=0.65`. *Arena=8,4,4*

Die Graphen 3−22 zeigen die Qualitätsentwicklung, die Entwicklung der Zustandsdifferenz zum aktuell besten Zustand und die Anzahl der Verknüpfungen (Links) zwischen benachbarten Protospinsystemen während eines Optimierungslaufes mit GENETIC. Für jeden genetischen Zyklus wird die Qualität zu zwei Zeitpunkten, nach dem Crossover (Phase 2 in Abb. 3−23) und nach der Mutationsphase (Phase 3), darge−

stellt. Vor dem Crossover wurde die Arena jeweils nach der Qualität sortiert, so daß wieder das Individuum mit der höchsten Qualität an Position 1 steht. Während der Mutation wird die Position der Individuen unabhängig von ihrer Qualität beibehalten, so daß man seine Entwicklung während einer Mutationsphase aus der Kurve ablesen kann. Dadurch entstehen die charakteristischen Rechteckstufen in den Qualitäts− , Verknüfungs− und Differenzgraphen.

Für beide Konfigurationen wurde jeweils die Qualitätsentwicklung (oben) und die Anzahl der Links (Mitte) für eine Auswahl der Individuen untereinander dargestellt. Zusätzlich wird die Entwicklung durch die gemittelte Länge der Aktionsdifferenz und die Länge der Zustandsdifferenz dist($S_1 - S_8$) charakterisiert (Gl. 2−16).

Die obere Darstellung enthält die Qualitätskurven Q(S_1) und Q(S_2) für das beste und zweitbeste Individuum, außerdem Q(S_4) für das Elternteil, das zu Anfang des genetischen Zyklus die niedrigste Qualität hatte, das zweite neue Individuum Q(S_2) und das letzte neue Individuum Q(S_8). Die Qualitätskurve Q(S_4) ist die Qualitätsentwicklung des schlechtesten Elternteils. Sie markiert die Grenze zwischen der Qualitätsentwicklung der Eltern und der der Kinder. Die breiten Rechtecklinien in der oberen Mitte der Graphen, unterhalb der Kurve für das schlechteste Elternteil Q(S_4), sind die Qualitätsentwicklung des schlechtesten und mittleren Kindes vor und nach dem Crossover. Die Höhe der Rechteckkurven und der Abstand zu Q(S_4) spiegelt die Intensität des Crossover−Schocks wieder.

Die Qualitätsentwicklung der gesamten Arena wird sowohl durch die Distanz Q(S_1)−Q(S_4) zwischen dem besten und schlechtesten Elternteil und durch den Abstand Q(S_4)−Q(S_8) zwischen dem schlechtesten Elternteil und dem mittleren neuen Individuum charakterisiert. Abb. 3−22 zeigt links einen leichten Crossover−Schock und rechts eine extreme Form des Schocks. Alle Graphen auf der rechten Seite zeigen deutlich, daß zwischen dem reproduktiven Teil der Arena und den Nachkommen kein Austausch erfolgen kann, da die Qualität des besten Nachkommens durchschnittlich mehr als 100 − 150 Qualitätseinheiten vom schlechtesten Elternteil entfernt bleibt und dies daher nicht überholen kann. Die Nachkommen bleiben daher unproduktiv. Auf der linken Seite beträgt der Abstand dagegen nur 10 − 75 Einheiten und überschneidet sich nach den ersten 23 Zyklen sogar. Daher kann die Lücke von sehr guten Nachkommen übersprungen werden.

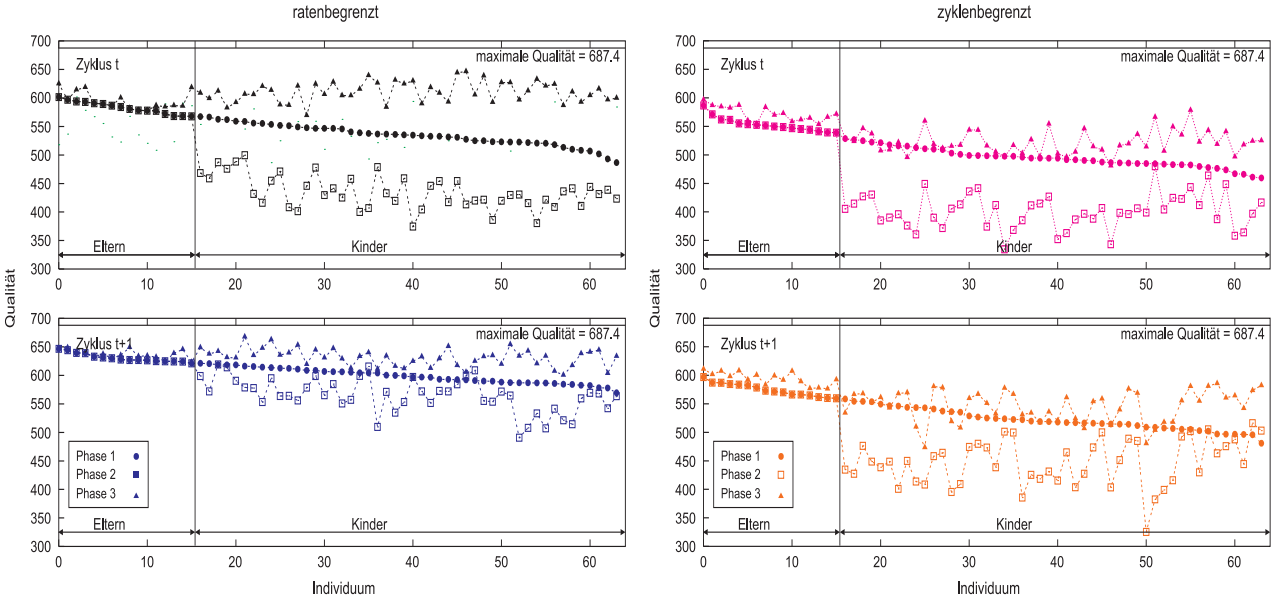


Abbildung 3−23 Wirkung des Crossover−Schocks während zweier aufeinander folgender Crossover−Zyklen in zwei verschiedenen Optimierungen. Dargestellt ist das Qualitätsprofil der Arena anhand einer zyklen− und einer ratenbegrenzten Konfiguration. Die aufeinander folgenden Zyklen sind untereinander dargestellt. Links: ratenbegrenzt, der Crossover−Schock wird durch die Ratenbegrenzung ausgeglichen. Rechts: zyklenbegrenzt, durch den Schock entwickeln sich die Nachkommen schlechter als die Eltern, die Stärke des Schocks entspricht Abb. 3−22 links. Arena=64,16,16

Der Vergleich der Graphen in Abb. 3−23 zeigt deutlich die Auswirkung des Crossover−Schocks bei zwei verschiedenen Konfigurationen. Die Zweiteilung der Arena in den Bereich der Eltern auf den Positionen 1 bis 16 und der Nachkommen von 17 bis zum Ende der Arena[10] ist an der unterschiedlichen Qualitätsentwicklung zu erkennen.

Jeder genetische Zyklus wird in 3 Phasen eingeteilt (dargestellt durch unterschiedliche Symbole). Phase 1, zu Beginn des Zyklus, zeigt das Profil, nachdem die Arena sortiert wurde. Danach erzeugt das Crossover neue Individuen. In Phase 2 ist das Profil nach dem Crossover dargestellt. Danach findet die Mutation statt. Phase 3 zeigt die Individuen nach der Mutation, aber noch bevor die Arena erneut sortiert wird. Damit ist der genetische Zyklus beendet.

Im linken Graph überholen die neuen Individuen die Eltern innerhalb eines genetischen Zyklus, im rechten bleiben sie unterhalb der Durchschnittsqualität der Eltern, obwohl diese sich selbst kaum verbessern. Daher kommt es auf der rechten Seite nur zu einem geringen Austausch zwischen den Eltern und den Kindern, während links die Eltern in jedem Zyklus vollständig durch die Nachkommen ersetzt werden.

10 Da die Arena vor dem Crossover neu sortiert wird, sind die Eltern immer am Anfang der Arena.

3.8 Schlußfolgerungen

Während der Suche nach der günstigsten Konfiguration für die Optimierung wurden einige Effekte beobachtet, die aus der Literatur nicht bekannt waren und der ursprünglichen Erwartung entgegenlaufen.

Die Optimierung des RANDOM Optimierers verlief in der erwarteten Weise. Sowohl die Zusammensetzung der Profile als auch die Zahlenwerte der TTL−Variablen entsprachen der theoretischen Erwartung.

Dagegen birgt die Optimierung des GENETISCHEN Optimierers einige Überraschungen. Entgegen der Erwartung (Seite 74) ist Gs2p2A, und nicht Gs3p2A, der bessere Operator, obwohl Gs3p2A Bereiche mit höherem Organisationsgrad kopiert. Ursache ist wahrscheinlich der unterschiedliche Ordnungsgrad der übertragenen Bereiche. Selektion s3 kann nur verknüpfte benachbarte Bereiche übertragen. Dies scheint ein Nachteil zu sein, da bei s3 keine Übertragung möglich ist, wenn in der unverknüpften Nachbarschaft schon ein Teil der globalen Lösung enthalten ist. Durch die niedrigeren Anforderungen an den Organisationsgrad der Nachbarschaft kann s2 häufiger längere Stücke kopieren als s3, da, gerade in der Anfangsphase der Optimierung, der Verknüpfungsgrad noch niedrig (~65%) ist und Bereiche, in denen s3 die vollen 4 Elemente kopieren kann, seltener sind als Bereiche mit 4 unverknüpften Nachbarn. Selektion s3 degeneriert dadurch fast zu s1, da es fast immer nur das erste und zweite Element in einer Nachbarschaft kopieren kann.

Die zweite Überraschung ist die Existenz und Bedeutung des Crossover−Schocks, der in der Literatur nicht beschrieben wird. Auf den ersten Blick ist der Schock eine Nachteil für die neu entstehenden Individuen, da er ihre Qualität soweit verringert, daß sie nicht mehr mit den Eltern direkt konkurrieren können. Die Zerstörungen, die den Schock erzeugen, führen aber auch zu einer Neuorganisation der betroffenen Bereiche. Da diese Bereiche nach dem Schock sehr veränderlich sind, halten sie die effektive Veränderungsrate hoch. Die Zerstörung der, durch Resourcenkonkurrenz, verknüpften Bereiche verschafft also den neuen Individuen über den Mutationsalgorithmus auf doppelte Weise einen Vorteil gegenüber den Eltern.

Durch die Konkurrenz werden gezielt mit dem im Crossover ersetzten Bereich korrelierende Bereiche freigegeben. Freie Bereiche werden aber schon nach wenigen RANDOM Zyklen neu belegt. Dabei kann das Protospinsystem, das vorher diese Position belegt hatte, nur mit einer geringen Wahrscheinlichkeit wieder auf diese Position gelangen und die Position bleibt frei bis ein neues Protospinsystem für diese Position gefunden wurde.

Gleichzeitig liegt die Veränderungsrate anfangs höher als die der Eltern, da die Kinder durch die Zerstörung näher an den Ausgangspunkt der Optimierung $\boldsymbol{S_0}$ rücken. Die Kinder befinden sich in Bezug auf ihren Belegungsgrad in einem Entwicklungsstadium, das die Eltern schon verlassen haben. Allerdings haben die schon vorhandenen Zuweisungen in den Kindern zusammen eine höhere Qualität als sie die Eltern in diesem Stadium, beim Erreichen des gleichen Belegungsgrades, hatten. Durch den niedrigeren Belegungsgrad befinden die Kinder sich aber auch im Einzugsbereich von Optima, die für die Eltern nicht mehr erreichbar sind. Sie können daher Optima des Lösungsraumes erreichen, die von den Eltern nicht erschlossen werden können. Diese Crossover−Operation beschränkt daher den erreichbaren Raum nicht auf den Lösungsraum entlang des direkten Pfades zwischen den Eltern, sondern erweitert ihn! Auch dies ein Unterschied zu den in der Literatur besprochenen Operatoren.

Das Ergebnis der Parameterabschätzung ist daher alles andere als trivial oder vorhersehbar.

Kapitel 4 Vergleich mit PASTA

4.1 Konfiguration nach Leutner et al.

Das folgende Kapitel bildet die Konfiguration, wie sie von Leutner *et al.* in [Leutner98] verwendet wurde, in den Datenstrukturen des RANDOM Algorithmus nach. Konfiguriert wird ein perfekter Datensatz mit einer theoretischen Peakliste für CheY, da nur in dieser Konfiguration vergleichbare Bedingungen gegeben sind.

- NMR Experimente:

  Für PASTA werden HNCO, ^{15}N −HSQC, HNCACB und CBCA(CO)NH aufgenommen. RANDOM erfordert HNCO, HNCA, CBCA(CO)NH und CBCANH.

- Peakabstände und Netzbildung.

  Leutner *et al.* bilden keine spezielle Datenstrukturen für die Verknüpfung der Peaks aus. Die Verknüpfung wird nur durch die Frequenzdistanzen hergestellt. Leutner *et al.* verwenden:

  Abstände in der H und N Dimension: 0.01 ppm maximale Differenz.

  Abstände in der CA Dimension: 0.1 ppm maximale Differenz.

- Alle theoretischen Peaks sind vorhanden (100% Peaks). Daher können alle Protospinsysteme *perfekt* gebildet werden. Außerdem sind alle Protospinsysteme vorhanden.

- Bei der Erzeugung der Protospinsystem lassen Leutner *et al.* die nicht eindeutigen Fälle vom Benutzer des Programms entscheiden. Zusammen mit den perfekten Spinsystemen folgt daraus, daß keine Subspinsysteme vorhanden sind und daß die Spinsysteme nicht überlappen dürfen. Jeder Peak kommt in nur einem Protospinsystem vor. Die Protospinsystemerkennung akzeptiert für eine perfekte Peakliste daher nur perfekte Spinsysteme, d.h. die theoretischen Spinsysteme. RANDOM kann mit mehrdeutigen Peakgruppen umgehen, indem es alle zulässigen Spinsysteme in den Lösungsraum aufnimmt. Unabsichtliche doppelte Nutzung des gleichen Peaks wird durch den (aquire,release) Mechanismus während der Optimierung verhindert. Im Pool können daher auch Protospinsysteme aufgenommen werden, die sich überlappen, d.h. in Konkurrenz um Peaks stehen.

- Leutner *et al.* benutzen in ihrer schnellsten Konfiguration keine Aminosäurentypenerkennung. Für die Cachekonfiguration bedeutet das, daß alle Protospinsysteme auf allen Aminosäurepositionen zulässig sind. Die schnellste Konfiguration für RANDOM ist die Konfiguration mit Aminosäureerkennung. Beide Programme können aber auch die jeweils andere Konfiguration verwenden.

- Definition der theoretischen Lösung (Endpunkt):

  Leutner *et al.* definierten 100% korrektes Assignment als jede Lösung, die alle benachbarten Protospin−systeme verknüpft und jeder Aminosäure ein Protospinsystem zuweisen. Für die Laufzeitmessungen wurde für die Proline ein künstliches Protospinsystem erzeugt, um nur einen durchgehenden Strang als theoretische Lösung zu erhalten. Da die Peak−Peak Abstände keinen Einfluß auf die Qualität der Zuwei−sung haben, sind unterschiedliche Lösungen, die sich nur in den Abständen der Permutation der Peaks aber nicht in der Konnektivität der Protospinsysteme unterscheiden, gleich! Der resultierende Lösungs−raum ist eine reine *hill climber* Umgebung mit nur wenigen flachen Nebenmaxima und wesentlich glatter als für die vergleichbare Umgebung mit RANDOM.

- In RANDOM wird dagegen für ein 100% korrektes Assignment nicht nur eine vollständige Verknüpfung aller benachbarten Protospinsysteme, sondern auch eine korrekte Plazierung der Peaks, eine Minimie−rung der Peak−Peak Abstände, eine Maximierung der Aminosäureerkennung und eine Maximierung der Anzahl der interpretierten Peaks verlangt. Außerdem verbietet RANDOM eine unabsichtliche doppelte Verwendung eines Peaks und erzeugt sich wechselweise auschließende Protospinsysteme zwischen denen es in der Optimierung auswählen muß. Der Lösungsraum ist daher rauher, mit vielen tiefen Nebenmaxima.

- Hardwareunterschiede:

  In [Leutner97] wurde eine SGI R4000 SC mit 133 MHz eingesetzt. 15000 Zyklen brauchten ungefähr 160 min (~ 93 L/min bzw. 1.5 L/sec). Dagegen liefen diese Untersuchungen auf einem Linux PC PII 400MHz. Der Geschwindigkeitsunterschied aufgrund der Frequenzen sollte daher (400/133 = 3.0075) betragen.

- Compiler und Implementierungs−Sprachen:

  PASTA wurde in C implementiert. Der Kompiler ist der kommerzielle SGI C Compiler. RANDOM wurde in C++ implementiert und mittels GNU egcs v 2.91.60 19981201 erzeugt. Der kommerzielle Compiler sollte eine bessere Optimierung des Objektcodes erzeugen. C++ gilt als die langsamere Sprache. Aller−dings kann man in C++ leichter komplexe Algorithmen nutzen als in C. Dadurch fällt es in C++ Program−men leichter, ausgefeiltere Algorithmen zu nutzen. Die Auswirkung dieser Unterschiede kann nicht vor−hergesagt werden.

- Profile:

  PASTA und RANDOM nutzen unterschiedliche Aktionsprofile. PASTA weist zu Anfang alle Protospinsy−steme je einer Aminosäure zu und vertauscht danach nur noch diese Protospinsysteme. Einen Pool mit ungenutzten Aminosäuren kennt PASTA nicht. RANDOM startet mit einem leeren Assignment und wählt dann sowohl die Protospinsysteme als auch die Positionen aus.

  Daher sind die Aktionsprofile unterschiedlich zusammengesetzt. PASTA braucht keine Aktionen, die Pro−tospinsysteme zwischen Zuweisung und Pool austauschen. Diese Aktionen gibt es nur bei RANDOM. Außerdem unterscheidet sich die *Implementierung* der vergleichbaren Aktionen, da PASTA nie auf Resourcenkonflikte stoßen kann.

4.1.1 Vergleich des Laufzeitverhaltens von PASTA mit RANDOM

Ein direkter Vergleich der Algorithmen ist nicht möglich, da die Implementierungen für unterschiedliche Probleme optimiert sind und vergleichbare Konfigurationen nur mit Einschränkungen erstellt werden können. Außer den Algorithmen unterscheiden sich die Compiler, die Hardware und die Definition des Endpunktes.

Die Anzahl der Zyklen liegt für die Konfigurationen mit theoretischen, perfekten Protospinsystemen bei beide Algorithmen in der gleiche Größenordnung. Der RANDOM Algorithmus ist im günstigsten Fall, mit AS−Typenerkennung und gefiltertem Protospinsystem−Cache, 24 mal schneller als PASTA. Dabei sind die unterschiedlichen CPU Frequenzen schon berücksichtigt. Ohne Typenerkennung ist der RANDOM Algorith−mus langsamer. Ob dies an den Algorithmen oder an der Implementierung liegt, kann nicht entschieden werden. Der RANDOM Algorithmus kann allerdings Probleme lösen, die PASTA nicht erreichen kann.

	PASTA	**PASTA**	**PASTA**	**PASTA**	**RANDOM**	**RANDOM**
Konfiguration				**schnellste**	**oPa**	**oP = schnellste**
	Anz. Zyklen	**Zeit [min]**	**Zyklen**	**Zeit [min]**	**Zeit [sec]**	**Zeit [sec]**
AS Typen	mit	mit	ohne	ohne	ohne	mit
Proteinlänge						
150	1600	~ 24 korrigiert 8	960	14,4 korrigiert 4,8		
125	1300	~ 19,5 korrigiert 6,4	733	11 korrigiert 3,66	17 min f = 0,64	8 sec f = 27
50	533	~ 5 korrigiert 1,66	240	3,6 korrigiert 1,2	3,8 min f = 0,94	4 sec f = 18

Tabelle 4−1 Vergleich der Laufzeiten für PASTA und RANDOM mit CheY. Die korrigierten Zeiten berück−sichtigen die verschiedenen CPU Geschwindigkeiten. Die Zeiten wurden aus der Abbildung in [Leutner et al. , S.34] rekonstruiert.

Kapitel 5 Automatische Zuweisung des Proteinfragments Trigger M

5.1 Trigger Faktor

Das Protein Trigger gehört zu einer Gruppe von Katalysatoren, die die Faltung neu erzeugter Proteine an den Ribosomen steuern. Es wurde zuerst im Cytosol von *e. coli* Bakterien entdeckt [Crooke87] und konnte seitdem auch aus anderen Organismen wie *bacillus subtilis* und *mycoplama genitalium* [Scholz98, Göthel97] gewonnen werden. Es hat ein Gewicht von 58 kDa und besteht aus 432 Aminosäuren [Hesterkamp96], die in drei strukturell verschiedene Regionen N, M und C aufgeteilt sind [Zarut97].

Der N−terminale Bereich N, mit den Aminosäuren 1 bis 145, hat vorwiegend α−helix Struktur und bindet das Protein an die Ribosomen. Bereich M, von 145 bis 251, enthält das katalytisch wirksame Zentrum. Der Bereich hat vorwiegend β−faltblatt Struktur. Die Funktion des Bereichs C, von 251 bis 432, ist noch nicht geklärt [Zarut97]. Es wird aber vermutet, daß C für die Bindung der noch ungefalteten Proteine an den Katalysator zuständig ist. Dadurch würde Trigger dann auch zur Gruppe der Chaperone gehören [Scholz98, Deuerling99].

Nach der Modellvorstellung von Zarut *et al.* [Zarut97] bindet sich Trigger mit dem N−Terminus an die Ribosomen und verbindet sich danach mit dem neugebildeten, noch ungefalteten Protein.

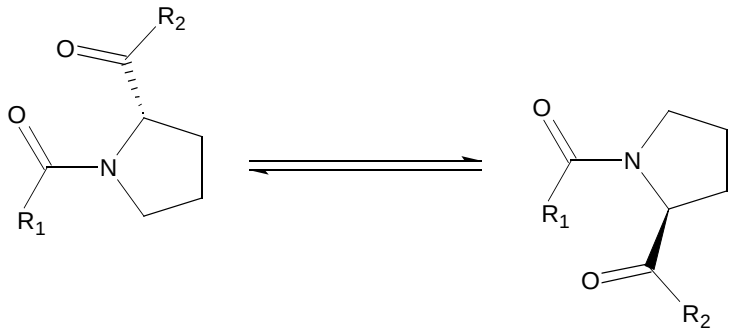


Abbildung 5−1 cis/trans − Isomerisierung der prolyl−Bindung durch Trigger

Aus Experimenten ist bekannt, daß Trigger sich nicht an schon teilweise gefaltete Proteine, sondern nur an ungefaltete binden kann. Dadurch wird die Ablösung des Katalyseproduktes vom Katalysator erzwungen. Andernfalls würde Trigger durch das Überangebot der im Cytosol vorhandenen gefalteten Proteine inhibiert [Scholz98].

Das aktive Zentrum wird im Abschnitt M vermutet. Es katalysiert die Isomerisierung der Aminosäurebindung vor den Prolinen. Die *prolyl*−Bindung tritt in aktiven Proteinen häufig in *cis*−Form auf, obwohl diese Bindung im Ribosom zuerst in *trans*−Form erzeugt wird und bei anderen Aminosäurepaaren normalerweise

die *trans*−Konformation in den aktiven Proteinen bevorzugt wird. In der Faltung der neugebildeten Proteine ist die Isomerisierung der prolyl−Bindung der ratenbestimmende Schritt [Reimer98]. Trigger kommt daher eine entscheidende Rolle in der Ausbildung der Funktion eines Proteins zu.

Aminosäure−bereich	Sequenz in 1 Buchstabennotation
1 − 51	**MRGSH HHHHH G**SEKL AKTKS TMVDV SDKKL ANGDI AIIDF TGIVD NKKLA
51 − 101	SASAQ NYELT IGSNS FIKGF ETGLI AMKVN QKKTL ALTFP SDYHV KELQS
101 − 113	KPVTF EVVLK

*Tabelle 5−1 Aminosäuresequenz des vermessenen Proteinfragments mit HIS−Tag. Aminosäure 12 entspricht der Aminosäure 151, Aminosäure 113 der Aminosäure 252 in der Sequenz des nativen Proteins. Der HIS−Tag ist **fett** gesetzt.*

5.1.1 Die Proteinprobe

Die Proteinprobe wurde von Dr. Andreas Pahl an der Universität Erlangen in e.coli Kultur präpariert. Die Sequenz der Probe (siehe Tab. 5−1) entspricht dem mittleren Abschnitt M des Trigger Proteins von Aminosäure 151 bis 252. Am N−terminalen Ende der Sequenz wurde ein sogenannter His−Tag und eine Linkersequenz angesetzt, um die Proteinsequenz am Harz zu verankern. Dieser Abschnitt verlängert das Protein um 11 auf 113 Aminosäuren. Für die NMR−Experimente wurde der His−Tag nicht abgetrennt. Allerdings wird das Protein langsam vom N−Terminus durch Hydrolyse abgebaut, so daß die effektiv vermessene Sequenz in den verschiedenen Experimenten unterschiedlich lang ist.

Der Abbau bleibt aber innerhalb des HIS−Tags und erreicht die eigentliche Sequenz nicht. Da die zu erwartenden Frequenzen der Aminosäuren des intakten His−Tags sich nur wenig von einander unterscheiden und daher stark überlagern und die unterschiedlich abgebauten His−Tags zu unvollständigen Spinsystemen führen, ist ein Nachweis dieses Abschnitts sehr unwahrscheinlich. Auf eine Zuweisung des Bereichs Aminosäure 1 bis 10 wurde daher verzichtet.

AS	H	N	CA	CB	CO
HIS			52−**54**−60 ppm	27−37	

Tabelle 5−2 Erwartete Frequenzen des His−Tags

Die Datenstruktur des Protein wurde für die Optimierung zwar mit einer Länge von 113 Aminosäuren angelegt, doch die Zuweisung zu den ersten 10 Aminosäurepositionen wird mit einem Qualitätsfaktor (asQFactor) von 0,1 bis 0,8 bestraft. Dadurch ist eine Zuweisung zwar möglich, erfolgt aber erst, wenn überschüssige Spinsysteme existieren, die an anderer Stelle entweder gar nicht oder nur sehr schlecht verwendet werden können.

5.2 Experimenteller Teil

Die Messungen wurden in einer 50 mmol Phosphatpufferlösung bei pH 6,5 durchgeführt.

Für die automatische Zuweisung werden ein 3D−HNCO [Maurer2000, Kay90, Grzesiek92b], ein 3D−HNCA [Maurer2000, Archer91], ein 3D−HN(CO)CACB [Maurer2000, Grzesiek92a] und ein 3D−HNCACB [Maurer2000, Grzesiek92a] benötigt. Die Spektren wurden von Tanja Parac und Markus Maurer auf einem Bruker 800 MHz bei T= 298 K aufgenommen. Die HN(CO)CACB− und HNCACB−Spektren unterscheiden sich von den CBCANH und CBCA(CO)NH durch die umgekehrte Aufnahmereihenfolge (*out−and−back*) [Maurer2000, Muhandiram94, Wittekind93] und das verbesserte Signal / Rausch −Verhältnis.

Diese Experimente werden in den folgenden Kapiteln als CBCANH und CBCA(CO)NH bezeichnet, um die Einheitlichkeit der Benennung mit den vorhergehenden Kapiteln beizubehalten.

Die prozessierten Spektren wurden manuell in Felix gepickt und die resultierenden Peaklisten exportiert. Für HNCA und CBCANH wurden, außer den Koordinaten, das Volumen oder die Höhe der Peaks exportiert, da sie für die Bewertung und die Erzeugung der Spinsysteme notwendig sind.

	HNCO	**HNCA**	**CBCA(CO)NH**	**CBCANH**
Koordinaten	x	x	x	x
Volumen mit Vorzeichen.	−	x	−	x

Tabelle 5−3 Exportierte Eigenschaften der Peaks.

5.2.1 Präparation der Daten

Die Peaks wurden entlang der ^{1}HN und ^{15}N Achse mit einer neuen Referenzierung auf die Peaks des HNCOs zentriert. Dazu werden, anhand einer H/N−Projektion aller Peaks des HNCOs mit den Peaks des zweiten auszurichtenden Spektrums, Gruppen aus je einem HNCO Peak und den dazugehörigen Peaks im zweiten Spektrum gebildet. Die Peaks der Peakgruppen wurden nur in isolierten Spinsystemen ausgewählt, bei denen die Zuordnung zum HNCO eindeutig ist. Die Abstände in der H/N−Ebene werden dann mit dem *least square* Verfahren bewertet. Dazu wird, für das zweite Spektrum, mittels *random walk* eine neue Refe−renzierung erzeugt, bis der gemittelte quadratische Abstand mit der neuen Referenzierung minimal ist.

$$f(Peakpaare, w) = \frac{\sum_{p=1\ldots a_p} \sqrt{\frac{1}{i_{max}} \sum_{i=1\ldots i_{max}} \frac{\Delta x_{i,p}^{\,2}}{w_i}}}{a_p} \qquad (5\text{−}1)$$

mit a_p Anzahl der Peakpaare
w Gewichtung je Achse
i_{max} Anzahl der verglichenen Dimensionen

Auf diese Weise wurden die Experimentpaare HNCO + HNCA, HNCO + CBCANH und HNCO + CBCA(CO)NH neu ausgerichtet.

Experimente	Peakpaare	VORHER	NACHHER
HNCO − HNCA	23	1,090	0,250
HNCO − CBCA_CONH	40	0,502	0,209
HNCO − CBCANH	58	1,218	0,438

Tabelle 5−4 Gemittelte Abstände (Gl. 5−1 vor und nach der Referenzierung.(Normierungsfaktoren: ^{1}H = 0,03 ppm , ^{15}N = 0,5 ppm)

Im nächsten Schritt wurden die Peaklisten und Experimente in den Datenpool importiert und in das For−mat des Assign−Projektes konvertiert. Dabei wurden alle Peaks auf 4 Dimensionen erweitert und die Achsen in die Reihenfolge [H, N,CO,CA/CB] gebracht. Gleichzeitig erhielten die Peaks neue Identifikationsnummern.

	HNCO	HNCA	CBCA(CO)NH	CBCANH
ID Bereich	1−999, virtuelle Peaks ab 200	1001−1999	2001−2999	3001−3999

Tabelle 5−5 Wertebereiche der neuen Identifikationsnummern

5.2.2 Mustersuche

Für die Mustersuche wurden die Peaks in einem PeakNet entlang der H/N Ebene mit den HNCO Peaks verknüpft. Der Abstand zwischen den verknüpften Peaks wird dazu mittels Gl. (B−1) für beide Dimensionen bewertet. Ein Peakpaar wird dann entlang beider Dimensionen verknüpft, wenn die Qualität beider Distan−zen größer 0.5 ist.

	HN	N
Abstand [ppm]	0.05	1 oder 2

Tabelle 5−6 Parameter für die Nachbarschaftsfunktion (Gl. B−1) in der 2D ^{15}N / ^{1}HN Ebene

Zur Qualitätssicherung wurde die Umgebung der isolierten Basispeaks und einzelner zufällig ausge−wählter Spinsysteme manuell kontrolliert. Dabei wurden die Parameter der Nachbarschaftsfunktion darauf getestet, ob sie alle Mitglieder eines Spinsystems erfaßt und hinreichend von den Nachbarspinsystemen trennt. Für die Experimente des Trigger Proteins waren die Standardwerte ausreichend.

5.2.3 Leere Basispeaks

Die Basispeaks [1, 8, 44, 66, 75, 85, 89, 90, 94] haben keine nahen Nachbarn in den anderen Experi−menten und wurden daher nicht in das PeakNet aufgenommen und auch später nicht zugeordnet. BP44 wurde als virtueller Basispeak für ein Spinsystem ohne Basispeak in einer Distanz von (H −0,057 ppm ; N 7,2 ppm) verwendet.

5.2.4 Konkurrenzgruppen

Im PeakNet sind 5 Konkurrenzgruppen enthalten. Diese Gruppen wurden manuell kontrolliert. Die Gruppen 1 bis 4 bestehen aus je einem Basispeakpaar, bei dem der Nachbarschaftsradius nur minimal überlappt. Die Spinsysteme sind nur über wenige (1 bis 2) nicht−Basispeaks gekoppelt und können in tri−vialer Weise getrennt werden.

Die Gruppe 0 enthält 9 Basispeaks, die in 2 eng gekoppelte Untergruppen und 3 trivial abtrennbare Basispeaks zerfällt. Sie besteht aus den Spinsystemen der Basispeaks [24, 31, 36, 37, 41, 49, 51, 53, 103]. Die Basispeaks 24, 31 und 103 sind trivial abtrennbar, da sie nur über wenige Peaks (1 bis 2) an die Gruppe gekoppelt sind und über in sich geschlossene Spinsysteme verfügen.

Die erste Untergruppe besteht aus den Basispeaks 36 und 37. Davon hat nur der Basispeak 37 ein geschlossenes Spinsystem aus 7 nicht−Basispeaks, Für Basispeak 36 bleiben dann nur 3 Peaks übrig, die ein Spinsystem mit sehr niedriger Qualität bilden. Außerdem stammen alle Peaks aus dem gleichen Experiment (CBCANH).

Die zweite Untergruppe aus den Basispeaks [41,49,53,51] füllt einen annähernd rechteckigen Bereich in der 2D HN/N−Ebene. Die Spinsysteme BP 41 und BP 49 können von der Gruppe abgetrennt werden, indem die nicht−Basispeaks dem nächstgelegenen Basispeak zugeschlagen werden, sie bilden geschlossene Spinsysteme von hoher Qualität.

Die Basispeaks 53 und 51 stehen sehr eng zusammen und können nicht getrennt werden. Die Anzahl der nicht−Basispeaks reicht nicht für zwei gute Spinsysteme. Die Verteilung der Frequenzen deutet aber auf ein gutes und ein sehr schlechtes Spinsystem hin. Das gute Spinsystem wurde willkürlich dem Basispeak 53 zugeordnet, da er dem gemittelten NH,N−Zentrum des Spinsystems näher liegt als BP 51.

5.2.5 Spinsysteme ohne Basispeaks

Während der manuellen Kontrolle wurden zwei Spinsysteme gefunden, die keinem Basispeak zugewiesen waren und auch nicht in der Nähe eines Basispeaks liegen.

Basispeaks	**enthalten in den Optimierungsstufen**	**HN [ppm]**	**N [ppm]**
44	1, 2, 3	7,59	122,2
200	1, 2, 3	8,31	115,0
201	2, 3	7,804	125,159
203	2, 3	6,92	113,75

Tabelle 5−7 Spinsysteme ohne benachbarte Basispeaks.

Das erste Spinsystem bei (HN 7,59 ppm; N 122,2 ppm) wurde dem leeren BP 44 zugewiesen, der bei (HN 7,647 ppm; N 115,0 ppm) liegt. Für das zweite Spinsystem wurde ein virtueller Basispeak BP200 bei (HN 8,31 ppm; N 115,0 ppm) erzeugt. Bei der Analyse der nicht zugewiesenen Peaks in den Optimierungsstufen 1 und 2 wurden zwei weitere Spinsysteme gefunden (BP201 und BP203).

5.2.6 Glycin Peaks

Aminosäurepaar	Anzahl	zugewiesene Basispeaks der möglichen Glycinpeaks
GX		
(hnca,cbcanh,cbca_conh)	9	21, 24, 48, 64, 97, 104, 110, 119, 121,
(cbcanh,cbca_conh)	1	125
(cbca_conh)	5	23, 34, 59, 86, 132
XG		
(hnca, cbcanh)	8	2, 7, 68, 83, 88, 93, 105, 106
XG oder GX		
(hnca)	3	30, 43, 133
Defekt (hnca * 2)	1	74
theoretische Anzahl	14	
zugewiesen in cont01	11	21,24,64,68,83,88,93,105,106,121,200

Tabelle 5−8 Vermutete Glycin−Spinsysteme

"Trigger mit HIS Tag" enthält 7 Glycine an den Aminosäurepositionen [3, 11, 33, 42, 62, 69, 72]. Position 3 liegt am Anfang des His−Tags und ist aufgrund des Abbaus wahrscheinlich nicht nachweisbar.

Mögliche Glycin Peaks wurden mit Hilfe der Glycin−Suchmuster identifiziert. Die Suchmuster finden nicht nur die Glycin−Spinsysteme, sondern auch Spinsysteme, in denen Teilspinsysteme wie ein Glycin aussehen, die aber zusammen mit den restlichen Peaks keine Glycin−Spinsystem ergeben. Ursache für die Fehlerkennung sind CB Peaks im Bereich 45 ppm von Leu und und falsche Peaks.

5.2.7 Peak Netze

Das PeakNet wurde in 3 verschiedenen Topologiestufen angelegt. Die Topologien bauen hierarchisch auf der vorhergehenden Stufe auf und erweitern diese. Als Nachbarschaftsfunktion dient zunächst (Gl. B−2). Der Nachbarschaftswert kann aber manuell beliebig gesetzt werden. Die Verknüpfungen werden mit den Programmen `'buildC21WNet'`, `'buildInnerLink'` und `'buildOuterLink'` angelegt.

Basispeak−Verknüpfung

Verknüpft werden die Basispeaks mit den, in der HN/N−Ebene, benachbarten nicht−Basispeaks.

Innere Verknüpfung

Alle nicht−Basispeaks, die zum gleichen Basispeak gehören, werden in der CA/CB−Dimension verknüpft, wenn sie die gleiche ^{13}C Frequenz besitzen.

Äußere Verknüpfung

Alle nicht−Basispeaks, auch wenn sie nicht zum gleichen Basispeak gehören, werden in der CA/CB−Dimension verknüpft, wenn sie der gleichen ^{13}C Frequenz benachbart sind.

Die Parameter für die einzelnen Verknüpfungen werden in den einzelnen Unterkapiteln beschrieben. Tabelle 5−9 faßt die Standardwerte für die Achsen zusammen.

Achse	H	N	CO	CA/CB
σ [ppm]	0.1	1.2	−	2.0 inter, 1.0 intra

Tabelle 5−9 Standardwerte des Abstandsparameters σ in [ppm] in Gleichung (Gl.B−2)

5.2.8 Suche nach Protospinsystemen

Für jeden Basispeak wurden die Protospinsysteme mit dem Programm 'genContextList_merged' gesucht. Dabei wurden die Standardparameter aus Tabelle 5−10 verwendet.

Parameter	Wert	Parameter	Wert
maximale Peakanzahl pro BP	9	aktive Substrukturfilter	ja
mindestens erforderliche Peaks (%)	95%	Muster	completeAS.noSub.small
CB Fuzzyfunktion	eng		

Tabelle 5−10 Standardparameter für die Protospinsystemsuche.

Für einen Teil der Basispeaks wurden mit den Standardparametern keine Spinsysteme gefunden. Für diese Basispeaks wurde mit einem angepaßten Parametersatz erneut gesucht.

Basispeak	CB Fuzzyfunktion	mindestens erforderliche Peaks (%)	aktive Filter	sonstiges.
34	wideCB	%40		link 1108 −> 2089
41		%85		
53		%40	keine Filter !	
86	wideCB			
106	wideCB	%65		
17	wideCB			
12	wideCB			
97	wideCB			

Tabelle 5−11 Angepaßte Parameter für einzelne Basispeaks. Für die leeren Felder wurden die Standard−werte aus Tabelle 5−10 eingesetzt.

Mit den automatisch generierten Nebenbedingungen konnte die Anzahl der Protospinsysteme je Basis−peak von 3568 (Mittelwert=34.64 je Peak) auf 158 bis 163 (Mittelwert=1.5 je Peak) reduziert werden. Dabei haben 89% der Basispeaks 2 oder weniger Protospinsysteme.

Die AS−Protospinsysteme der Basispeaks wurden mit 'genCx2ASMapping' mit Hilfe der Aminosäure−typen auf die möglichen Aminosäurepositionen verteilt. Für jede Optimierungsstufe wurde eine Konfiguration mit und eine ohne Aminosäuretypenerkennung erzeugt. Die Anzahl der Protospinsysteme je Aminosäure ist in Tabelle 5−12 aufgeführt.

Stufe	Anzahl der Protospinsy−steme je Basispeak	Anzahl je AS mit AS−Typenerkennung	Anzahl je AS ohne AS−Typenerkennung
1	1,5	14,3	158
2	1,62	14,4	162
3	1,6	14,5	162

Tabelle 5−12 Mittlere Anzahl der AS−Protospinsysteme je Aminosäure und Basispeak in den verschiedenen Optimierungsstufen.

5.2.9 Positionsqualitätsfaktoren

Name \ AS #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ohne His−Tag	0	0,1	0,1	0,1	0,2	0,3	0,3	0,3	0,5	0,8	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
ohne His−Tag, 11−20 schlecht	0	0,1	0,1	0,1	0,2	0,3	0,3	0,3	0,5	0,8	0,8	0,8	0,8	0,8	0,9	0,9	0,9	0,9	1,0	1,0

Tabelle 5−13 Positionsqualitätsfaktoren für die ersten 20 Aminosäuren. Der Faktor für die restlichen Ami−nosäuren ist 1,0.

Die Wahrscheinlichkeit der Zuweisung eines Spinsystems zu einer AS−Position im Protein wurde mit Positionsqualitätsfaktoren gewichtet.

"*ohne His−Tag*" berücksichtigt, daß der Bereich des His−Tags nicht nachgewiesen werden kann. "*ohne His−Tag ,11−20 schlecht*" nimmt zusätzlich an, daß auch die Aminosäuren 11 − 20 schlechter als alle nach−folgenden Aminosäuren zuzuweisen sind.

5.3 Optimierung des Assignments

Für die Optimierung wurde der genetische Algorithmus verwendet, da er das stabilste Konvergenzver−halten zeigt. Die Parameter des Algorithmus wurden so ausgewählt, daß sie die Konvergenzwahrscheinlich−keit für einen einzelnen Durchlauf noch weiter erhöhen und ein Optimierungszyklus über Nacht beendet werden konnte.

5.3.1 Parameter der Optimierung

Die Optimierung wurde in mehreren Stufen durchgeführt. Die einzelnen Stufen unterscheiden sich im wesentlichen in der Anzahl der Basispeaks und der Konfiguration der Spinsysteme. Zwischen den Optimie−rungen wurde jeweils eine Analyse der letzten Optimierungsstufe durchgeführt und die Konfiguration für die nachfolgende Stufe angepaßt.

Jede Stufe startet mit einer leeren Zuweisung und verwendet nicht das Ergebnis der vorhergehenden Optimierung als Anfangswert. Die Konfiguration des Optimierers ist im wesentlichen für alle Optimierungs−stufen gleich [Tabelle 5−15 bis 5−21]. Nur die Eingabedaten wurden von Stufe zu Stufe verändert.

Innerhalb jeder Stufe wurde erst eine Optimierung mit der Positionsvorauswahl durch die AS−Typenin−formation durchgeführt, die dann in einem Optimierungslauf ohne AS−Typeninformation verfeinert wurde. In

der ersten Stufe wurde außerdem die Wirkung unterschiedlicher Nachbarschaftsdistanzen in der CA/CB−Dimension getestet. Die dritte Stufe verändert zusätzlich die Positionsbeschränkungen durch die Positionsqualitätsfaktoren, indem sie die ersten 10 Aminosäuren des Proteins nach dem HIS−Tag mit einer Positionsstrafe belegt.

Stufe	**Name**	**Veränderung der Konfiguration**
Stufe 1		kein His−Tag
	fast01	ΔCA/CB=2.0, mit AS−Typen
	cont01	ΔCA/CB=*1.0*, ohne AS−Typen
	cont02	ΔCA/CB=2.0, ohne AS−Typen
Stufe 2		neue BP, kein His−Tag
	fast02	ΔCA/CB=2.0, mit AS−Typen
	cont03	ohne AS−Typen
Stufe 3		neue BP, "kein His−Tag, 11−20 schlecht"
	fast03	ΔCA/CB=2.0, mit AS−Typen
	cont04	ohne AS−Typen

Tabelle 5−14 Übersicht über die zeitliche Abfolge der einzelnen Optimierungen und ihre Verteilung auf die Stufen.

5.3.1.1 Optimierungsstufe 1

Für die erste Stufe wurde der in den vorhergehenden Kapiteln präparierte AS−Cache verwendet. Der AS−Cache enthält schon die virtuellen Spinsysteme BP44 und BP200. Die Positionen der Spinsysteme im Protein wurde mit den Positionsqualitätsfaktoren "*ohne His−Tag*" gewichtet.

Die erste Optimierung "**fast01**" startet mit einer leeren Arena und erzeugt die erste Zuweisung. "**cont01**" verfeinert das "**fast01**" Ergebnis in einem PeakNet mit niedrigerer Nachbarschaftsdistanz ohne Positionsbeschränkungen der Aminosäuren. "**cont02**" optimiert das Ergebnis von "**cont01**" in PeakNet mit normaler Nachbarschaftsdistanz. Die Konfiguration der Optimierer ist in den Tabellen Tabelle 5−15 bis 5−17 aufgeführt.

Die nachfolgenden Tabellen enthalten die Konfiguration für den genetischen Optimierer. Die erste Tabelle enthält die Basiskonfiguation, die auch für **"fast01"** verwendet wurde. Für die restlichen Optimierungen sind nur noch die Parameter aufgeführt, die relativ zu "**fast01**" verändert wurden. Diese Felder sind in allen Tabellen grau oder grün unterlegt.

Parameter	Wert	Parameter	Wert
Aufgabe:	**"Trigger, fast 01"**		
Protein:	Trigger	Startzustand:	(leer)
Cache:	default	AS−Positionsstrafen:	ohne His−Tag
Net:	inner.outer	Positionsvorwahl durch AS−Typen	JA
Distanz zum Nachbar (CA)	2.0		
Genetischer Optimierer	**(GENETIC)**		
Crossover Operator	Gs2p2	crossOverPct (Anteil)	0.15
Arena (A, E, Ü)	32 A, 16 E , 8 Ü	genqTolerance, decay (wer)	0.999, 0.05
genCrossoverThreshold	0.9		
maxLoops:	1 000 000	maxTime [min]	3600
maxIdleLoops	15	maxEqualLoops	5
SubOptimierer	**(RANDOM)**		
Profil:	mix1v2	qTolerance, decay	0.9 0.005
maxLoops:	4000	TTL (min,max)	2, 5
maxIdleLoops:	10 000	maxTime [min]	10
Finaler Optimierer	**(RANDOM)**		
Profil:	mix1v2	qTolerance, decay	0.9 0.005
maxLoops	200 000	TTL (min,max)	2, 5
maxIdleLoops	5000	maxTime [min]	20

Tabelle 5−15 Konfiguration für fast01

Parameter	Wert	Parameter	Wert
Aufgabe:	**"Trigger, cont 01"**		
Distanz zum Nachbar (CA)	1.0	Positionsvorwahl durch AS−Typen	NEIN
Startzustand:	Endzustand von fast01	AS−Positionsstrafen:	ohne His−Tag
Genetischer Optimierer	**(GENETIC)**		
crossOverPct (Anteil)	0.5		
SubOptimierer	**(RANDOM)**		
maxLoops:	32 000		
Finaler Optimierer	**(RANDOM)**	unverändert.	

Tabelle 5−16 Konfiguration für cont01

Parameter	Wert	Parameter	Wert
Aufgabe:	**"Trigger, cont 02"**		
Distanz zum Nachbar (CA)	2.0	Positionsvorwahl durch AS−Typen	NEIN
Startzustand:	Endzustand von cont01	AS−Positionsstrafen:	ohne His−Tag
Genetischer Optimierer	**(GENETIC)**		
crossOverPct (Anteil)	0.5		
SubOptimierer	**(RANDOM)**		
maxLoops:	32 000		
Finaler Optimierer	**(RANDOM)**	unverändert.	

Tabelle 5−17 Konfiguration für cont02

5.3.1.2 Optimierungsstufe 2

Die zweite Optimierungsstufe enthält 2 neue Basispeaks (siehe Tabelle 5−7). "**fast02**" startet mit einer leeren Konfiguration. Seine Lösung wird von "**cont02**" optimiert.

Parameter	**Wert**	**Parameter**	**Wert**
Aufgabe:	**"Trigger, fast 02"**		
Distanz zum Nachbar (CA)	2.0	Positionsvorwahl durch AS−Typen	JA
Startzustand:	Endzustand von cont02	AS−Positionsstrafen:	ohne His−Tag
Genetischer Optimierer	**(GENETIC)**		
crossOverPct (Anteil)	0.5		
SubOptimierer	**(RANDOM)**		
maxLoops:	32 000		
Finaler Optimierer	**(RANDOM)**	unverändert.	

Tabelle 5−18 Konfiguration für fast02

Parameter	**Wert**	**Parameter**	**Wert**
Aufgabe:	**"Trigger, cont 03"**		
Distanz zum Nachbar (CA)	2.0	Positionsvorwahl durch AS−Typen	NEIN
Startzustand:	Endzustand von fast02	AS−Positionsstrafen:	ohne His−Tag
Genetischer Optimierer	**(GENETIC)**		
crossOverPct (Anteil)	0.5		
SubOptimierer	**(RANDOM)**		
maxLoops:	32 000		
Finaler Optimierer	**(RANDOM)**	unverändert.	

Tabelle 5−19 Konfiguration für cont03

5.3.1.3 Optimierungsstufe 3

Stufe 3 verwendet die Positionsgewichtung "kein HIS−Tag, 11−20 schlecht". Die erste Lösung ist "**fast03**". "**cont04**" verwendet den Endzustand von "**fast03**" als Startwert und optimiert ihn.

Parameter	**Wert**	**Parameter**	**Wert**
Aufgabe:	**"Trigger, fast 03"**		
Distanz zum Nachbar (CA)	2.0	Positionsvorwahl durch AS−Typen	JA
Startzustand:	Endzustand von cont03	AS−Positionsstrafen:	ohne His−Tag,11 bis 20 schlecht
Genetischer Optimierer	**(GENETIC)**		
crossOverPct (Anteil)	0.5	maxTime [min]:	240
SubOptimierer	**(RANDOM)**		
maxLoops:	32 000		
Finaler Optimierer	**(RANDOM)**	unverändert.	

Tabelle 5−20 Konfiguration für fast04

Parameter	Wert	Parameter	Wert
Aufgabe:	**"Trigger, cont 04"**		
Distanz zum Nachbar (CA)	2.0	Positionsvorwahl durch AS−Typen	NEIN
Startzustand:	Endzustand von fast03	AS−Positionsstrafen:	ohne His−Tag,11 bis 20 schlecht
Genetischer Optimierer	**(GENETIC)**		
crossOverPct (Anteil)	0.5	maxTime [min]:	3600
SubOptimierer	**(RANDOM)**		
maxLoops:	32 000		
Finaler Optimierer	**(RANDOM)**	unverändert.	

Tabelle 5−21 Konfiguration für cont04

Folgende Seiten:

Tabellen 5−22 bis 5−25: Vergleich des manuellen Assignments für Trigger »Nachher« mit dem automatischen Assignment "cont02". Die erste und zweite Tabelle zeigen »Nachher«, die dritte und vierte »Cont02«. Felder mit unterschiedlichem Inhalt sind farbig unterlegt. Die Frequenzen werden mit einer Toleranz von Δ=0.05 ppm *verglichen, die Qualitäten mit einer Toleranz von* Δ=0.05 *Einheiten. Die Identifikationsnummern der Peaks müssen exakt übereinstimmen.*

Farbkodierung:

Weiß = gleicher Inhalt in beiden Tabellen.

Blau = fehlt in dieser Tabelle, zugewiesen in der anderen.

Grün = hier zugewiesen, fehlt in der anderen Tabelle

Gelb = unterschiedliche Zuweisung in den Tabellen, beide zugewiesen.

AS3Char		AS ID	CA−1	CB−1	CA	CB	LINK_Q	Q	cbca(CO)nh CA−1	cbca(CO)nh CB−1	cbcanh CA	cbcanh CA−1	cbcanh CB	cbcanh CB−1	hnca CA		Vol. Verhältnis hnca CA−1	hnc
Met	1																	
Arg	2																	
Gly	3																	
Ser	4																	
His	5																	
His	6																	
His	7																	
His	8																	
His	9																	
His	10																	
Gly	11																	
Ser	12																	
Glu	13	50.890	63.085	56.893	29.507		2.06		2088	3097		3099	3096	1111		1112	30	
Lys	14																	
Leu	15																	
Ala	16	54.602	41.689	52.166	19.149	2.47	2.30	2061	2060	3066	3065	3068	3067	1126		1127	103	
Lys	17	52.080	19.171	57.398	33.583	3.49	1.84			3234	3233	3232	3231				201	
Thr	18	55.870	32.525	61.587	69.385		2.26	2225	2226	3218	3217	3219	3216	1021		1020	120	
Lys	19	57.076	29.460	57.405	33.531		0.93	2045	2046	3051		3052	3339			1155	101	
Ser	20																	
Thr	21	57.802	63.374	61.667	69.171	3.11	2.56	2205	2206	3221	3220	3223		1023			77	
Met	22	61.317	69.392	56.053	32.712	4.19	2.07			3102	3101	3103	3100	1119		1120	25	
Val	23	55.078	32.482	61.958		2.68	1.35	2130	2129	3278	3277		3276	1082		1081	112	
Asp	24	61.757	32.496	53.806	41.055	4.49	2.02	2069	2066	3070	3069	3071		1125		1124	23	
Val	25	53.825	40.937	62.237	32.194	4.40	2.10	2146	2142	3261	3260	3169	3258	1170		1169	51	
Ser	26	62.006	31.677	59.214	63.266	4.38	2.45	2170	2169	3287	3289	3290	3288	1051		1052	127	
Asp	27	59.086	63.147	53.642	41.044	4.44	2.61	2113	2114	3110	3111	3109	3112	1094		1095	41	
Lys	28	53.457	40.925	56.698	33.241	4.05	2.04	2109	2110	3061	3062	3064	3063	1090		1089	44	
Lys	29	56.906	32.849	53.683	35.082	4.46	1.51	2092	2091	3271		3270	3269	1105		1106	37	
Leu	30	53.395	34.752	55.978	42.592	4.49	1.59	2076	2075	3092	3093	3094	3095	1117		1118	27	
Ala	31	55.894	42.473	49.951	22.656	4.52	2.41	2004	2003	3251	3252	3249	3253	1163		1164	2	
Asn	32	49.980	22.633	55.336	38.154	4.49	2.50	2198	2197	3291	3292	3293	3294	1029	<	1030	119	
Gly	33	55.271	37.889	44.833		3.61	2.46	2215	2216	3129	3128		3130	1016		1017	83	
Asp	34	44.871		54.564	42.248	4.54	2.24	2149		3254	3255	3256		1066		1067	106	
Ile	35	54.507	42.236	61.186	37.787	4.43	1.63	2157	2156	3198	3197	3195	3196	1064		1063	116	
Ala	36	61.136	37.610	49.661	22.175	4.46	2.33	2013	2014	3002	3001	3004	3003	1161		1160	7	
Ile	37	49.674	21.711	59.769	36.080	4.39	2.14	2126	2125	3174	3173	3172	3171	1086		1085	48	
Ile	38	59.470	35.951	58.875	42.137	3.74	1.05	2160	2161	3329	3328	3327	3326			1065	131	
Asp	39	58.556	42.134	52.336		2.77	1.25	2162	2164	3200	3199		3201	1061		1062	117	
Phe	40	52.318	42.402	55.527		2.77	1.61	2218	2217	3325	3324		3323	1019		1018	84	
Thr	41	55.505	42.694	61.014	71.243	4.46	2.19	2174	2173	3189	3188	3190	3187	1054		1053	63	
Gly	42	60.928	71.004	45.335		4.01	2.85	2193	2194	3204	3203		3202	1034		1035	68	
Ile	43	45.202		60.037	39.202	4.37	2.11	2058		3048	3049	3050		1130	<	1131	21	
Val	44	60.151	39.113	63.428	34.745		1.37	2030	2029		3033	3035	3034	1150	<	1149	98	
Asp	45	60.350	34.506	55.676	38.929	2.80	1.74	2019	2020	3010	3009	3011	3012	1157		1158	96	
Asn	46	55.668	38.441	54.882		1.62	0.47	2257	2256					1001			109	
Lys	47	54.486	37.651		34.904	2.32	1.37	2148	2147		3115	3116	3117			1069	57	
Lys	48	54.516	34.817	56.632	32.126	4.44	1.61	2085	2086	3265	3264	3262	3263	1115		1116	28	
Leu	49	56.575	31.817	52.914	42.996		1.58	2035	2036	3037	3036	3038	3039	1147		1148	12	
Ala	50																	
Ser	51																	
Ala	52	58.850	61.855	50.188		1.65	1.98	2062	2063					1129		1128	133	
Ser	53	50.081	19.588	57.447	66.217	4.43	2.98	2248	2247	3139	3140	3138	3141	1006		1007	110	
Ala	54	57.208	66.183	52.133	23.164		2.97	2108	2107	3273	3274	3272	3275	1096		1097	124	
Gln	55																	
Asn	56	53.625	31.848	50.753	36.887		1.48	2111	2112		3079	3080	3081	1092	<	1091	43	
Tyr	57	54.284	36.669	58.110	40.289	3.83	1.64	2096	2095	3085	3086	3087	3088	1099		1100	39	

Tabelle 5−22 Manuelle Lösung, Teil 1

AS3Char	AS ID	CA−1	CB−1	CA	CB	LINK_Q	Q	cbca(CO)nh CA−1	cbca(CO)nh CB−1	cbcanh CA	cbcanh CA−1	cbcanh CB	cbcanh CB−1	hnca CA	Vol_Verhältnis	hnca CA−1	hnc
Glu	58	58.036	40.137	54.884	29.892	3.41	1.07	2032	2031	3058	3057	3060	3059				102
Leu	59	53.595	29.812	55.104		1.63	0.82	2024	2023	3008	3007		3005			1159	9
Thr	60	55.481	42.181		30.593		0.90	2097	2098		3089	3091	3090				33
Ile	61	61.121	67.723	62.815	33.583		1.74	2005	2006	3244	3248	3243	3247	1166	<	1165	3
Gly	62	60.561	33.571	45.170		3.85	1.67	2260	2259	3083	3082		3084	1075	<	1074	105
Ser	63	44.882		59.232	64.328	3.73	2.97	2203		3225	3224	3226		1025		1024	121
Asn	64	59.006	64.400	53.853	36.548	4.33	2.00	2073	2074	3299	3301	3300	3302	1121			128
Ser	65	54.307	36.519	59.105	64.153	2.71	2.31	2228	2227	3132	3133	3131	3134	1010		1011	87
Phe	66	58.141	63.971	55.419	40.664	2.81	1.68	2079	2080	3107		3108		1171			31
Ile	67	55.128	40.384				0.38	2212	2211								130
Lys	68	61.384	39.262	58.678	32.013	4.44	1.61	2041	2042	3041	3040	3043		1138		1139	99
Gly	69	58.673	31.910	44.869		3.92	2.22	2230	2229	3126	3125		3127	1012		1013	88
Phe	70	44.645		60.615	40.033	4.35	2.23	2072		3104	3105	3106		1113		1114	24
Glu	71	60.086	39.865		29.908	2.32	1.11	2209	2210		3122	3124	3123			1027	108
Thr	72	59.890	29.799	64.614	67.831	4.41	2.35	2255	2254	3297	3296	3298	3295	1002		1003	92
Gly	73	64.233	67.862	45.713		4.12	2.86	2251	2258	3135	3136		3137	1004		1005	93
Leu	74	45.582		54.288	41.088	4.44	2.21	2179		3146	3147	3148		1047		1048	64
Ile	75	54.310	41.014	63.708	36.518	4.44	2.00	2182	2183	3142	3143	3144	3145	1042		1041	72
Ala	76	63.456	36.453	54.175	16.770	4.50	2.72	2093	2094	3310	3311	3308	3309	1109		1110	32
Met	77	54.198	16.687	57.479	35.345	4.41	2.14	2167	2168	3156	3157	3158	3159	1058		1057	61
Lys	78	57.407	35.121	53.619		2.75	1.12	2187	2186	3154	3153		3155	1039		1040	67
Val	79	53.510	35.108	65.686	30.886	4.46	1.59	2152	2153	3186	3185	3183	3184	1071		1070	114
Asn	80	65.516	30.864	55.641	36.660	3.43	1.92	2177	2178	3119	3118	3120	3121	1049	<	1050	107
Gln	81	55.842	36.578	56.234		1.66	1.02	2143	2141				3257	1078			49
Lys	82	56.104	29.654	53.580	35.064	4.41	1.41	2137	2138	3165	3164	3166	3167	1084		1172	113
Lys	83	53.710	34.973	55.567	37.620	4.44	1.28	2054	2055	3075	3076	3077	3078	1134		1135	18
Thr	84	55.461	37.512	60.955	69.340	4.47	2.10	2151	2150	3181	3180	3182	3179	1073		1072	56
Leu	85	60.798	69.296	53.167	44.883	4.47	2.66	2039	2040	3019	3018	3020	3017	1142		1143	17
Ala	86	53.221	44.712	51.275	18.606	4.42	2.66	2028	2027	3030	3029	3032	3031	1151		1152	97
Leu	87	50.834	18.430	52.621	45.697	4.45	2.65	2065	2064	3306	3305	3304	3303	1122		1123	104
Thr	88	52.782	45.480	61.092	71.276	4.49	2.66	2220	2219	3229	3228	3230	3227	1015		1014	86
Phe	89	61.078	71.012	58.313	38.438		2.36	2047	2048	3046	3045	3047	3044	1136		1137	100
Pro	90																
Ser	91	61.971	32.688	60.248	62.945	3.76	2.41	2175	2176	3192	3193	3194	3191	1055		1056	115
Asp	92	60.246	62.511	51.322	39.280	3.09	1.86	2190			3210	3212	3209	1032		1033	74
Tyr	93	51.331	39.337	59.351	38.027	3.29	1.12	2154	2155	3331	3330	3332	3333	1068			132
His	94	59.215	38.611	58.710	28.959	3.84	0.92	2134	2133	3340		3342	3341				134
Val	95	58.400	28.803	61.323	30.644	4.35	1.43	2159	2158	3337	3336	3335	3334	1077		1076	55
Lys	96	60.989	30.608	59.397	32.300		1.42	2009	2010	3026	3025	3027	3028			1162	5
Glu	97																
Leu	98	58.450	28.230	53.432	42.159	4.45	1.56	2181	2180	3150	3149	3151	3152	1037		1038	70
Gln	99	53.180	42.062	57.968	27.418	4.41	1.94	2184	2185	3286	3285	3283	3284	1045		1046	126
Ser	100	57.717	27.071	60.197	60.937	2.19	2.51	2241	2242	3320	3317	3319	3316	1009	<	1008	129
Lys	101	60.711		53.394	31.377		0.81	2127		3113		3114		1079		1080	47
Pro	102																
Val	103	62.361	31.566	58.180	33.724	1.35	1.56	2123	2124	3177	3178	3176	3175	1087		1088	46
Thr	104	58.322		60.155	70.172	3.76	1.30			3267	3268	3266					36
Phe	105	60.237	69.927	56.017	42.423		2.20	2077	2078	3313	3314	3312	3315	1103		1104	35
Glu	106																
Val	107	55.334	30.414	60.664	34.488	4.40	1.75	2034	2033	3021	3022	3023	3024	1141		1140	15
Val	108	60.392	34.359	63.443			1.37	2056	2057		3073		3074	1133	<	1132	20
Leu	109	60.100	35.162	54.239	41.303	4.08	1.76	2038	2037	3054	3053	3055	3056	1145		1146	16
Lys	110	54.016	41.234	55.686	31.229	3.80	1.18	2026	2025	3013		3016	3015	1154		1153	135
Ala	111	55.667	31.020	51.890	21.382	3.77	1.66	2189	2188	3215	3213	3214				1043	65
Ile	112	51.830	21.405	60.941	40.440	4.47	2.49	2165	2166	3160	3161	3162	3163	1060		1059	59
Lys	113	60.811	40.275	57.786	33.747		1.90	2002	2001	3241	3242	3239	3240	1167		1168	122
stat	−−−	91	84	87	76	75		87	81	80	80	76	76	78	78	79	91

Tabelle 5−23 Manuelle Lösung, Teil 2

AS3Char	AS ID	CA−1	CB−1	CA	CB	LINK_Q	Q	cbca(CO)nh CA−1	cbca(CO)nh CB−1	cbcanh CA	cbcanh CA−1	cbcanh CB	cbcanh CB−1	hnca CA		hnca CA−1	hnco
Met	1																
Arg	2																
Gly	3																
Ser	4																
His	5																
His	6																
His	7																
His	8																
His	9																
His	10																
Gly	11																
Ser	12	44.64		58.34		1.7	1.63	2204		3236	3235			1026			200
Glu	13	58.14	63.97	55.42	40.66	2.81	1.47	2079	2080	3107		3108		1171			31
Lys	14	55.13	40.38				0.21	2212	2211								130
Leu	15																
Ala	16	52.87	45.26	62.74		1.64	1.28	2089	2090					1107		1108	34
Lys	17	62.36	31.57	58.18	33.72	1.35	1.39	2123	2124	3177	3178	3176	3175	1087		1088	46
Thr	18	58.32		60.16	70.17	2.66	1.36			3267	3268	3266					36
Lys	19	61.13	68.68	55.37		1.67	1.69	2099	2100					1102		1101	125
Ser	20	55.35	32.46				0.5	2101	2102							1098	123
Thr	21	57.80	63.37	61.67	69.17	3.11	2.56	2205	2206	3221	3220	3223		1023			77
Met	22	61.32	69.39	56.05	32.71	4.19	2.07			3102	3101	3103	3100	1119		1120	25
Val	23	55.08	32.48	61.96		2.68	1.35	2130	2129	3278	3277		3276	1082		1081	112
Asp	24	61.76	32.50	53.81	41.06	4.49	2.02	2069	2066	3070	3069	3071		1125		1124	23
Val	25	53.83	40.94	62.24	32.19	4.40	2.10	2146	2142	3261	3260	3169	3258	1170		1169	51
Ser	26	62.01	31.68	59.21	63.27	3.74	2.45	2170	2169	3287	3289	3290	3288	1051		1052	127
Asp	27	59.44	63.15	53.64	41.04	4.44	2.49		2114	3110	3111	3109	3112	1094		1095	41
Lys	28	53.46	40.93	56.70	33.24	4.05	2.04	2109	2110	3061	3062	3064	3063	1090		1089	44
Lys	29	56.91	32.85	53.68	35.08	4.46	1.51	2092	2091	3271		3270	3269	1105		1106	37
Leu	30	53.40	34.75	55.98	42.59	4.49	1.59	2076	2075	3092	3093	3094	3095	1117		1118	27
Ala	31	55.89	42.47	49.95	22.66	4.52	2.41	2004	2003	3251	3252	3249	3253	1163		1164	2
Asn	32	49.98	22.63	55.34	38.15	4.49	2.50	2198	2197	3291	3292	3293	3294	1029	<	1030	119
Gly	33	55.27	37.89	44.83		2.67	2.46	2215	2216	3129	3128		3130	1016		1017	83
Asp	34	45.23		54.56	42.25	4.54	2.11			3254	3255	3256		1066		1067	106
Ile	35	54.51	42.24	61.19	37.79	4.43	1.63	2157	2156	3198	3197	3195	3196	1064		1063	116
Ala	36	61.14	37.61	49.66	22.18	4.46	2.33	2013	2014	3002	3001	3004	3003	1161		1160	7
Ile	37	49.67	21.71	59.77	36.08	4.39	2.14	2126	2125	3174	3173	3172	3171	1086		1085	48
Ile	38	59.47	35.95	58.88	42.14	3.74	1.05	2160	2161	3329	3328	3327	3326			1065	131
Asp	39	58.56	42.13	52.34		2.77	1.25	2162	2164	3200	3199		3201	1061		1062	117
Phe	40	52.32	42.40	55.53		2.77	1.61	2218	2217	3325	3324		3323	1019		1018	84
Thr	41	55.51	42.69	61.01	71.24	4.46	2.19	2174	2173	3189	3188	3190	3187	1054		1053	63
Gly	42	60.93	71.00	45.34		4.01	2.85	2193	2194	3204	3203		3202	1034		1035	68
Ile	43	45.20		60.04	39.20	4.37	2.11	2058		3048	3049	3050		1130	<	1131	21
Val	44	60.15	39.11	63.43	34.75		1.37	2030	2029		3033	3035	3034	1150	<	1149	98
Asp	45	60.35	34.51	55.68	38.93	2.80	1.74	2019	2020	3010	3009	3011	3012	1157		1158	96
Asn	46	55.67	38.44	54.88		1.62	0.47	2257	2256					1001			109
Lys	47	54.49	37.65		34.90	2.32	1.37	2148	2147		3115	3116	3117			1069	57
Lys	48	54.52	34.82	56.63	32.13	4.44	1.61	2085	2086	3265	3264	3262	3263	1115		1116	28
Leu	49	56.58	31.82	52.91	43.00	4.24	1.58	2035	2036	3037	3036	3038	3039	1147		1148	12
Ala	50	53.18	42.06	57.97	27.42	4.41	1.22	2184	2185	3286	3285	3283	3284	1045		1046	126
Ser	51	57.72	27.07	60.2	60.94	3.23	1.77	2241	2242	3320	3317	3319	3316	1009	<	1008	129
Ala	52	58.85	61.86	50.19		1.65	1.98	2062	2063					1129		1128	133
Ser	53	50.08	19.59	57.45	66.22	4.43	2.98	2248	2247	3139	3140	3138	3141	1006		1007	110
Ala	54	57.21	66.18	52.13	23.16	3.79	2.97	2108	2107	3273	3274	3272	3275	1096		1097	124
Gln	55	52.26	22.91	53.6	32.07	4.46	1.83	2195	2196	3205	3206	3207	3208	1031			118
Asn	56	53.63	31.85	50.75	36.89		1.48	2111	2112		3079	3080	3081	1092	<	1091	43
Tyr	57	54.28	36.67	58.11	40.29	3.83	1.64	2096	2095	3085	3086	3087	3088	1099		1100	39

Tabelle 5−24 Automatisches Assignment "cont02", Teil 1

AS3Char	AS ID	CA−1	CB−1	CA	CB	LINK_Q	Q	cbca(CO)nh CA−1	cbca(CO)nh CB−1	cbcanh CA	cbcanh CA−1	cbcanh CB	cbcanh CB−1	hnca CA		hnca CA−1	hnco
Glu	58	58.04	40.14	54.88	29.89	3.47	1.070	2032	2031	3058	3057	3060	3059				102
Leu	59	54.83	29.81	53.11			0.9	2024	2023	3007	3008		3005	1159			9
Thr	60	50.89	63.09	56.89	29.51	3.68	0.76		2088	3097		3099	3096	1111		1112	30
Ile	61	56.84	29.46	60.71	33.53	3.42	0.48	2045	2046		3051	3052	3339	1156			101
Gly	62	60.56	33.57	45.17		3.850	1.670	2260	2259	3083	3082		3084	1075	<	1074	105
Ser	63	44.88		59.23	64.33	3.730	2.970	2203		3225	3224	3226		1025		1024	121
Asn	64	59.01	64.40	53.85	36.55	4.330	2.000	2073	2074	3299	3301	3300	3302	1121			128
Ser	65	54.31	36.52	59.11	64.15	2.12	2.310	2228	2227	3132	3133	3131	3134	1010		1011	87
Phe	66	60.71		53.39	31.38	2.55	0.77	2127		3113		3114		1079		1080	47
Ile	67	55.23	31.94	61.84		1.64	0.43	2117	2118					1083			111
Lys	68	61.38	39.26	58.68	32.01	4.440	1.610	2041	2042	3041	3040	3043		1138		1139	99
Gly	69	58.67	31.91	44.87		3.920	2.220	2230	2229	3126	3125		3127	1012		1013	88
Phe	70	44.65		60.62	40.03	4.350	2.230	2072		3104	3105	3106		1113		1114	24
Glu	71	60.09	39.87		29.91	2.320	1.110	2209	2210		3122	3124	3123			1027	108
Thr	72	59.89	29.80	64.61	67.83	4.410	2.350	2255	2254	3297	3296	3298	3295	1002		1003	92
Gly	73	64.23	67.86	45.71		4.120	2.860	2251	2258	3135	3136		3137	1004		1005	93
Leu	74	45.58		54.29	41.09	4.440	2.210	2179		3146	3147	3148		1047		1048	64
Ile	75	54.31	41.01	63.71	36.52	3.77	2.000	2182	2183	3142	3143	3144	3145	1042		1041	72
Ala	76	63.69	36.45	54.18	16.77	4.500	2.66		2094	3310	3311	3308	3309	1109		1110	32
Met	77	54.20	16.69	57.48	35.35	4.410	2.140	2167	2168	3156	3157	3158	3159	1058		1057	61
Lys	78	57.41	35.12	53.62		2.750	1.120	2187	2186	3154	3153		3155	1039		1040	67
Val	79	53.51	35.11	65.69	30.89	4.460	1.590	2152	2153	3186	3185	3183	3184	1071		1070	114
Asn	80	65.52	30.86	55.64	36.66	3.430	1.920	2177	2178	3119	3118	3120	3121	1049	<	1050	107
Gln	81	55.84	36.58	56.23		1.660	1.020	2143	2141				3257	1078			49
Lys	82	56.10	29.65	53.58	35.06	4.410	1.410	2137	2138	3165	3164	3166	3167	1084		1172	113
Lys	83	53.71	34.97	55.57	37.62	4.440	1.280	2054	2055	3075	3076	3077	3078	1134		1135	18
Thr	84	55.46	37.51	60.96	69.34	4.470	2.100	2151	2150	3181	3180	3182	3179	1073		1072	56
Leu	85	60.80	69.30	53.17	44.88	4.470	2.660	2039	2040	3019	3018	3020	3017	1142		1143	17
Ala	86	53.22	44.71	51.28	18.61	3.71	2.660	2028	2027	3030	3029	3032	3031	1151		1152	97
Leu	87	50.6	18.43	52.62		2.69	1.89		2064	3306	3305		3303	1122		1123	104
Thr	88	52.78	45.48	61.09	71.28	4.490	2.660	2220	2219	3229	3228	3230	3227	1015		1014	86
Phe	89	61.08	71.01	58.31	38.44		2.360	2047	2048	3046	3045	3047	3044	1136		1137	100
Pro	90																
Ser	91	61.97	32.69	60.25	62.95	3.760	2.410	2175	2176	3192	3193	3194	3191	1055		1056	115
Asp	92	60.25	62.51	51.32	39.28	3.090	1.860	2190			3210	3212	3209	1032		1033	74
Tyr	93	51.33	39.34	59.35	38.03	2.7	1.120	2154	2155	3331	3330	3332	3333	1068			132
His	94	59.22	38.49				0.58	2134	2133								134
Val	95	58.40	28.80	61.32	30.64	4.350	1.430	2159	2158	3337	3336	3335	3334	1077		1076	55
Lys	96	60.99	30.61	59.40	32.30	3.28	1.420	2009	2010	3026	3025	3027	3028			1162	5
Glu	97	59.82	31.67	58.6	28.1	4.44	1.4	2208	2207	3281	3282	3279	3280	1022			60
Leu	98	58.45	28.23	53.43	42.16	3.61	1.560	2181	2180	3150	3149	3151	3152	1037		1038	70
Gln	99	54.42	41.69	52.17	19.15		0.98	2061	2060	3066	3065	3068	3067	1126			103
Ser	100	55.35	18.03				0.21	2252	2253								91
Lys	101																
Pro	102																
Val	103	60.66	67.72	62.82	33.58		1.12		2006	3244	3248	3243	3247	1166	<	1165	3
Thr	104	55.87	32.53	61.59	69.39	4.09	1.98	2225	2226	3218	3217	3219	3216	1021		1020	120
Phe	105	60.24	69.93	56.02	42.42	3.37	2.200	2077	2078	3313	3314	3312	3315	1103		1104	35
Glu	106	55.33	42.18	55.63	30.59	3.81	0.9	2097	2098	3089		3091	3090				33
Val	107	55.33	30.41	60.66	34.49	4.400	1.750	2034	2033	3021	3022	3023	3024	1141		1140	15
Val	108	60.39	34.36	63.44			1.370	2056	2057		3073		3074	1133	<	1132	20
Leu	109	60.10	35.16	54.24	41.30	4.080	1.760	2038	2037	3054	3053	3055	3056	1145		1146	16
Lys	110	54.02	41.23	55.69	31.23	3.800	1.180	2026	2025	3013		3016	3015	1154		1153	135
Ala	111	55.67	31.02	51.89	21.38	3.770	1.660	2189	2188	3215	3213	3214				1043	65
Ile	112	51.83	21.41	60.94	40.44	4.470	2.490	2165	2166	3160	3161	3162	3163	1060		1059	59
Lys	113	60.81	40.28	57.79	33.75		1.900	2002	2001	3241	3242	3239	3240	1167		1168	122
stat	---	98	90	92	75	86		90	88	81	82	75	76	86	86	79	98

Tabelle 5−25 Automatisches Assignment "cont02", Teil 2

ID	AS	Konkurrierende	Zugewiesene HNCO ID								
		Spinsysteme	Stufe1			Stufe2		Stufe3		Manuell	
			Fast	Cont1	Cont2	Fast2	Cont3	Fast3	Cont4	Vor	Nach
1	Met										
2	Arg										
3	Gly										
4	Ser										
5	His										
6	His										
7	His										
8	His										
9	His	47						47			
10	His	47							47		
11	Gly	200, 111					200		111		
12	Ser	134, 200	200	200	200	200	134	200			
13	Glu	30, 31	30	31	31	30	30	30	30	30	30
14	Lys	55, 101, 101a, 130	101	130	130	55	55	101	101		101 a
15	Leu	12, 16, 23, 123	12			23	23	123	16		
16	Ala	34, 103		34	34	103	103	103	103	103	103
17	Lys	46, 201		46	46	201	201	201	201		201
18	Thr	36, 120	120	120	36	120	120	120	120	120	120
19	Lys	3, 25, 101 b, 125	25	25	125	25	25	3	3	101 b	101 b
20	Ser	111,123, 200	111	123	123	111	111	111	200		
21	Thr		77	77	77	77	77	77	77	77	77
22	Met	25, 125	125	125	25	125	125	25	25	25	25
23	Val	15, 55, 112	55	112	112	15	15	112	112	112	112
24	Asp	16, 23	23	23	23	16	16	23	23	23	23
25	Val	49, 51	51	51	51	51	51	51	51		51
26	Ser		127	127	127	127	127	127	127	127	127
27	Asp		41	41	41	41	41	41	41	41	41
28	Lys		44	44	44	44	44	44	44	44	44
29	Lys		37	37	37	37	37	37	37	37	37
30	Leu		27	27	27	27	27	27	27	27	27
31	Ala		2	2	2	2	2	2	2	2	2
32	Asn		119	119	119	119	119	119	119	119	119
33	Gly		83	83	83	83	83	83	83	83	83
34	Asp		106	106	106	106	106	106	106	106	106
35	Ile		116	116	116	116	116	116	116	116	116
36	Ala		7	7	7	7	7	7	7	7	7
37	Ile		48	48	48	48	48	48	48	48	48
38	Ile		131	131	131	131	131	131	131	131	131
39	Asp		117	117	117	117	117	117	117	117	117
40	Phe		84	84	84	84	84	84	84	84	84
41	Thr		63	63	63	63	63	63	63	63	63
42	Gly		68	68	68	68	68	68	68	68	68
43	Ile		21	21	21	21	21	21	21	21	21
44	Val		98	98	98	98	98	98	98	98	98
45	Asp		96	96	96	96		96	96	96	96
46	Asn		109	109	109	109	96	109	109	109	109
47	Lys		57	57	57	57	57	57	57	57	57
48	Lys		28	28	28	28	28	28	28	28	28
49	Leu	12, 123	123	12	12	123	123	12	12	12	12
50	Ala	103, 109, 126	103	103	126		109		126		???
51	Ser	91, 129		91	129		91		129		???
52	Ala			133	133				133	133	133
53	Ser		110	110	110	110	110	110	110	110	110
54	Ala		124	124	124	124	124	124	124	124	124
55	Gln		118	118	118	118	118	118	118	118	118
56	Asn		43	43	43	43	43	43	43	43	43
57	Tyr		39	39	39	39	39	39	39	39	39
58	Glu		102	102	102	102	102	102	102	102	102
59	Leu		9	9	9	9	9	9	9	9	9

ID	AS		Zugewiesene HNCO ID								
			Stufe1			Stufe2		Stufe3		Manuell	
			Fast	Cont1	Cont2	Fast2	Cont3	Fast3	Cont4	Vor	Nach
60	Thr	30, 33, 34	34	30	30	34	34	86	34	33	33
61	Ile	3, 100, 101, 125	3	101	101	3	3	100	125	3	3
62	Gly		105	105	105	105	105	105	105	105	105
63	Ser		121	121	121	121	121	121	121	121	121
64	Asn		128	128	128	128	128	128	128	128	128
65	Ser		87	87	87	87	87	87	87	87	87
66	Phe	31,47	31		47	31	31	31	31	31	31
67	Ile	111, 130		111	111		130		130		130
68	Lys		99	99	99	99	99	99	99	99	99
69	Gly		88	88	88	88	88	88	88	88	88
70	Phe		24	24	24	24	24	24	24	24	24
71	Glu		108	108	108	108	108	108	108	108	108
72	Thr		92	92	92	92	92	92	92	92	92
73	Gly		93	93	93	93	93	93	93	93	93
74	Leu		64	64	64	64	64	64	64	64	64
75	Ile		72	72	72	72	72	72	72	72	72
76	Ala		32	32	32	32	32	32	32	32	32
77	Met		61	61	61	61	61	61	61	61	61
78	Lys		67	67	67	67	67	67	67	67	67
79	Val		114	114	114	114	114	114	114	114	114
80	Asn		107	107	107	107	107	107	107	107	107
81	Gln		49	49	49	49	49	49	49	51	49
82	Lys		113	113	113	113	113	113	113		113
83	Lys		18	18	18	18	18	18	18	18	18
84	Thr		56	56	56	56	56	56	56	56	56
85	Leu		17	17	17	17	17	17	17	17	17
86	Ala		97	97	97	97	97	97	97	97	97
87	Leu		104	104	104	104	104	104	104	104	104
88	Thr	34, 86	86	86	86	86	86	34	86	86	86
89	Phe	100, 125	100	100	100	100	100	125	100	100	100
90	Pro									−P−	−P−
91	Ser		115	115	115	115	115	115	115	115	115
92	Asp		74	74	74	74	74	74	74	74	74
93	Tyr		132	132	132	132	132	132	132	132	132
94	His	47, 134	47	134	134	47	47	134	134	134	134
95	Val	55, 112	112	55	55	112	112	55	55	55	55
96	Lys		5	5	5	5	5	5	5	5	5
97	Glu		60	60	60	60	60	60	60	60	60
98	Leu		70	70	70	70	70	70	70	70	70
99	Gln		126	126	103	126	126	126	135	126	126
100	Ser	91, 123, 129	129	129	91	129	129	129	123	129	129
101	Lys	47, 91, 133	133	47		133	133	133	91	47	47
102	Pro									−P−	−P−
103	Val	3, 46	46	3	3	46	46	46		46	46
104	Thr	36, 120	36	36	120	36	36	36	36	36	36
105	Phe		35	35	35	35	35	35	35	35	35
106	Glu	33	33	33	33	33	33	33	33		???
107	Val	15, 101	15	15	15	101	101	15	15	15	15
108	Val		20	20	20	20	20	20	20	20	20
109	Leu	12, 16, 46	16	16	16	12	12	16	46	16	16
110	Lys		135	135	135	135	135	135		135	135
111	Ala		65	65	65	65	65	65	65	65	65
112	Ile		59	59	59	59	59	59	59	59	59
113	Lys		122	122	122	122	122	122	122	122	122

Tabelle 5−26 Übersicht der Endzustände. Grüne Felder im automatischen Assignment markieren Unterschiede zur manuellen Zuweisung. Hellblaue Felder in der manuellen Zuweisung markieren Verbesserungen, die während des Vergleichs mit den automatischen Zuweisungen gefunden wurden.

5.3.3 Vergleich der Endzustände

Die berechneten Lösungen weisen alle mehr Aminosäurepositionen zu als im manuellen Assignment benutzt werden. Außerdem nutzen sie mehr Spinsysteme und Peaks.

	Manuell		**Stufe 1**			**Stufe 2**		**Stufe 3**	
	Vorher	**Nachher**	**fast01**	**cont01**	**cont02**	**fast02**	**cont03**	**fast03**	**cont04**
Q	430,436	467,243	473,32	465,99	481,27	482,56	489,56	482,23	491,86
zugewiesene AS	89	93	95	98	98	96	99	97	99
entstehende Links	70	79	81	80	86	84	87	84	89
erklärte Peaks	585	602	612	615	615	616	621	617	625
ungenutzte Spinsy−steme	14	9	6	3	3	5	2	4	2
Anz. erfüllte Regeln	475	502	510	524	529	524	517	517	527
gleiches Spinsystem mit "Nachher" ?	107	(alle)	99	100	94	95	91	103	98
Laufzeit									
parallele Zeit [min]			17	22	13	19	10	na	25
lineare Zeit [min]			572	713	409	619	321	na	803
genetische Zyklen			36	34	21	39	15	na	35
minimum Loops			85592	102662	52076	112196	54476	na	125558
maximum Loops			89453	114059	63095	123668	63858	na	135108
Cache									
Cachegröße			20449	24964	24964	21609	26244	na	26244
ungültige Kombina−tionen (LINKS)			5530	20355	18161	5855	18015	na	20642
gültige + gesehene Kombinationen (LINKS)			1598	918	2235	1654	2182	na	2636

Tabelle 5−27 Statistische Daten aller Optimierungsstufen. Für "fast03" *sind nur noch die Daten der Zuwei−sung vorhanden. Für die manuellen Zuweisungen können die statistischen Parameter des genetischen Algorithmus nicht ermittelt werden. "Loops" ist die Zyklenzahl in RANDOM.*

5.3.3.1 Konstante Bereiche

Alle Zuweisungen stimmen in vier konstanten Bereichen überein. Der längste Bereich erstreckt sich vom Ende des unsicheren Anfangsbereich AS 25 bis zu den manuell nicht zugewiesenen AS 50 und 51. Dahinter folgt ein kleiner Bereich von AS 53 bis 59. Der dritte Bereich erstreckt sich von AS 68 bis AS 87. Der Unterschied an Position 87 beruht auf einer zusätzlich eingefügten Peakgruppe (CB bei 45 ppm). Wenn man diesen Unterschied vernachlässigt, reicht der übereinstimmende Bereich sogar bis Tyr 93, d.h. über

das Prolin 90 hinaus. Danach folgt ein Bereich von 13 Aminosäuren in dem 11 Aminosäuren unterschiedlich zugewiesen wurden.

Der C−Terminus wird ab Val 107 wieder gleich zugewiesen und ist in allen Zuweisungen konstant.

5.3.3.2 Unterschiede

Die Endzustände fast01 bis cont04 unterscheiden sich auffälligerweise in denselben Bereichen. Der His−Tag AS 1−11 bleibt in allen, bis auf die Zustände der dritten Stufe, frei. Selbst in der dritten Stufe werden nur je eine der AS 9 bis 10 belegt.

Der anschließende Bereich bis einschließlich AS24 variiert stark. Von Stufe 1 zu Stufe 3 enthält dieser Bereich zunehmend fest zugewiesene Spinsysteme, die mit dem manuellen Assignment übereinstimmen.

Der zweite Bereich mit Veränderungen betrifft die Umgebung des Prolins 102 (AS 94 bis 103).

Dazwischen liegen drei kleine Stellen aus einzelnen Zuweisungen, die sich von der manuellen Zuweisung unterscheiden. Der Bereich Ala 50 + Ser 51 ist manuell nicht zugewiesen. Da alle automatischen Zuweisungen, bis auf fast01, diesen Bereich belegen und mit den benachbarten AS verknüpft sind, verschiebt das alle anderen Zuweisungen.

In Stufe 2 wurden die Spinsysteme des BP 201 in den AS−Cache eingeführt. Diese Verbesserung führt zu einer Kaskade von Veränderungen, die 12 weitere Spinsysteme verschiebt. Diese Veränderungen von Stufe 1 zu Stufe 2 sind in Tabelle 5−28 dokumentiert.

Die manuell zugewiesenen Spinsysteme waren in vier Fällen nicht in einer der automatischen Lösungen vertreten. Glu 106 wird konstant mit BP33 erklärt und ist gut mit den Nachbarn verknüpft. Thr 60 hat im manuellen Assignment den Basispeak von Glu 106. Da dieser konstant Glu 106 zugewiesen wird, kann Thr 60 nicht zugewiesen werden. Lys 19 und Ser 20 liegen im unsicheren Anfangsbereich. Ser 20 ist manuell nicht zugewiesen. Lys 19 ist manuell mit dem gespaltenen Spinsystem 101B belegt. Diese Aufspaltung wurde im automatischen Assignment nicht nachgebildet.

AS	AS Nr.	BP neu in Stufe 2	BP ehem. in Stufe 1	verschoben zum AS Nr.
Lys	17	201	46	103
Val	103	46	3	61
Ile	61	3	101	107
Val	107	101	15	23
Val	23	15	112	95
Val	95	112	55	14
Lys	14	55	130	67
Ile	67	130	111	20
Ser	20	111	123	49
Leu	49	123	12	109
Leu	109	12	16	24
Asp	24	16	23	15
Leu	15	23	–	–

Tabelle 5−28 Veränderung des Endzustandes durch das neue Spinsystem BP201 in der Stufe 2 **cont03**.

AS	i	Link zu i+1?	hnco CO−1 CONT02	Link zu i+1?	hnco CO−1 Manuell	Cont02 BP in manuellem Assignment	Manueller BP in cont02	Kommentar
Ser	12	Y	200			−−U−−	−−	
Glu	13	Y	31		30	Phe 66	−−U−−	
Lys	14		130			Ile 67	−−	
Leu	15							
Ala	16	Y	34	Y	103	−−U−−	Glu 99	
Lys	17	Y	46	Y	201	Val 103	−−U−−	aber p201 in Stufe 2 ff.
Thr	18	Y	36	–	120	Thr 104	Thr 104	−− Paar
Lys	19	Y	125		101	−−U−−	Ile 61	
Ser	20	–	123			−−U−−	−−	
Ala	50	Y	126			Glu 99	−−	
Ser	51	Y	129			Ser 100	−−	
Leu	59	–	9	Y	9	Leu 59	Leu 59	verändertes Spinsystem, da Peaks getauscht + anders gelinkt.
Thr	60	Y	30	–	33	Glu 13	Glu 106	
Ile	61	Y	101	–	3	Lys 19	Val 103	
Phe	66	Y	47	Y	31	Lys 101	Glu 13	
Ile	67	Y	111	–	130	−−U−−	Lys 14	
Gln	99	–	103	Y	126	Ala 16	Ala 50	
Ser	100	–	91	Y	129	−−U−−	Ser 51	
Lys	101			–	47	−−	Phe 66	
Pro	102							
Val	103	–	3	Y	46	Ile 61	Lys 17	
Thr	104	Y	120	Y	36	Thr 18	Thr 18	−− Paar
Phe	105	Y	35		35	−−I−−	−−I−−	−−− identisch

Tabelle 5−29 Bereiche mit unterschiedlicher Zuweisung im korrigierten manuellen und dem ***cont02*** *Endzu−stand.*

Abbildung 5−2 (auf den drei folgenden Seiten)

Gegenüberstellung der HN / N−Projektionen mit allen Peaks und nur mit den nicht genutzten Peaks in Trigger für cont01. Links sind die Peaks aus allen Experimenten in der HN/N. Rechts sind nur die Peaks dargestellt, die nicht im Assignment verwendet wurden.

Die Herkunft der Peaks aus den Peaklisten ist in der Peakform kodiert.

Ovale	*HNCO,*
Dreiecke	*HNCA,*
Rechtecke	*CBCANH,*
Rhomben	*CBCA(CO)NH.*

Die HNCO Peak Identifikationsnummern sind blau.

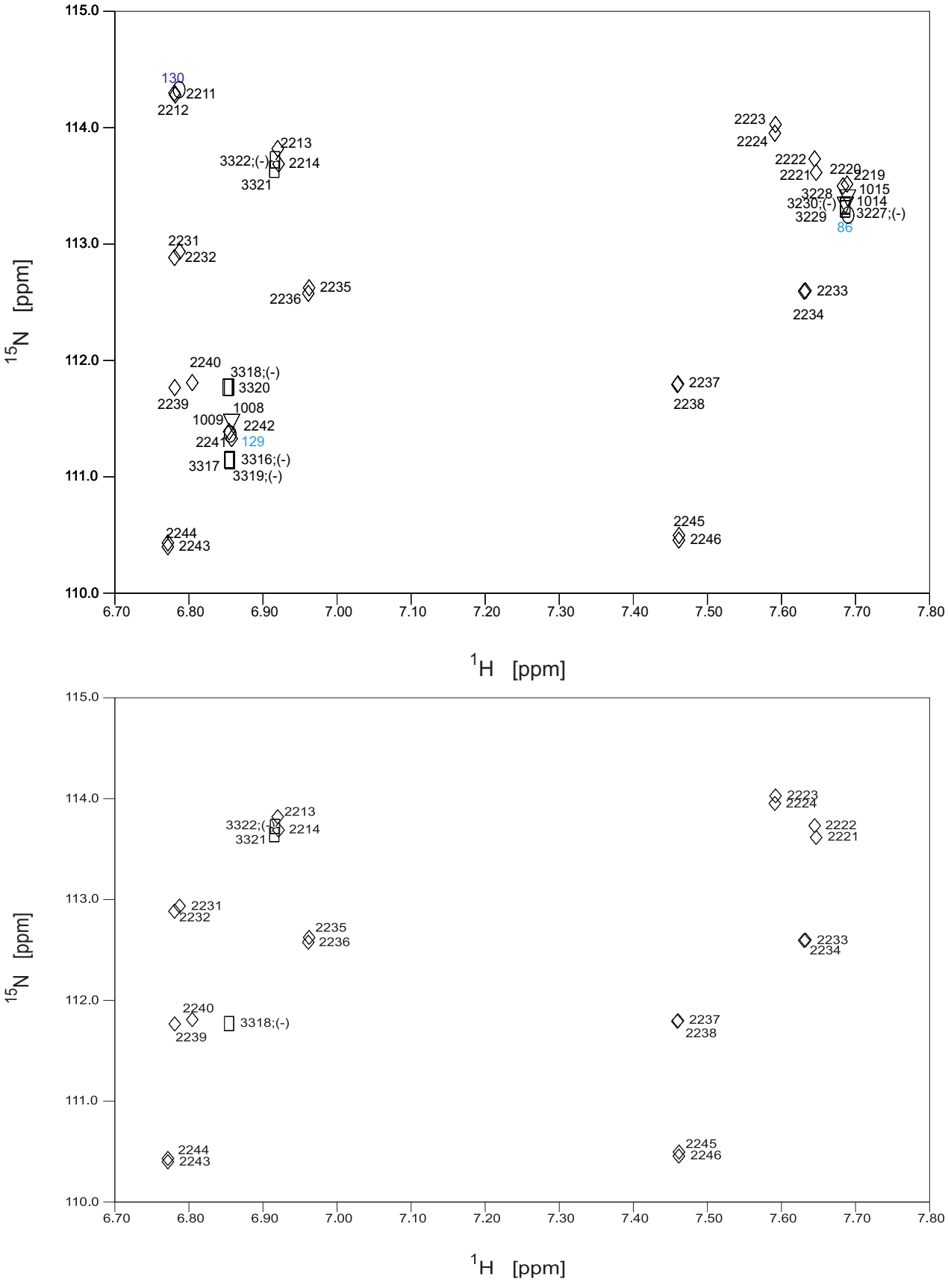


Abbildung 5−2 a) Der Bereich von ^{15}N [110.0 ppm − 115.0 ppm] und NH [6.7 ppm − 7.8 ppm].

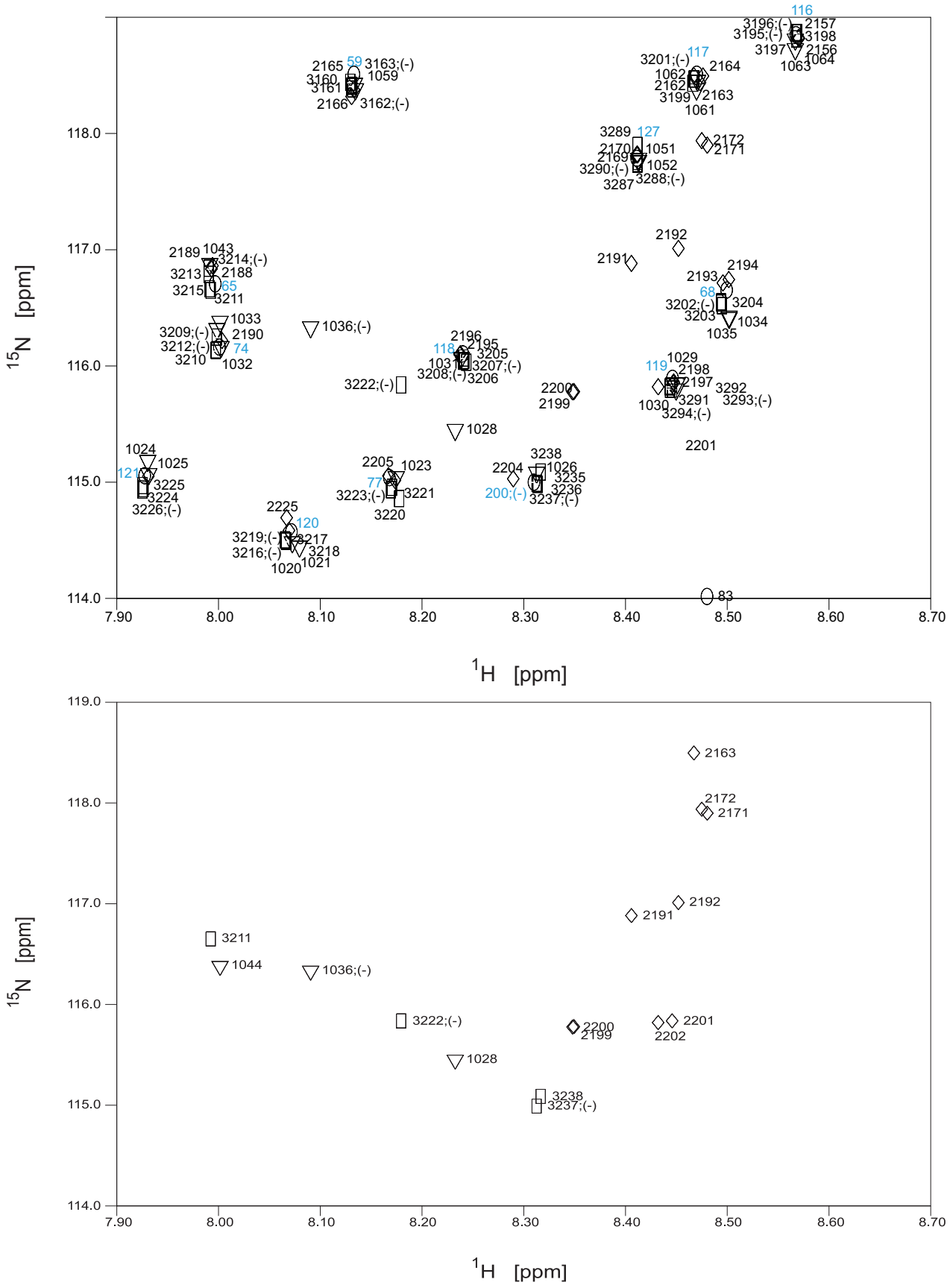


Abbildung 5−2 b) Der Bereich von ^{15}N [114.0 ppm − 119.0 ppm] und NH [7.9 ppm − 8.7 ppm].

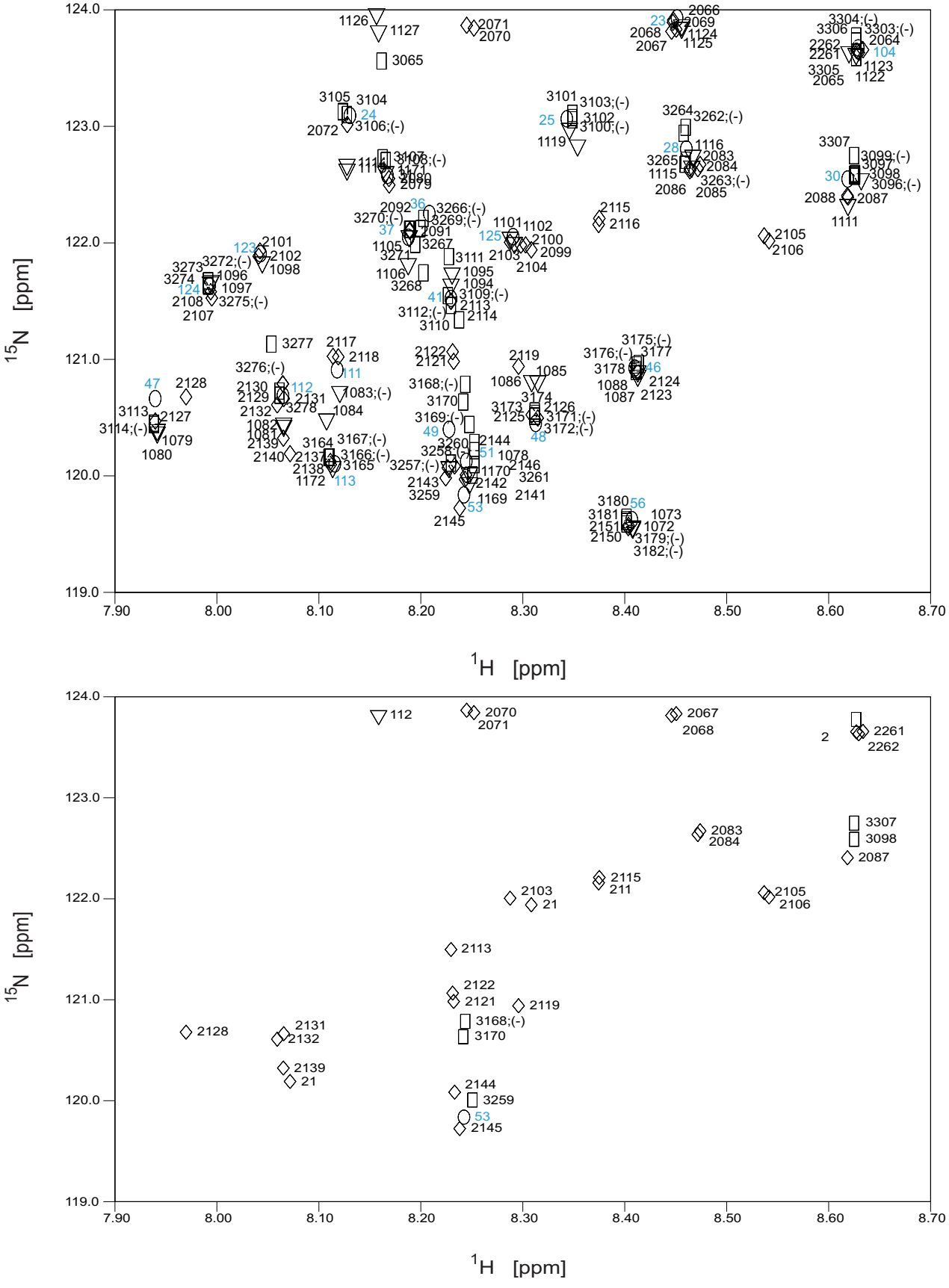


Abbildung 5−2 c) Der Bereich von ^{15}N [119.0 ppm − 124.0 ppm] und NH [7.9 ppm − 8.7 ppm].

5.3.3.3 Ungenutzte Peaks

Alle Zuweisungen nutzen nur einen Teil der gepickten Peaks. Die ungenutzten Peaks wurden für **cont02** in der H/N Ebene dargestellt und die Plots auf ungenutzte Spinsysteme untersucht. Für das manuelle Assignment können diese Plots prinzipbedingt nicht erstellt werden. Um dennoch Vergleiche zwischen manuellem und automatischem Assignment ziehen zu können, wurde eine spezielle Konfiguration des AS−Caches erstellt, mit der man das manuelle Assignment nachvollziehen kann.

Dazu wurden jeder Aminosäure alle Protospinsysteme des Basispeaks zugewiesen, der im manuellen Assignment zugewiesen war. Wenn für einen Basispeak mehrere Protospinsysteme möglich sind, wurden alle zugewiesen. Diese Konfiguration wurde dann von RANDOM mit dem Profil "exchangeOnly" optimiert. Dabei werden die Protospinsysteme auf ihre Nachbarn abgestimmt und die optimale Nachbarschaftskombination ausgewählt. Die entstehende Lösung sollte dem manuellen Assignmentergebnis sehr nahe kommen. Möglicherweise unterscheiden sie sich in einigen n−BP Peaks. Da während des manuellen Assignments keine Dokumentation für die Verwendung der einzelnen Peaks erzeugt wird, kann der eventuell vorhandene Unterschied nicht festgestellt werden.

Konfiguration	Summe	BP	n−BP	n−BP ohne BP	HNCA	CBCANH	CBCA(CO)NH
Manuell Vor	169	14	165	78	17	39	97
Manuell Nach	146	10	136	74	14	30	91
fast01	123	6	122	70	6	28	87
cont01	119	3	116	67	6	28	81
cont02	119	3	116	67	6	28	81
fast02	128	5	118	69	6	24	87
cont03	114	2	112	66	6	24	81
fast03	118	4	114	68	7	21	85
cont04	111	2	109	66	5	21	81

Tabelle 5−31 Anzahl der Peaks, die in den Lösungen nicht benutzt wurden. BP = Basispeak. n−BP = nicht−Basispeak, "ohne BP" = n−BPs die nicht in der Nähe eines Basispeaks sind.

Bei der Analyse der Spinsysteme ohne Basispeak wurden 4 Spinsysteme gefunden und mit virtuellen Basispeaks verbunden. Bei der nachträglichen Untersuchung der ungenutzten Peaks wurden keine zusätzlichen Spinsysteme gefunden. Die verbleibenden Peakgruppen passen weder zu einem bestehenden Spinsystem, noch kann man aus ihnen ein akzeptables neues Spinsystem bilden. Auffällig ist die große Anzahl ungenutzter CBCA(CO)NH Peakpaare, die aber der höheren Empfindlichkeit des Spektrums entspricht. Tabelle 5−31 schlüsselt die ungenutzten Peaks nach Experiment und Basispaks auf. Die "cont xx" Rechnungen benutzen erkennbar mehr Peaks als die "fast xx" Rechnungen, da sie durch die fehlende AS−Typeninformation, die Protospinsysteme freier verteilen können.

Abbildung 5−2 a)− c) stellen die Verteilung der gepickten Peaks und der ungenutzten Peaks in ausgewählten Ausschnitten des Spektrums für "**cont01**" dar. Oben ist die Projektion aller Peaks auf die HN/N Ebene dargestellt, darunter der selbe Ausschnitt mit den ungenutzten Peaks. In Abbildung 5−2 c) ist das ungenutzte Protospinsystem BP53 zu erkennen.

5.3.4 Fazit

Das automatische Assignment erreicht die Qualität des manuellen Assignments. Ein nicht zu unterschätzender Vorteil des automatischen Assignments ist die automatisch entstehende Dokumentation über die Verwendung der Peaks.

Die Differenzen zwischen automatischem und manuellem Assignment spiegeln die Unsicherheiten im Datenmaterial wieder. Bereiche, bei denen das automatische Assignment mehrere Lösungen findet, sind auch im manuellen Assignment unsichere Bereiche.

Kapitel 6 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein neuer Algorithmus für die automatische Optimierung einer NMR−Zuweisung des *back bones* in gelabelten Proteinen und seine Implementierung vorgestellt.

Der Algorithmus ermöglicht eine Zuweisungsstrategie, die sich näher an der manuellen Vorgehensweise orientiert als vergleichbare Implementierungen von Lukin (LUK[11]) [Lukin97] oder Leutner (PASTA) [Leutner98], da er die Entscheidung über konkurrierende Protospinsysteminterpretationen in die globale Optimierung der Zuweisung verlegt und die Benutzer nicht zwingt, sich vorzeitig auf eine Interpretation pro Aminosäure und Peak zu beschränken. Der Algorithmus löst daher nicht nur das Reihenfolgenproblem für einen schon vorgegebenen Satz von Protospinsystemen, sondern auch das Auswahlproblem zwischen verschiedenen, sich widersprechenden, Interpretationen der gleichen Peakgruppe.

Durch das zusätzliche Auswahlproblem erhöht sich die Komplexität der Optimierungsaufgabe, der Lösungsraum wächst daher um mehrere Größenordnungen. Dies verlängert die Laufzeit für den einfachen RANDOM Algorithmus gegenüber einem vergleichbaren Reihenfolgeproblem. Es wurden daher verschiedene Methoden untersucht, um die Laufzeit wieder zu reduzieren. Die erzielten Verbesserungen führen nicht nur zu einer besseren Konvergenz, sondern ermöglichen auch erstmals eine echte parallele Implementierung des Algorithmus.

Die algorithmischen Verbesserungen sind die Reduktion des Protospinsystemcaches und der GENETISCHe Algorithmus. Zusätzlich wurde eine verbesserte Konfiguration zur Kontrolle der Suboptimierer gefunden.

Die Implementierung der zugrundeliegenden Datenstrukturen ermöglicht die Umsetzung zusätzlicher Nebenbedingungen für die Positionen der Protospinsysteme innerhalb der Zuweisung. Ein Beispiel für die Nebenbedingungen sind die knotenlokalen Filter und die Strukturfilter aus Kapitel 2. Ihre Effizienz wird am besten durch die Aminosäuretypenerkennung demonstriert, die die Größe des Lösungsraumes um 168 Größenordnungen für Trigger reduziert. Dadurch verringert sich die Laufzeit des RANDOM Algorithmus bis zu einem Faktor von 132. Für PASTA wurde in [Leutner98] statt dessen der umgekehrte Effekt auf die Laufzeit festgestellt. In ihrer Implementierung verlängert sich die Laufzeit auf das Doppelte. Der Unterschied kann nur durch eine unterschiedliche Nutzung der Aminosäuretypenerkennung im Protospinsystempool erklärt werden. PASTA nutzt diese Information anscheinend nicht, um den Pool und damit den Lösungsraum zu reduzieren, sondern nur bei der Bewertung der Zuweisung. Diese Konfiguration kann in RANDOM näherungsweise nachvollzogen werden, wenn man im AS−Cache jeder Aminosäure jedes Protospinsystem

11 Lukin et al. geben ihrem Programm selbst keinen Namen. Die Bezeichnung LUK ist daher rein willkürlich gewählt.

zuweist. Die Bewertung der Aminosäuretypen und den Austausch zwischen Pool und Individuum kann man aber nicht abschalten.

Ein möglicher Nachteil der Reduktion des AS−Caches mittels der Aminosäuretypenerkennung ist die Möglichkeit, daß Aminosäuren mit untypischen Frequenzen der falschen Aminosäure zugewiesen werden oder nicht in den Cache für die theoretische Aminosäure aufgenommen werden. Um eine falsche Zuweisung durch die Reduktion auszuschließen oder zu überprüfen, kann man zuerst mit einem AS−Cache mit Typenerkennung optimieren und das Ergebnis in einer zweiten Optimierung mit einem AS−Cache ohne Typenerkennung nachoptimieren. Diese Möglichkeit wurde bei der Zuweisung von Trigger demonstriert.

Für Trigger konnte durch den Wechsel des Caches gezeigt werden, daß auch mit dem reduzierten Cache eine stabile Zuweisung gefunden wird, die der Zuweisung ohne Aminosäuretypenerkennung weitgehend entspricht. Die Zuweisungen unterscheiden sich in Aminosäuren mit unsicherer Typenzuweisung. In Trigger wird nur ein Protospinsystem (Phe66,Ile67) aufgrund der Typeninformation von der Position im AS−Cache ausgeschlossen, der es in der manuellen und der Zuweisung ohne Reduktion zugewiesen wurde. In anderen Fällen führt erst die Typeninformation zur richtigen Zuweisung (zB. Thr60,Ile61), während die Zuweisung ohne Typenerkennung eine Fehlzuweisung erzeugt. Man sollte daher immer beide Versionen berechnen und dann vergleichen, da eine manuelle Zuweisung beide Defekte aufweisen kann, weil sie die Zuweisungsregeln nicht so strikt optimiert, wie es ein automatischer Algorithmus kann.

Für beide Algorithmen wurde der Parameterraum untersucht und eine sowohl optimale als auch robuste Konfiguration gesucht. Dabei wurde der Einfluß jedes Parameters des Algorithmus auf die Laufzeit bis zur Konvergenz zum globalen theoretischen Maximum bestimmt.

Bei RANDOM waren hauptsächlich die Zusammensetzung des Aktionsprofils und die minimale und maximale Lebenszeit der veränderten Zustände, minTTL und maxTTL, für die Laufzeit entscheidend. Alle drei kontrollieren, welche Pfade durch den Lösungsraum ab einem Zustand erlaubt sind.

Die gleiche Untersuchung wurde auch für die Parameter der zu berechnenden Probleme (Proteine) durchgeführt. Für den Datensatz wurde sowohl der Einfluß der Proteingröße, des maximalen CA Abstandes zwischen den Peaks der benachbarten Aminosäuren und der Aufbereitung des Protospinsystemcaches auf die Laufzeit, als auch der Einfluß eines unvollständigen Datensatzes auf die Konvergenz untersucht. Der Algorithmus zeigte sich dabei sehr robust gegen fehlende Peaks, da bis zu 45% der theoretischen Peaks fehlen können, ohne daß die Konvergenz zum theoretischen Optimum verloren geht. Unterhalb von 55% hängt die Übereinstimmung der gefundenen Lösung mit der theoretischen Lösung von der Zusammensetzung der gebildeten Protospinsystemfragmente ab. Proteingröße und maximaler CA Abstand haben dagegen großen Einfluß auf die Laufzeit.

Während die Parametrisierung des RANDOM Algorithmus sich auf die Untersuchung des Wertebereichs der Kontrollparameter beschränken kann, muß die Parametrisierung des GENETISCHen Algorithmus auch die verschiedenen Crossover Operatoren und die Kontrolle der Suboptimierer umfassen.

In Kapitel 3 wurden die verschiedene Crossover Algorithmen und unterschiedliche Konfigurationsvarianten für den Suboptimierer im GENETISCHen Algorithmus untersucht. Die aus der Literatur bekannten Crossover Operatoren waren dabei den spezialisierten G Operatoren unterlegen. Auch die Varianten *elitär*

und *statistisch*, die keinen genetischen Austausch zwischen verschiedenen Individuen zulassen, waren den speziellen Operatoren immer unterlegen. Selbst kleine Anteile eines Genaustausches bewirkten eine Verbesserung der Laufzeit und die Qualität der Lösung. Dabei verbesserte sich besonders die Konvergenz durch den genetischen Austausch.

Die speziellen G Operatoren übertragen nur die Differenz der Eltern statt, wie im klassischen Crossover, blind irgendeinen zusammenhängenden Bereich zu kopieren. Dadurch vermeiden sie es, hauptsächlich identische Bereiche zu kopieren, sondern übertragen nur Protospinsysteme, die sich in den Eltern unterscheiden.

Außerdem wurde die Effizienz der speziellen Operatoren noch weiter gesteigert, indem die übertragenen Informationen mit Hilfe von Filtern gezielt auswählt wurden. Die übertragene Differenz wird dazu aufgrund ihrer Nachbarn und der Verknüpfung mit den Nachbarn ausgewählt. Mit unterschiedlichen Filtern wurden so Bereiche mit einem unterschiedlich hohen lokalen Optimierungsgrad für die Übertragung ausgesucht.

Die Laufzeituntersuchungen in Kapitel 3 zeigen, daß es eine Grenze für die Effizienzsteigerung durch die Übertragung lokal voroptimierter Bereiche gibt. Während es beim Wechsel von unkoordinierten, punktförmigen Übertragungen (s1) zur Übertragung von kurzen Strecken aus benachbarten Protospinsystemen, mit oder ohne Verknüpfung (s2), noch zu einer geringen Verbesserung der Laufzeit und der Konvergenz kommt, führt die zusätzliche Koordinierung der übertragenen Nachbarn durch die Nebenbedingung der Verknüpfung in den s3 Algorithmen sogar zu einer geringen Verschlechterung der Laufzeit. Die Gründe für die Verschlechterung konnten nicht eindeutig nachgewiesen werden.

Möglicherweise verringert sich die Anzahl der kopierbaren Bereiche durch die zusätzlichen Nebenbedingungen so sehr, daß nicht genügend unterschiedliche Differenzen für die Übertragung in die neuen Individuen gebildet werden. Die neuen Individuen erhalten dann hauptsächlich die gleichen neuen Gene. Dadurch kann es zu einer Verarmung des genetischen Pools, d.h. zum Verlust der genetischen Diversität in der Population kommen. Die neuen Individuen suchen dann identische Bereiche im Lösungsraum ab und nehmen dadurch die lokale Optimierung doppelt vor. Dadurch verschwendet diese Konfiguration CPU Zeit auf schon untersuchte Bereiche.

Neben der direkten Übertragung zusammenhängender Bereiche gibt es noch einen weiteren Faktor der die spezialisierten Operatoren, gegenüber dem klassischen Crossover, verbessert. Die Differenzbildung im Crossover nimmt auf die Konkurrenz um die Peaks zwischen den Protospinsystemen keine Rücksicht. Dadurch befinden sich die neuen Individuen zuerst meist im illegalen Teil des Zustandsraumes $\boldsymbol{S}^n$. Nach dem Crossover wird daher ein Teil der ursprünglichen Protospinsysteme des neuen Individuums durch den Reparaturmechanismus gelöscht und dadurch schon optimierte Bereiche zerstört. Daher kommt dem Reparaturmechanismus für illegale Zustände die entscheidende Rolle zu, denn er erzwingt die Neuberechnung der Positionen, die vor dem Crossover durch einen Konkurrenten eines übertragenen Protospinsystems belegt waren. Die ehemaligen Zuweisungen in den zerstörten Bereichen sind dabei nur über die Resourcenkonkurrenz mit den im Crossover neu injizierten korreliert. Das Crossover mit Zerstörungen erweitert dadurch den erreichbaren Lösungsraum, statt ihn, wie beim klassischen Crossover, auf den Raum zwischen den Eltern einzuschränken! Die Zerstörungen sind für die Optimierung daher von Vorteil.

Außerdem zeigte sich in Kapitel 3, daß für die speziellen Operatoren eine besondere Konfiguration des Suboptimierers besonders effizient ist, die den einzelnen Individuen nur die CPU−Zeit zuteilt, die sie, effizienter als ihre Konkurrenten, zu einer Verbesserung nutzen können. Diese Verteilung der Rechenzeit wird durch die sogenannte Ratenbegrenzung erreicht. Die veränderlichsten Individuen erhalten dabei die meiste Rechenzeit. Dies sind normalerweise die Individuen, die in einem der letzten genetischen Zyklen neu erzeugt wurden. Sobald ein Individuum in ein tiefes lokales Optimum, d.h. eine Sackgasse, geraten ist, erhält es weniger Rechenzeit, da seine Verbesserungsrate unter den Schwellwert sinkt. Dadurch darf man den einzelnen Individuen eine höhere maximale Laufzeit zuteilen, da sie sie nur nutzen können, wenn sie sich währenddessen verbessern.

Die Kombination "ratenbegrenzt" mit Operator Gs2p2A verbraucht trotz des komplexeren Algorithmus weniger CPU−Zeit bei gesteigerter Konvergenzwahrscheinlichkeit und geringerer paralleler Laufzeit als jede andere Konfiguration. Sie ist damit die beste bekannte Konfiguration für GENETISCH.

Im Praxistest in Kapitel 5 mit dem Proteinabschnitt Trigger M bestanden zwischen den automatischen Zuordnungen und der manuellen Zuweisung nur geringe Unterschiede. Die Unterschiede zwischen den Optimierungen mit unterschiedlichen Caches entstehen hauptsächlich in den Bereichen, die an einem Ende keinen verknüpften Nachbarn haben oder bei denen die manuelle Zuweisung unsicher ist, wie im Bereich 10 − 20. Durch die automatische Zuweisung konnte die manuelle Zuweisung sogar an einigen Stellen vervollständigt werden. Sie ist daher einer manuellen Zuweisung gleichwertig. Darüber hinaus entsteht bei der automatischen Zuweisung immer eine Dokumentation über die Verwendung der Peaks, so daß man ungenutzte Spinsysteme leichter finden kann.

Der genetische Algorithmus kann optimal auf einem Netzwerkcluster implementiert werden, daher bietet sich ein echte parallele Implementierung an. GENETISCH braucht dabei keine besondere Hardware, wie Vektorrechner, *shared memory* oder besonders schnelle Netzwerkverbindungen, sondern kann auf einer Gruppe von normalen Rechnern mit einer üblichen Ethernet100 Netzwerkverbindung implementiert werden, da nur zum Austausch der Individuen, bzw. ihrer Differenz, eine Kommunikation zwischen den Rechnern notwendig ist. Dadurch kann die Kommunikation auf wenige KB alle paar Sekunden beschränkt bleiben. GENETISCH sollte außerdem leicht zu skalieren sein, da man die Anzahl der Individuen leicht an größere Probleme anpassen kann. Eine echte parallele Implementierung erlaubt daher sowohl die Rechenzeit für komplexere Problem auf wenige Minuten zu reduzieren als auch größere und mehrdeutigere Spektren zuzuweisen.

Außerdem kann man die Wahl der Prozessparameter selbst auch dem selben Optimierungsprozeß unterwerfen, der bisher für die Optimierung der Zuweisung verwendet wurde. Dadurch sollte es möglich sein, die Optimierungsparameter, analog dem Profil der Abkühlung im *simulated annealing*, dynamisch an die Phase der Optimierung anzupassen und den Individuen immer die optimalsten Optimierungsparameter zu bieten, ohne diese vorher von Hand suchen zu müssen. Diese Veränderung erfordert aber eine große Anzahl an Individuen und damit die echte parallele Implementierung.

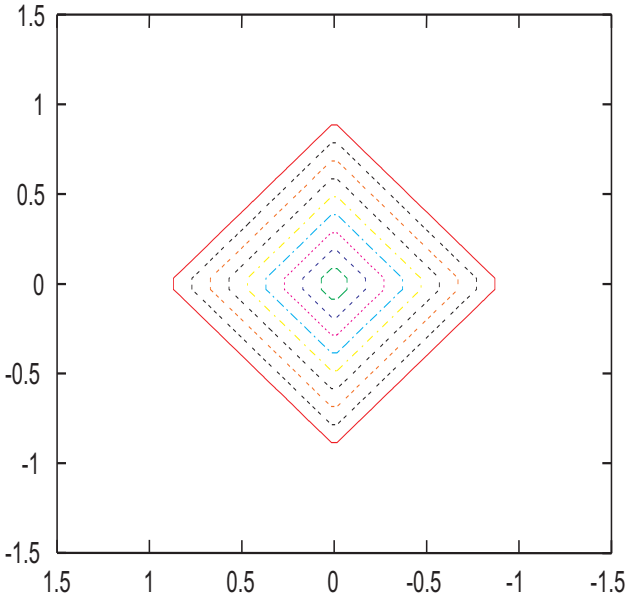
Anhang A Pseudozufallszahlengenerator

Der Pseudozufallszahlengenerator wurde der GNU LibG++ 2.8x entnommen und entspricht dem Additiven Zahlengenerator wie er von Knuth im Volume II von "The Art of Computer Programming" beschrieben wurde. Dieser Generator ist dem Standard UNIX Generator (`random()` und `rand()`) in der Qualität (Zufälligkeit) der generierten Zahlen überlegen. Der folgende Kommentar stammt vom Autor der Implementierung:

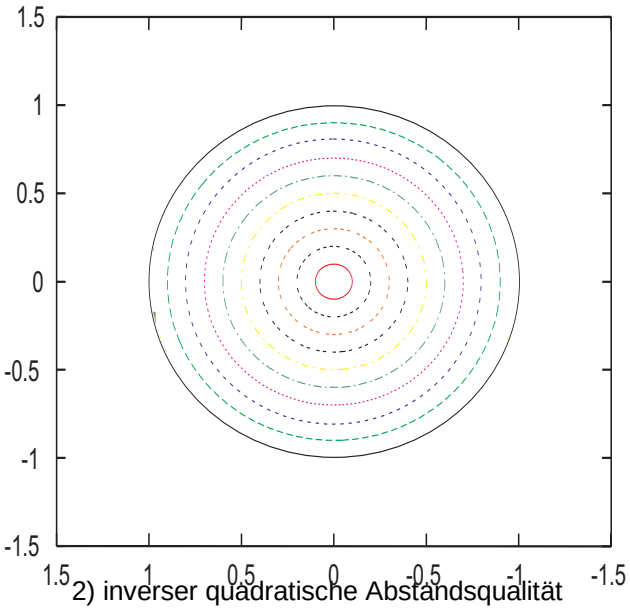
```
written by Dirk Grunwald (grunwald@cs.uiuc.edu)
This is an extension of the older implementation of Algorithm M which I previously supplied. The
main difference between this version and the old code are:

      + Andres searched high & low for good constants for the LCG.
      + there is more bit chopping going on.

The following contains his comments.

      agn@UNH.CS.CMU.EDU sez..

The generator below is based on 2 well known methods: Linear Congruential (LCGs) and Additive Con-
gruential generators (ACGs).

The LCG produces the longest possible sequence of 32 bit random numbers, each being unique in that
sequence (it has only 32 bits of state). It suffers from 2 problems:

a)
      Independence isn't great, that is the (n+1)th number is somewhat related to the preceding
      one, unlike flipping a coin where knowing the past outcomes dont help to predict the next
      result.
b)
      Taking parts of a LCG generated number can be quite non-random: for example, looking at only
      the least significant byte gives a permuted 8-bit counter (that has a period length of only
      256).  The advantage of an LCA is that it is perfectly uniform when run for the entire period
      length (and very uniform for smaller sequences too, if the parameters are chosen carefully).

      ACGs have extremly long period lengths and provide good independence.  Unfortunately,
      uniformity isnt not too great. Furthermore, I didn't find any theoretically analysis of ACGs
      that addresses uniformity.

      The RNG given below will return numbers generated by an LCA that are permuted under
      control of a ACG. 2 permutations take place: the 4 bytes of one LCG generated number are
      subjected to one of 16 permutations selected by4 bits of the ACG. The permutation a such that
      byte of the result may come from each byte of the LCG number. This effectively destroys the
      structure within a word. Finally, the sequence of such numbers is permuted within a range of
      256 numbers. This greatly improves independence.


Algorithm M as described in Knuths "Art of Computer Programming", Vol 2. 1969 chapter 3.2.2 p.32 is
used with a linear congruential generator (to get a good uniform distribution) that is permuted
with a Fibonacci additive congruential generator to get good independence.

      Bit, byte, and word distributions were extensively tested and pass Chi-squared test near
      perfect scores (>7E8 numbers tested, Uniformity assumption holds with probability > 0.999)

      Run-up tests for on 7E8 numbers confirm independence with probability > 0.97.

      Plotting random points in 2d reveals no apparent structure.

      Autocorrelation on sequences of 5E5 numbers (A(i) = SUM X(n)*X(n-i),i=1..512)results in no
      obvious structure (A(i) ~ const).

Except for speed and memory requirements, this generator outperforms random() for all tests. (ran-
dom() scored rather low on uniformity tests, while independence test differences were less dra-
matic).

AGN would like to..
       thanks to M.Mauldin, H.Walker, J.Saxe and M.Molloy for inspiration & help.

And I would (DGC) would like to thank Donald Kunth for AGN for letting me
use his extensions in this implementation.
```

Die Implementierung wurde auf die Gleichmäßigkeit der Zahlenverteilung getestet und zeigt keine Auffälligkeiten in der Verteilung der Zufallszahlen.

Anhang B Abstandsfunktionen



1) inverser lineare Abstandsqualität



2) inverser quadratische Abstandsqualität

Die Abstandsfunktionen müssen Qualitätswerte im Bereich der Fuzzy Logik Wahrheitswerte zurückgeben. Außerdem muß das Maximum der Qualitätsfunktion bei minimalem Abstand liegen.

Daher werden die Distanzfunktionen f() invertiert und auf 1 normiert.

1. inverser lineare Abstandsqualität

$$\begin{aligned} f(x_1, x_2, x_{maxDist}) &= \frac{x_{maxDist} - |x_1 - x_2|}{x_{maxDist}} \\ F1(x_1, y_1, x_2, y_2) &= \max(f(x_1, y_1, x_2, y_2), 0.0) \\ F(x_1, x_2, y_1, y_2) &= \min(F1(x_1, x_2, x_{maxDist}), F1(y_1, y_2, y_{maxDist})) \end{aligned} \qquad (B\text{-}1)$$

2. inverser quadratische Abstandsqualität

$$\begin{aligned} f(x_1, y_1, x_2, y_2) &= \sqrt{\frac{1}{2}\left(\frac{(x_1 - x_2)^2}{x_{maxDist}^2} + \frac{(y_1 - y_2)^2}{y_{maxDist}^2}\right)} \\ F1(x_1, y_1, x_2, y_2) &= 1 - \frac{f(x_1, y_1, x_2, y_2)}{sqrtMax} \\ F(x_1, y_1, x_2, y_2) &= \max(F1(x_1, y_1, x_2, y_2), 0.0) \end{aligned} \qquad (B\text{-}2)$$

$x_{MaxDist}$	:	maximaler Abstand für die erste Dimension
$y_{MaxDist}$	:	maximaler Abstand für die zweite Dimension
maxDist	:	das Maximum der f() Funktion

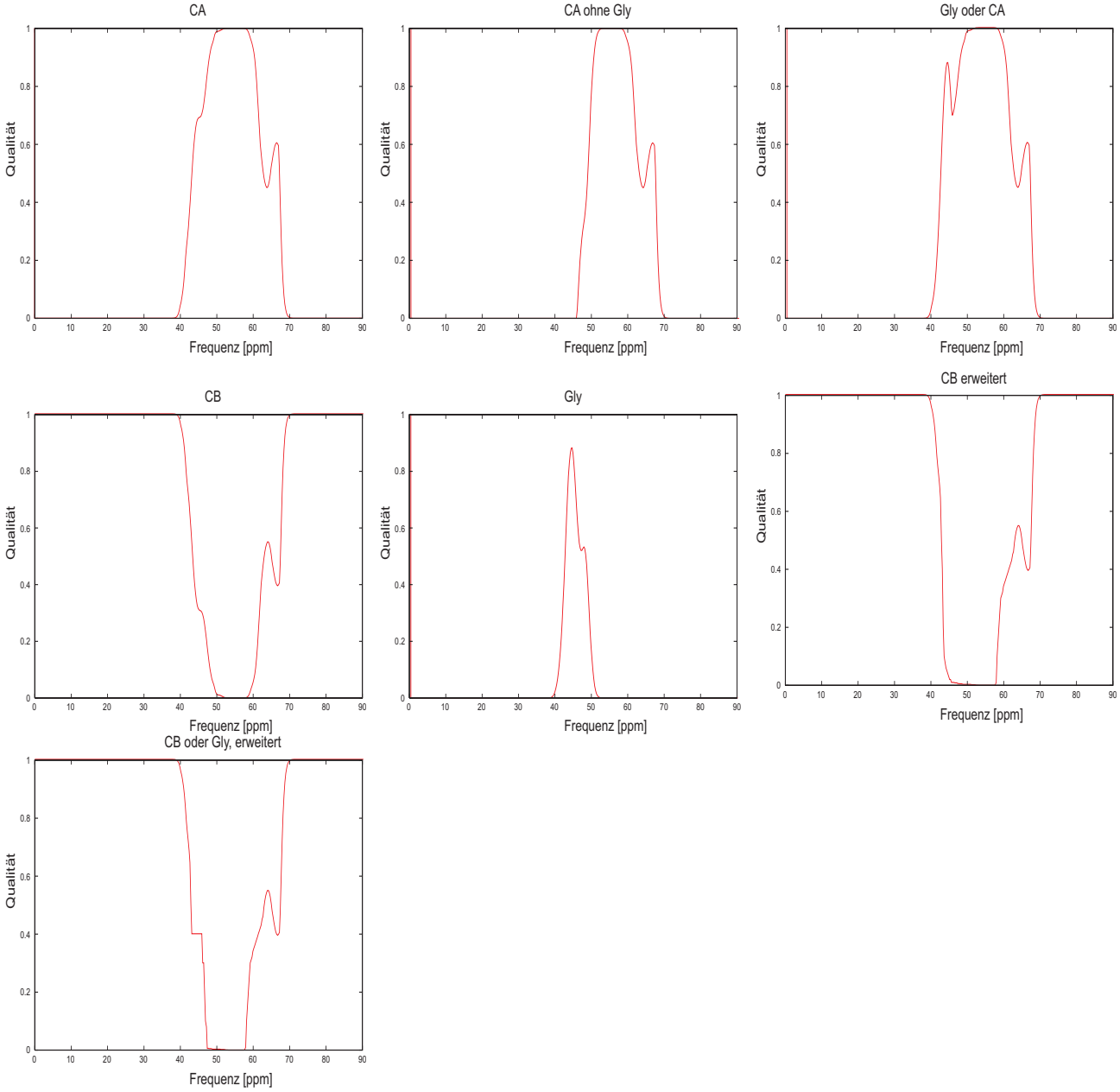
mit

$$\max(x_1, x_2) = \begin{cases} x_1 & x_1 > x_2 \\ x_2 & x_1 \le x_2 \end{cases} \quad \text{und} \quad \min(x_1, x_2) = \begin{cases} x_1 & x_1 < x_2 \\ x_2 & x_1 \ge x_2 \end{cases} \qquad (B\text{-}3)$$

Anhang C Aminosäuretypenerkennung

C.1 1D Funktionen für CA und CB

Die Aminosäuretypenerkennung benutzt die Funktionen aus [Grzesiek93] zur Erkennung der CA/CB−Paare. Die 1D Funktionen wurden aus der Grzesiek−Funktion gewonnen, in dem die Wahrscheinlichkeit des CA oder CB Wertes mit einem CB oder CA Wert von 0 ppm berechnet wurde. Danach wurden die Funktionen mit den Frequenzangaben aus [Groß88] und [Wishart91] überprüft und in den Bereichen 59 − 61 ppm und 40 − 45 ppm für CB angehoben. Die Funktionen berechnen die Wahrscheinlichkeit, daß eine Fre−quenz in diesem Bereich eine CA− oder CB−Frequenz ist.



C.2 2D Funktionen für CA/CB Frequenzpaare

Die 2D Funktionen wurden unverändert aus [Grzeiek93] übernommen. Die folgenden Abbildungen zeigt den Überblick über die gesamte Entscheidungsfunktion unter verschiedenen Vorbedingungen. Konturlevel jeweils bei [0.9,0.7,0.6,0.5,0.1,0.05,0.005,0.001, 0.0005, 0.0001, 0.00001], wenn nicht anders angegeben. Danach folgen die Wahrscheinlichkeitsfunktionen der einzelnen Aminosäuren.

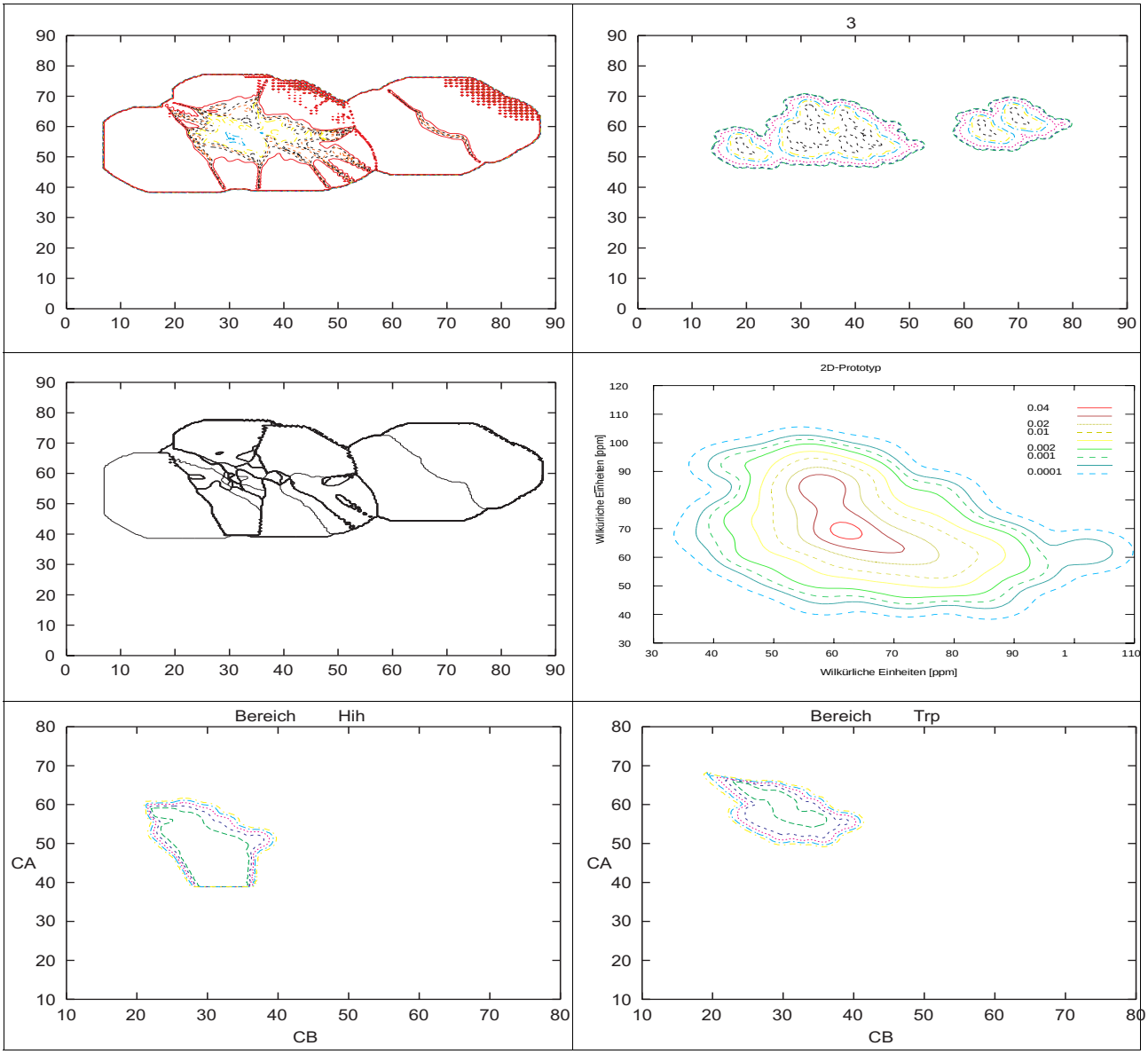
Oben links: Maximale global normierte Wahrscheinlichkeit über alle Aminosäuretypen.

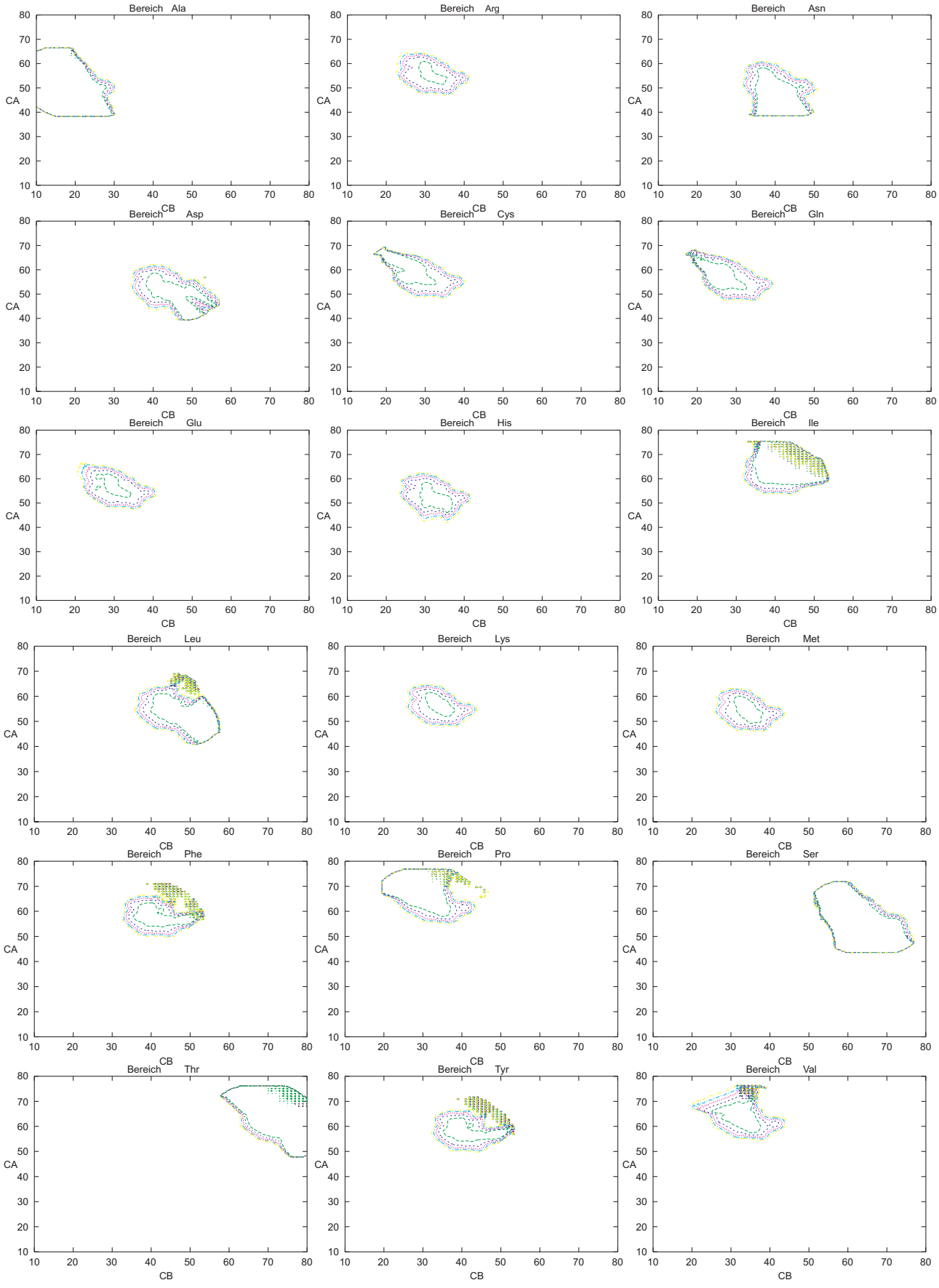
Oben rechts: Die Wahrscheinlichkeit ohne Normierung auf den globalen Maximalwert.

Mitte links: Die Entscheidungsfunktion, die aus der global normierten Wahrscheinlichkeit resultiert.

Mitte rechts: Der 2D−Prototyp für die AS Wahrscheinlichkeitsfunktionen aus [Grzeiek93].

Unten links und rechts und folgende Seite: Die Wahrscheinlichkeiten für die einzelne AS. Hih ist His proto− niert (His+) .




Bereich Ala
Bereich Arg
Bereich Asn
Bereich Asp
Bereich Cys
Bereich Gln
Bereich Glu
Bereich His
Bereich Ile
Bereich Leu
Bereich Lys
Bereich Met
Bereich Phe
Bereich Pro
Bereich Ser
Bereich Thr
Bereich Tyr
Bereich Val
CA
CB

Anhang D Protospinsystemmuster

D.1 Suchmuster für AS−Protospinsysteme

Protospinsystemmuster "completeAS.noSub.wideCB" für die Aminosäureerkennung von XX Nach−barpaaren.

```
// -*- c++ -*-
//   CA intra Verknüpfung, Volumen (HNCA), Vorzeichen CB(+-1) in CBCANH
//   für Gly (beidseitig),   mit Weitem CB
//
// MAKEPATTERN MIN:  $condCount - 10
#include "ASContexte.macros.new"

#define  CA_SCHWELLE     0.05
#define  CB_SCHWELLE     0.05
#define  TOL_PCT         0.05

#define  HNCO            "hnco CO-1 #1"
#define  HNCA_CA_m1      "hnca CA-1 #1"
#define  HNCA_CA         "hnca CA #1"

#define  CBCA_CONH_CA_m1  "cbca_conh CA-1 #1"
#define  CBCA_CONH_CB_m1  "cbca_conh CB-1 #1"

#define  CBCANH_CA_m1     "cbcanh CA-1 #1"
#define  CBCANH_CB_m1     "cbcanh CB-1 #1"
#define  CBCANH_CA        "cbcanh CA #1"
#define  CBCANH_CB        "cbcanh CB #1"
    PEAKLISTMEMBER(REQUIRED,HNCO,"hnco")
    LOCKPEAK(REQUIRED,HNCO)

//  alle CA-1 Nodes, dann mittelwert bilden und als CA bewerten
    find2PATH(OPTIONAL, HNCO,HNCA_CA_m1,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,AXIS_hnco_n,AXIS_hnca_n)
    find2PATH(OPTIONAL, HNCO,CBCA_CONH_CA_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,AXIS_hnco_n,AXIS_cbca_conh_n)
    find2PATH(OPTIONAL, HNCO,CBCANH_CA_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn, AXIS_hnco_n,AXIS_cbcanh_n)
    IFEXISTS1(REQUIRED,CBCANH_CA_m1,TESTVOLUME(OPTIONAL,CBCANH_CA_m1,1))
    IFEXISTS2(REQUIRED,CBCA_CONH_CA_m1,CBCANH_CA_m1,TESTNODESARELINKED(OPTIONAL,
                CBCA_CONH_CA_m1,CBCANH_CA_m1,AXIS_cbca_conh_ca,AXIS_cbcanh_ca))
    IFEXISTS2(REQUIRED,CBCA_CONH_CA_m1,HNCA_CA_m1,TESTNODESARELINKED(OPTIONAL,
                CBCA_CONH_CA_m1,HNCA_CA_m1,AXIS_cbca_conh_ca,AXIS_hnca_ca))
    IFEXISTS2(REQUIRED,CBCANH_CA_m1,HNCA_CA_m1,TESTNODESARELINKED(OPTIONAL, CBCANH_CA_m1,HNCA_CA_m1,AXIS_cbcanh_ca,AXIS_hnca_ca))
    AverageFreq3(REQUIRED,"CA-1 #1",AVRES_hnca_ca_m1,AVRES_cbca_conh_ca_m1,AVRES_cbcanh_ca_m1)
    IFEXISTS1(REQUIRED,"CA-1 #1",TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CA-1 #1",CA_SCHWELLE,
                "/simul/va/assign/Gr/linFunc.istEinCA.data"))

//  alle CA Nodes, dann mittelwert bilden und als CA bewerten
     find2PATH(OPTIONAL, HNCO,HNCA_CA,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,AXIS_hnco_n,AXIS_hnca_n)
     IFEXISTS2(REQUIRED,HNCA_CA_m1,HNCA_CA, TESTCOMPVOLUME(OPTIONAL,HNCA_CA,HNCA_CA_m1,TOL_PCT,0.5))
     find2PATH(OPTIONAL, HNCO,CBCANH_CA,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
     IFEXISTS1(REQUIRED,CBCANH_CA,TESTVOLUME(OPTIONAL,CBCANH_CA,1))
     IFEXISTS2(REQUIRED,CBCANH_CA,HNCA_CA,TESTNODESARELINKED(OPTIONAL, HNCA_CA,CBCANH_CA,AXIS_cbcanh_ca,AXIS_hnca_ca))
     AverageFreq2(REQUIRED,"CA #1",AVRES_hnca_ca_p1, AVRES_cbcanh_ca_p1)
     IFEXISTS1(REQUIRED,"CA #1",TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CA #1",CA_SCHWELLE,
                "/simul/va/assign/Gr/linFunc.istEinCA.data"))

// alle CB-1, siehe CA-1
    // if CA == GLY
    find2PATH(OPTIONAL, HNCO,CBCA_CONH_CB_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,AXIS_hnco_n, AXIS_cbca_conh_n)
    find2PATH(OPTIONAL, HNCO,CBCANH_CB_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
    IFEXISTS1(REQUIRED,CBCANH_CB_m1,TESTVOLUME(OPTIONAL,CBCANH_CB_m1,-1))
    IFEXISTS2(REQUIRED,CBCA_CONH_CB_m1,CBCANH_CB_m1,TESTNODESARELINKED(OPTIONAL,
                CBCA_CONH_CB_m1,CBCANH_CB_m1,AXIS_cbca_conh_ca,AXIS_cbcanh_ca))
    AverageFreq2(REQUIRED,"CB-1 #1", AVRES_cbca_conh_cb_m1,AVRES_cbcanh_cb_m1)
    IFEXISTS1(REQUIRED,"CB-1 #1",TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CB-1 #1",CB_SCHWELLE, "/simul/va/assign/Gr/linFunc.ist-
                EinCBOrGly.manuell.data"))

// // alle CB
   // if CA-1 == GLY
     find2PATH(OPTIONAL, HNCO,CBCANH_CB,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
     IFEXISTS1(REQUIRED,CBCANH_CB,TESTVOLUME(OPTIONAL,CBCANH_CB,-1))
     IFEXISTS1(REQUIRED,CBCANH_CB,AverageFreq1(OPTIONAL,"CB #1", AVRES_cbcanh_cb_p1))
     IFEXISTS1(REQUIRED,"CB #1", TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CB #1",CB_SCHWELLE, "/simul/va/assign/Gr/linFunc.ist-
                EinCBOrGly.manuell.data"))

    IFANY4(REQUIRED,2,"CA-1 #1", "CB-1 #1", "CA #1", "CB #1",CONST_TRUE(REQUIRED))
// test auf moegliche AS
    IFANY2(OPTIONAL,1,"CA-1 #1", "CB-1 #1",AminoTest(REQUIRED,"GRIE #1","CA-1 #1", "CB-1 #1"))
    IFANY2(OPTIONAL,1,"CA #1", "CB #1",AminoTest(REQUIRED,"GRIE #2","CA #1", "CB #1"))
// bestimme den Typ
    IFANY2(OPTIONAL,1,"CA-1 #1", "CB-1 #1",GrieMap(REQUIRED,"GRIE #1","CA-1 #1", "CB-1 #1"))
    IFANY2(OPTIONAL,1,"CA #1", "CB #1",GrieMap(REQUIRED,"GRIE #2","CA #1", "CB #1"))
```

Text D−1 completeAS.noSub.wideCB

```
// -*- c++ -*-
//
// sucht nach einem Gly-X Muster.
//
// berücksichtigt:
//   CA intra Verknüpfung,Volumen (HNCA), Vorzeichen CB(+-1) in CBCANH
//   für Gly-X
//
// noch nicht:  vorbelegen mit Resourcen   (zB aus lokalen Bedingungen) IFNOTEXISTS( ..., ...)
// noch nicht:  strategie varianten je nach lokalen Bedingungen (lokalen mini Contexten)
// MAKEPATTERN MIN:  $condCount - 10
//   #34
#include "ASContexte.macros.new"

#define  CA_SCHWELLE     0.05
#define  GLY_SCHWELLE    0.1
#define  CB_SCHWELLE     0.05

#define  TOL_PCT         0.05

#define  HNCO            "hnco CO-1 #1"
#define  HNCA_CA_m1      "hnca CA-1 #1"
#define  HNCA_CA         "hnca CA #1"

#define  CBCA_CONH_CA_m1  "cbca_conh CA-1 #1"
//#define  CBCA_CONH_CB_m1  "cbca_conh CB-1 #1"

#define  CBCANH_CA_m1     "cbcanh CA-1 #1"
//#define  CBCANH_CB_m1     "cbcanh CB-1 #1"
#define  CBCANH_CA        "cbcanh CA #1"
#define  CBCANH_CB        "cbcanh CB #1"

    PEAKLISTMEMBER(REQUIRED,HNCO,"hnco")
    LOCKPEAK(REQUIRED,HNCO)

//  alle CA-1 Nodes, dann mittelwert bilden und als CA bewerten
    find2PATH(OPTIONAL, HNCO,HNCA_CA_m1,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,AXIS_hnco_n,AXIS_hnca_n)
    find2PATH(OPTIONAL, HNCO,CBCA_CONH_CA_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,AXIS_hnco_n,AXIS_cbca_conh_n)
    find2PATH(OPTIONAL, HNCO,CBCANH_CA_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
    IFEXISTS1(REQUIRED,CBCANH_CA_m1,TESTVOLUME(OPTIONAL,CBCANH_CA_m1,1))
    IFEXISTS2(REQUIRED,CBCA_CONH_CA_m1,CBCANH_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCA_CONH_CA_m1,CBCANH_CA_m1,
              AXIS_cbca_conh_ca,AXIS_cbcanh_ca))
    IFEXISTS2(REQUIRED,CBCA_CONH_CA_m1,HNCA_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCA_CONH_CA_m1,HNCA_CA_m1,
              AXIS_cbca_conh_ca,AXIS_hnca_ca))
    IFEXISTS2(REQUIRED,CBCANH_CA_m1,HNCA_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCANH_CA_m1,HNCA_CA_m1,AXIS_cbcanh_ca,AXIS_hnca_ca))
    AverageFreq3(REQUIRED,"CA-1 #1",AVRES_hnca_ca_m1,AVRES_cbca_conh_ca_m1,AVRES_cbcanh_ca_m1)
    IFEXISTS1(REQUIRED,"CA-1 #1",TESTEXTERNFUZZYMAP1D_VALUE(REQUIRED,"CA-1 #1",GLY_SCHWELLE,
              "/simul/va/assign/Gr/linFunc.istEinGly.data"))

//  alle CA Nodes, dann mittelwert bilden und als CA bewerten
    find2PATH(OPTIONAL, HNCO,HNCA_CA,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,AXIS_hnco_n,AXIS_hnca_n)
//  kein volumenvergleich HNCA CA <-> CA-1 im suchmuster!
//    IFEXISTS2(REQUIRED,HNCA_CA_m1,HNCA_CA, TESTCOMPVOLUME(OPTIONAL,HNCA_CA,HNCA_CA_m1,TOL_PCT,0.5))
    find2PATH(OPTIONAL, HNCO,CBCANH_CA,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
    IFEXISTS1(REQUIRED,CBCANH_CA,TESTVOLUME(OPTIONAL,CBCANH_CA,1))
    IFEXISTS2(REQUIRED,CBCANH_CA,HNCA_CA,TESTNODESARELINKED(OPTIONAL,HNCA_CA,CBCANH_CA,AXIS_cbcanh_ca,AXIS_hnca_ca))
    AverageFreq2(REQUIRED,"CA #1",AVRES_hnca_ca_p1, AVRES_cbcanh_ca_p1)
    IFEXISTS1(REQUIRED,"CA #1",TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CA #1",CA_SCHWELLE,
              "/simul/va/assign/Gr/linFunc.istEinCA_ohneGly.data"))

// kein CB-1 ( Gly-X Pattern !!)

// #ifdef FIND_CB
// // alle CB
   // if CA-1 == GLY
     find2PATH(OPTIONAL, HNCO,CBCANH_CB,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
     IFEXISTS1(REQUIRED,CBCANH_CB,TESTVOLUME(OPTIONAL,CBCANH_CB,-1))
     IFEXISTS1(REQUIRED,CBCANH_CB,AverageFreq1(OPTIONAL,"CB #1", AVRES_cbcanh_cb_p1))
     IFEXISTS1(REQUIRED,"CB #1", TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CB
               #1",CB_SCHWELLE,"/simul/va/assign/Gr/linFunc.istEinCB.manuell.data"))
// #endif

    IFANY4(REQUIRED,2,"CA-1 #1", "CB #1", "CB #1", "CA #1",CONST_TRUE(REQUIRED))
// test auf moegliche AS     (CB-1 muss drin bleiben !!!)
    IFANY2(OPTIONAL,1,"CA-1 #1", "CB-1 #1",AminoTest(REQUIRED,"GRIE #1","CA-1 #1", "CB-1 #1"))
    IFANY2(OPTIONAL,1,"CA #1", "CB #1",AminoTest(REQUIRED,"GRIE #2","CA #1", "CB #1"))
// bestimme den Typ
    IFANY2(OPTIONAL,1,"CA-1 #1", "CB-1 #1",GrieMap(REQUIRED,"GRIE #1","CA-1 #1", "CB-1 #1"))
    IFANY2(OPTIONAL,1,"CA #1", "CB #1",GrieMap(REQUIRED,"GRIE #2","CA #1", "CB #1"))
```

Text D−2 completeGly.noSub.m1

```
// -*- c++ -*-
//
// sucht nach einem Gly-X Muster.
//
// berücksichtigt:
//   CA intra Verknüpfung,Volumen (HNCA), Vorzeichen CB(+-1) in CBCANH
//   für Gly-X
//
// noch nicht:  vorbelegen mit Resourcen   (zB aus lokalen Bedingungen) IFNOTEXISTS( ..., ...)
// noch nicht:  strategie varianten je nach lokalen Bedingungen (lokalen mini Contexten)
// MAKEPATTERN MIN:  $condCount - 10
//   #34
#include "ASContexte.macros.new"

#define  CA_SCHWELLE     0.05
#define  GLY_SCHWELLE    0.1
#define  CB_SCHWELLE     0.05
#define  TOL_PCT         0.05

#define  HNCO            "hnco CO-1 #1"
#define  HNCA_CA_m1      "hnca CA-1 #1"
#define  HNCA_CA         "hnca CA #1"

#define  CBCA_CONH_CA_m1  "cbca_conh CA-1 #1"
//#define  CBCA_CONH_CB_m1  "cbca_conh CB-1 #1"

#define  CBCANH_CA_m1     "cbcanh CA-1 #1"
//#define  CBCANH_CB_m1     "cbcanh CB-1 #1"
#define  CBCANH_CA        "cbcanh CA #1"
#define  CBCANH_CB        "cbcanh CB #1"

    PEAKLISTMEMBER(REQUIRED,HNCO,"hnco")
    LOCKPEAK(REQUIRED,HNCO)

//  alle CA-1 Nodes, dann mittelwert bilden und als CA bewerten
    find2PATH(OPTIONAL, HNCO,HNCA_CA_m1,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,AXIS_hnco_n,AXIS_hnca_n)
    find2PATH(OPTIONAL, HNCO,CBCA_CONH_CA_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,AXIS_hnco_n,AXIS_cbca_conh_n)
    find2PATH(OPTIONAL, HNCO,CBCANH_CA_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
    IFEXISTS1(REQUIRED,CBCANH_CA_m1,TESTVOLUME(OPTIONAL,CBCANH_CA_m1,1))
    IFEXISTS2(REQUIRED,CBCA_CONH_CA_m1,CBCANH_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCA_CONH_CA_m1,CBCANH_CA_m1,
              AXIS_cbca_conh_ca,AXIS_cbcanh_ca))
    IFEXISTS2(REQUIRED,CBCA_CONH_CA_m1,HNCA_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCA_CONH_CA_m1,HNCA_CA_m1,
              AXIS_cbca_conh_ca,AXIS_hnca_ca))
    IFEXISTS2(REQUIRED,CBCANH_CA_m1,HNCA_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCANH_CA_m1,HNCA_CA_m1,AXIS_cbcanh_ca,AXIS_hnca_ca))
    AverageFreq3(REQUIRED,"CA-1 #1",AVRES_hnca_ca_m1,AVRES_cbca_conh_ca_m1,AVRES_cbcanh_ca_m1)
    IFEXISTS1(REQUIRED,"CA-1 #1",TESTEXTERNFUZZYMAP1D_VALUE(REQUIRED,"CA-1 #1",GLY_SCHWELLE,
              "/simul/va/assign/Gr/linFunc.istEinGly.data"))

//  alle CA Nodes, dann mittelwert bilden und als CA bewerten
    find2PATH(OPTIONAL, HNCO,HNCA_CA,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,AXIS_hnco_n,AXIS_hnca_n)
//  kein volumenvergleich HNCA CA <-> CA-1 im suchmuster!
//    IFEXISTS2(REQUIRED,HNCA_CA_m1,HNCA_CA, TESTCOMPVOLUME(OPTIONAL,HNCA_CA,HNCA_CA_m1,TOL_PCT,0.5))
    find2PATH(OPTIONAL, HNCO,CBCANH_CA,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
    IFEXISTS1(REQUIRED,CBCANH_CA,TESTVOLUME(OPTIONAL,CBCANH_CA,1))
    IFEXISTS2(REQUIRED,CBCANH_CA,HNCA_CA,TESTNODESARELINKED(OPTIONAL,HNCA_CA,CBCANH_CA,AXIS_cbcanh_ca,AXIS_hnca_ca))
    AverageFreq2(REQUIRED,"CA #1",AVRES_hnca_ca_p1, AVRES_cbcanh_ca_p1)
    IFEXISTS1(REQUIRED,"CA #1",TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CA #1",CA_SCHWELLE,
              "/simul/va/assign/Gr/linFunc.istEinCA_ohneGly.data"))

// kein CB-1 ( Gly-X Pattern !!)

// #ifdef FIND_CB
// // alle CB
   // if CA-1 == GLY
     find2PATH(OPTIONAL, HNCO,CBCANH_CB,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
     IFEXISTS1(REQUIRED,CBCANH_CB,TESTVOLUME(OPTIONAL,CBCANH_CB,-1))
     IFEXISTS1(REQUIRED,CBCANH_CB,AverageFreq1(OPTIONAL,"CB #1", AVRES_cbcanh_cb_p1))
     IFEXISTS1(REQUIRED,"CB #1", TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CB #1",CB_SCHWELLE,
              "/simul/va/assign/Gr/linFunc.istEinCB.manuell.data"))
// #endif

    IFANY4(REQUIRED,2,"CA-1 #1", "CB #1", "CB #1", "CA #1",CONST_TRUE(REQUIRED))
// test auf moegliche AS     (CB-1 muss drin bleiben !!!)
    IFANY2(OPTIONAL,1,"CA-1 #1", "CB-1 #1",AminoTest(REQUIRED,"GRIE #1","CA-1 #1", "CB-1 #1"))
    IFANY2(OPTIONAL,1,"CA #1", "CB #1",AminoTest(REQUIRED,"GRIE #2","CA #1", "CB #1"))
// bestimme den Typ
    IFANY2(OPTIONAL,1,"CA-1 #1", "CB-1 #1",GrieMap(REQUIRED,"GRIE #1","CA-1 #1", "CB-1 #1"))
    IFANY2(OPTIONAL,1,"CA #1", "CB #1",GrieMap(REQUIRED,"GRIE #2","CA #1", "CB #1"))
```

Text D−3 completeGly.noSub.np

```
// -*- c++ -*-
//
// sucht nach einem Gly-Gly Muster.
//
// berücksichtigt:
//   CA intra Verknüpfung,Volumen (HNCA)
//
// PATTERN MAX_USED_NODES:      6
// MAKEPATTERN MIN:  $condCount - 10
//   #34
#include "ASContexte.macros.new"

#define  CA_SCHWELLE     0.05
#define  GLY_SCHWELLE    0.1
#define  CB_SCHWELLE     0.05
#define  TOL_PCT         0.05

#define  HNCO            "hnco CO-1 #1"
#define  HNCA_CA_m1      "hnca CA-1 #1"
#define  HNCA_CA         "hnca CA #1"

#define  CBCA_CONH_CA_m1  "cbca_conh CA-1 #1"

#define  CBCANH_CA_m1     "cbcanh CA-1 #1"
#define  CBCANH_CA        "cbcanh CA #1"

    PEAKLISTMEMBER(REQUIRED,HNCO,"hnco")
    LOCKPEAK(REQUIRED,HNCO)

//  alle CA-1 Nodes, dann mittelwert bilden und als CA bewerten
    find2PATH(OPTIONAL, HNCO,HNCA_CA_m1,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,AXIS_hnco_n,AXIS_hnca_n)
    find2PATH(OPTIONAL, HNCO,CBCA_CONH_CA_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,AXIS_hnco_n,AXIS_cbca_conh_n)
    find2PATH(OPTIONAL, HNCO,CBCANH_CA_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
    IFEXISTS1(REQUIRED,CBCANH_CA_m1,TESTVOLUME(OPTIONAL,CBCANH_CA_m1,1))
    IFEXISTS2(REQUIRED,CBCA_CONH_CA_m1,CBCANH_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCA_CONH_CA_m1,CBCANH_CA_m1,
                AXIS_cbca_conh_ca,AXIS_cbcanh_ca))
    IFEXISTS2(REQUIRED,CBCA_CONH_CA_m1,HNCA_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCA_CONH_CA_m1,HNCA_CA_m1,
                AXIS_cbca_conh_ca,AXIS_hnca_ca))
    IFEXISTS2(REQUIRED,CBCANH_CA_m1,HNCA_CA_m1,TESTNODESARELINKED(OPTIONAL,CBCANH_CA_m1,HNCA_CA_m1,AXIS_cbcanh_ca,AXIS_hnca_ca))
    AverageFreq3(REQUIRED,"CA-1 #1",AVRES_hnca_ca_m1,AVRES_cbca_conh_ca_m1,AVRES_cbcanh_ca_m1)
    IFEXISTS1(REQUIRED,"CA-1 #1",TESTEXTERNFUZZYMAP1D_VALUE(REQUIRED,"CA-1 #1",GLY_SCHWELLE,
                "/simul/va/assign/Gr/linFunc.istEinGly.data"))

//  alle CA Nodes, dann mittelwert bilden und als CA bewerten
    find2PATH(OPTIONAL, HNCO,HNCA_CA,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,AXIS_hnco_n,AXIS_hnca_n)
//  kein volumenvergleich HNCA CA <-> CA-1 im suchmuster!
//    IFEXISTS2(REQUIRED,HNCA_CA_m1,HNCA_CA, TESTCOMPVOLUME(OPTIONAL,HNCA_CA,HNCA_CA_m1,TOL_PCT,0.5))
    find2PATH(OPTIONAL, HNCO,CBCANH_CA,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,AXIS_cbcanh_n)
    IFEXISTS1(REQUIRED,CBCANH_CA,TESTVOLUME(OPTIONAL,CBCANH_CA,1))
    IFEXISTS2(REQUIRED,CBCANH_CA,HNCA_CA,TESTNODESARELINKED(OPTIONAL,HNCA_CA,CBCANH_CA,AXIS_cbcanh_ca,AXIS_hnca_ca))
    AverageFreq2(REQUIRED,"CA #1",AVRES_hnca_ca_p1, AVRES_cbcanh_ca_p1)
    IFEXISTS1(REQUIRED,"CA #1",TESTEXTERNFUZZYMAP1D_VALUE(OPTIONAL,"CA
                #1",CA_SCHWELLE,"/simul/va/assign/Gr/linFunc.istEinGly.data"))

// kein CB-1 ( Gly-Gly Pattern !!)
// kein CB   ( Gly-Gly Pattern !!)

//  IFANY4(REQUIRED,2,"CA-1 #1", "CB #1", "CB #1", "CA #1",CONST_TRUE(REQUIRED))
// test auf moegliche AS     (CB-1 und CB  muss drin bleiben !!!)
    IFANY2(OPTIONAL,1,"CA-1 #1", "CB-1 #1",AminoTest(REQUIRED,"GRIE #1","CA-1 #1", "CB-1 #1"))
    IFANY2(OPTIONAL,1,"CA #1", "CB #1",AminoTest(REQUIRED,"GRIE #2","CA #1", "CB #1"))
// bestimme den Typ
    IFANY2(OPTIONAL,1,"CA-1 #1", "CB-1 #1",GrieMap(REQUIRED,"GRIE #1","CA-1 #1", "CB-1 #1"))
    IFANY2(OPTIONAL,1,"CA #1", "CB #1",GrieMap(REQUIRED,"GRIE #2","CA #1", "CB #1"))
```

Text D−4 completeGly.noSub.both

D.2 Automatische knotenlokale Bedingungen

```
// -*- c++ -*-
// finde die HNCA Peaks zu einem HNCO
//
//#7
#include "ASContexte.macros"
#define  HNCO_CO       "hnco CO-1 #1"
#define  HNCA_CA       "hnca CA #1"
#define  HNCA_CA_m1    "hnca CA-1 #1"
    PEAKLISTMEMBER(HNCO_CO,"hnco")
    LOCKPEAK(HNCO_CO)
    find2PATH(REQUIRED,HNCO_CO,HNCA_CA,"hnca",AXIS_hnco_hn,AXIS_hnca_hn, AXIS_hnco_n,AXIS_hnca_n)
    find2PATH(REQUIRED,HNCO_CO,HNCA_CA_m1,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,
            AXIS_hnco_n,AXIS_hnca_n)
    AverageFreq1("CA-1",AVRES_hnca_ca_m1)
    AverageFreq1("CA",AVRES_hnca_ca_p1)
```

Muster D−5 listHNCA.complete

```
// -*- c++ -*-
// finde eine zweier Gruppe moeglicher CA Peaks zu einem HNCO
// sucht in hnca, cbcanh
//
// #8
#include "ASContexte.macros"
#define  HNCO_CO       "hnco CO-1 #1"
#define  CBCANH_CA_p1  "cbcanh CA #1"
#define  HNCA_CA_p1    "hnca CA #1"
    PEAKLISTMEMBER(HNCO_CO,"hnco")
    LOCKPEAK(HNCO_CO)
    find2PATH(REQUIRED,HNCO_CO,HNCA_CA_p1,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,
            AXIS_hnco_n,AXIS_hnca_n)
    find2PATH(REQUIRED,HNCO_CO,CBCANH_CA_p1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,AXIS_hnco_n,
            AXIS_cbcanh_n)
    TESTNODESARELINKED(HNCA_CA_p1,CBCANH_CA_p1,AXIS_cbcanh_ca,AXIS_hnca_ca)
    AverageFreq2("CA",AVRES_hnca_ca_p1,AVRES_cbcanh_ca_p1)
    TESTVOLUME(CBCANH_CA_p1,1)
    TESTEXTERNFUZZYMAP1D_VALUE("CA",0.4,"/simul/va/assign/Gr/linFunc.istEinCA.data")
```

Text D−6 listCA.complete

```
// -*- c++ -*-
// finde die CBCA(CO)NH Peaks zu einem HNCO
//
// #8
#include "ASContexte.macros"
#define  HNCO_CO       "hnco CO-1 #1"
#define  CBCA_CONH_CB_m1    "cbca_conh CB-1 #1"
#define  CBCA_CONH_CA_m1    "cbca_conh CA-1 #1"
    PEAKLISTMEMBER(HNCO_CO,"hnco")
    LOCKPEAK(HNCO_CO)
    find2PATH(REQUIRED,HNCO_CO,CBCA_CONH_CB_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,
            AXIS_hnco_n,AXIS_cbca_conh_n)
    find2PATH(REQUIRED,HNCO_CO,CBCA_CONH_CA_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,
            AXIS_hnco_n,AXIS_cbca_conh_n)
    AverageFreq1("CA-1",AVRES_cbca_conh_ca_m1)
    AverageFreq1("CB-1",AVRES_cbca_conh_cb_m1)
    TESTEXTERNFUZZYMAP1D_VALUE("CA-1",0.4,"/simul/va/assign/Gr/linFunc.istEinCA.data")
    TESTEXTERNFUZZYMAP1D_VALUE("CB-1",0.4,"/simul/va/assign/Gr/linFunc.istEinCB.data")
```

Muster D−7 listCBCA_CONH.complete

```
// -*- c++ -*-
// finde die CBCANH Peaks zu einem HNCO
//
// # 18
#include "ASContexte.macros"
#define  HNCO_CO          "hnco CO-1 #1"
#define  CBCANH_CB_m1     "cbcanh CB-1 #1"
#define  CBCANH_CA_m1     "cbcanh CA-1 #1"
#define  CBCANH_CB_p1     "cbcanh CB #1"
#define  CBCANH_CA_p1     "cbcanh CA #1"
    PEAKLISTMEMBER(HNCO_CO,"hnco")
    LOCKPEAK(HNCO_CO)
    find2PATH(REQUIRED,HNCO_CO,CBCANH_CB_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,
            AXIS_hnco_n,AXIS_cbcanh_n)
    find2PATH(REQUIRED,HNCO_CO,CBCANH_CA_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,
            AXIS_hnco_n,AXIS_cbcanh_n)
    find2PATH(REQUIRED,HNCO_CO,CBCANH_CB_p1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,
            AXIS_hnco_n,AXIS_cbcanh_n)
    find2PATH(REQUIRED,HNCO_CO,CBCANH_CA_p1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,
            AXIS_hnco_n,AXIS_cbcanh_n)
    TESTVOLUME(CBCANH_CB_m1,-1)
    TESTVOLUME(CBCANH_CB_p1,-1)
    TESTVOLUME(CBCANH_CA_m1,1)
    TESTVOLUME(CBCANH_CA_p1,1)
    AverageFreq1("CA-1",AVRES_cbcanh_ca_m1)
    AverageFreq1("CB-1",AVRES_cbcanh_cb_m1)
    AverageFreq1("CA"  ,AVRES_cbcanh_ca_p1)
    AverageFreq1("CB"  ,AVRES_cbcanh_cb_p1)
    TESTEXTERNFUZZYMAP1D_VALUE("CA-1",0.4,"/simul/va/assign/Gr/linFunc.istEinCA.data")
    TESTEXTERNFUZZYMAP1D_VALUE("CB-1",0.4,"/simul/va/assign/Gr/linFunc.istEinCB.data")
    TESTEXTERNFUZZYMAP1D_VALUE("CA",0.4,"/simul/va/assign/Gr/linFunc.istEinCA.data")
    TESTEXTERNFUZZYMAP1D_VALUE("CB",0.4,"/simul/va/assign/Gr/linFunc.istEinCB.data")
```

Muster D−8 listCBCANH.complete

```
// -*- c++ -*-
// finde eine dreier Gruppe moeglicher CA-1 Peaks zu einem HNCO
// sucht in hnca, cbcanh, cbca_conh
//
// #10
#include "ASContexte.macros"
#define  HNCO_CO       "hnco CO-1 #1"
#define  CBCA_CONH_CB_m1    "cbca_conh CB-1 #1"
#define  CBCA_CONH_CA_m1    "cbca_conh CA-1 #1"
#define  CBCANH_CA_m1       "cbcanh CA-1 #1"
#define  HNCA_CA            "hnca CA #1"
#define  HNCA_CA_m1         "hnca CA-1 #1"
    PEAKLISTMEMBER(HNCO_CO,"hnco")
    LOCKPEAK(HNCO_CO)
    find2PATH(REQUIRED,HNCO_CO,HNCA_CA_m1,"hnca",AXIS_hnco_hn,AXIS_hnca_hn,
            AXIS_hnco_n,AXIS_hnca_n)
    find2PATH(REQUIRED,HNCO_CO,CBCA_CONH_CA_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,
            AXIS_hnco_n,AXIS_cbca_conh_n)
    TESTNODESARELINKED(HNCA_CA_m1,CBCA_CONH_CA_m1,AXIS_cbca_conh_ca,AXIS_hnca_ca)
    find2PATH(REQUIRED,HNCO_CO,CBCANH_CA_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,
            AXIS_hnco_n,AXIS_cbcanh_n)
    TESTNODESARELINKED(CBCANH_CA_m1,CBCA_CONH_CA_m1,AXIS_cbca_conh_ca,AXIS_cbcanh_ca)
    TESTVOLUME(CBCANH_CA_m1,1)
    AverageFreq3("CA-1",AVRES_hnca_ca_m1,AVRES_cbca_conh_ca_m1,AVRES_cbcanh_ca_m1)
    TESTEXTERNFUZZYMAP1D_VALUE("CA-1",0.4,"/simul/va/assign/Gr/linFunc.istEinCA.data")
```

Muster D−9 listCA_m1.complete

```
// -*- c++ -*-
// finde eine zweier Gruppe moeglicher CB-1 Peaks zu einem HNCO
// sucht in cbcanh, cbca_conh
//
// #8
#include "ASContexte.macros"
#define  HNCO_CO            "hnco CO-1 #1"
#define  CBCA_CONH_CB_m1    "cbca_conh CB-1 #1"
#define  CBCANH_CB_m1       "cbcanh CB-1 #1"
    PEAKLISTMEMBER(HNCO_CO,"hnco")
    LOCKPEAK(HNCO_CO)
    find2PATH(REQUIRED,HNCO_CO,CBCA_CONH_CB_m1,"cbca_conh",AXIS_hnco_hn,AXIS_cbca_conh_hn,
          AXIS_hnco_n,AXIS_cbca_conh_n)
    find2PATH(REQUIRED,HNCO_CO,CBCANH_CB_m1,"cbcanh",AXIS_hnco_hn,AXIS_cbcanh_hn,
          AXIS_hnco_n,AXIS_cbcanh_n)
    TESTNODESARELINKED(CBCANH_CB_m1,CBCA_CONH_CB_m1,AXIS_cbca_conh_ca,AXIS_cbcanh_ca)
    TESTVOLUME(CBCANH_CB_m1,-1)
    AverageFreq2("CB-1",AVRES_cbca_conh_cb_m1,AVRES_cbcanh_cb_m1)
    TESTEXTERNFUZZYMAP1D_VALUE("CB-1",0.4,"/simul/va/assign/Gr/linFunc.istEinCB.data")
```

Muster D−10 listCB_m1.complete

D.3 Linkprotospinsystemmuster

```
// -*- c++ -*-
// teste die CA, und CB Brücken
// sucht in hnca, cbcanh, cbca_conh
// MAKEPATTERN TITLE: "teste alle Links von i nach i+1"
// #10
#include "../ASContexte.macros.new"
#define  HNCO_CO              "hnco CO-1 #1"
//#define  CBCA_CONH_CB_m1      "cbca_conh CB-1 #1"
//#define  CBCA_CONH_CA_m1      "cbca_conh CA-1 #1"
//#define  CBCANH_CA_m1         "cbcanh CA-1 #1"
//#define  CBCANH_CB_m1         "cbcanh CB-1 #1"
#define  CBCANH_CA            "cbcanh CA #1"
#define  CBCANH_CB            "cbcanh CB #1"
#define  HNCA_CA              "hnca CA #1"
#define  HNCA_CA_m1           "hnca CA-1 #1"

#define  xHNCO_CO             "hnco CO-1 #2"
#define  xCBCA_CONH_CB_m1     "cbca_conh CB-1 #2"
#define  xCBCA_CONH_CA_m1     "cbca_conh CA-1 #2"
#define  xCBCANH_CA_m1        "cbcanh CA-1 #2"
#define  xCBCANH_CB_m1        "cbcanh CB-1 #2"
// #define  xCBCANH_CA           "cbcanh CA #2"
// #define  xCBCANH_CB           "cbcanh CB #2"
// #define  xHNCA_CA             "hnca CA #2"
#define  xHNCA_CA_m1          "hnca CA-1 #2"
    PEAKLISTMEMBER(REQUIRED,HNCO_CO,"hnco")
    LOCKPEAK(REQUIRED,HNCO_CO)

    PEAKLISTMEMBER(REQUIRED,xHNCO_CO,"hnco")
    LOCKPEAK(REQUIRED,xHNCO_CO)
// teste CA Brücken von HNCA CA
    IFEXISTS2(REQUIRED, HNCA_CA,xHNCA_CA_m1,
         TESTNODESARELINKED(OPTIONAL,HNCA_CA,xHNCA_CA_m1,AXIS_hnca_ca,AXIS_hnca_ca))
    IFEXISTS2(REQUIRED, HNCA_CA,xCBCANH_CA_m1,
         TESTNODESARELINKED(OPTIONAL,HNCA_CA,xCBCANH_CA_m1,AXIS_cbcanh_ca,AXIS_hnca_ca))
    IFEXISTS2(REQUIRED, HNCA_CA,xCBCA_CONH_CA_m1,
         TESTNODESARELINKED(OPTIONAL,HNCA_CA,xCBCA_CONH_CA_m1,AXIS_cbca_conh_ca,AXIS_hnca_ca))

// teste CA Brücken von CBCANH CA
    IFEXISTS2(REQUIRED, CBCANH_CA,xHNCA_CA_m1,
         TESTNODESARELINKED(OPTIONAL,CBCANH_CA,xHNCA_CA_m1,AXIS_hnca_ca,AXIS_cbcanh_ca))
    IFEXISTS2(REQUIRED, CBCANH_CA,xCBCANH_CA_m1,
         TESTNODESARELINKED(OPTIONAL,CBCANH_CA,xCBCANH_CA_m1,AXIS_cbcanh_ca,AXIS_cbcanh_ca))
    IFEXISTS2(REQUIRED, CBCANH_CA,xCBCA_CONH_CA_m1,
         TESTNODESARELINKED(OPTIONAL,CBCANH_CA,xCBCA_CONH_CA_m1,AXIS_cbca_conh_ca,AXIS_cbcanh_ca))

// teste CB Brücken von CBCANH CB
    IFEXISTS2(REQUIRED, CBCANH_CB,xCBCANH_CB_m1,
         TESTNODESARELINKED(OPTIONAL,CBCANH_CB,xCBCANH_CB_m1,AXIS_cbcanh_ca,AXIS_cbcanh_ca))
    IFEXISTS2(REQUIRED, CBCANH_CB,xCBCA_CONH_CB_m1,
         TESTNODESARELINKED(OPTIONAL,CBCANH_CB,xCBCA_CONH_CB_m1,AXIS_cbca_conh_ca,AXIS_cbcanh_ca))
```

Muster D−11 testBridge

Anhang E Peaklisten des Trigger Proteins

Die folgenden Listen enthalten alle Peaks der Trigger Spektren. Die virtuellen HNCO Peaks sind in den Listen enthalten und haben Identifikationsnummern ab 200.

E.1 HNCO

ID	HN	N	CO	Volumen
1	9.399	132.981	172.502	100000000.000
2	9.699	131.930	173.235	100000000.000
3	9.400	131.701	172.282	100000000.000
5	8.520	129.220	173.038	100000000.000
7	8.993	128.945	171.198	100000000.000
8	8.903	128.750	171.302	100000000.000
9	9.213	127.872	170.874	100000000.000
12	8.667	126.540	173.462	100000000.000
15	9.304	126.350	173.360	100000000.000
16	7.779	126.337	172.386	100000000.000
17	9.381	126.061	171.020	100000000.000
18	9.140	124.968	173.146	100000000.000
20	9.076	124.727	171.102	100000000.000
21	8.832	124.447	168.498	100000000.000
23	8.451	123.930	172.318	100000000.000
24	8.130	123.093	172.127	100000000.000
25	8.343	123.065	171.363	100000000.000
27	8.764	122.847	173.352	100000000.000
28	8.460	122.807	171.529	100000000.000
30	8.619	122.548	171.851	100000000.000
31	8.167	122.579	171.423	100000000.000
32	9.118	122.317	173.948	100000000.000
33	8.980	122.279	172.484	100000000.000
34	8.916	122.335	171.201	100000000.000
35	9.415	122.276	170.689	100000000.000
36	8.208	122.252	169.475	100000000.000
37	8.188	122.048	171.743	100000000.000
39	9.298	122.057	170.349	100000000.000
41	8.230	121.523	171.423	100000000.000
43	9.748	121.196	172.181	100000000.000
44	7.647	121.192	172.177	100000000.000
46	8.410	120.926	172.071	100000000.000
47	7.939	120.663	170.396	100000000.000
48	8.313	120.450	172.391	100000000.000
49	8.228	120.399	171.707	100000000.000
51	8.245	120.125	173.575	100000000.000
53	8.242	119.837	174.654	100000000.000
55	5.408	119.645	171.949	100000000.000
56	8.407	119.625	170.892	100000000.000
57	7.787	119.590	170.740	100000000.000
59	8.133	118.510	170.774	100000000.000
60	9.267	114.775	175.972	100000000.000
61	7.870	118.043	173.147	100000000.000
63	8.839	117.789	171.096	100000000.000
64	7.203	116.970	170.132	100000000.000
65	7.997	116.706	174.037	100000000.000
66	9.151	116.706	172.083	100000000.000
67	7.752	116.641	171.311	100000000.000
68	8.499	116.653	170.021	100000000.000
70	7.324	116.447	173.451	100000000.000
72	6.704	116.451	173.352	100000000.000
74	8.001	116.162	170.441	100000000.000
75	7.681	115.882	172.716	100000000.000
77	8.168	115.039	171.635	100000000.000
83	8.480	114.017	173.375	100000000.000
84	8.937	113.749	172.932	100000000.000
85	8.237	113.741	171.848	100000000.000
86	7.690	113.249	172.700	100000000.000
87	8.473	112.885	172.600	100000000.000
88	8.715	112.396	173.527	100000000.000
89	8.950	112.142	172.930	100000000.000
90	8.633	110.245	173.353	100000000.000
91	8.146	108.389	176.593	100000000.000
92	7.745	107.311	175.666	100000000.000
93	7.293	108.429	174.113	100000000.000
94	10.282	105.949	169.149	100000000.000
96	9.607	128.185	173.048	100000000.000
97	8.510	127.354	171.182	100000000.000
98	8.604	127.113	172.724	100000000.000
99	8.825	126.145	173.081	100000000.000
100	8.694	125.522	170.968	100000000.000
101	7.921	127.453	170.504	100000000.000
102	7.709	126.815	171.316	100000000.000
103	8.159	124.005	173.979	100000000.000
104	8.629	123.674	172.920	100000000.000
105	9.715	119.920	173.991	100000000.000
106	7.752	119.348	170.825	100000000.000
107	9.355	117.439	174.028	100000000.000
108	8.964	114.807	172.791	100000000.000
109	8.601	106.467	172.120	100000000.000
110	7.086	109.243	170.351	100000000.000
111	8.118	120.910	172.768	100000000.000
112	8.065	120.690	172.779	100000000.000
113	8.116	120.103	169.920	100000000.000
114	8.847	119.632	173.532	100000000.000
115	9.139	117.722	175.561	100000000.000
116	8.568	118.850	171.388	100000000.000
117	8.470	118.510	171.329	100000000.000
118	8.240	116.104	172.159	100000000.000
119	8.446	115.893	172.815	100000000.000
120	8.072	114.573	173.775	100000000.000
121	7.927	115.056	171.225	100000000.000
122	9.146	133.229	172.484	100000000.000
123	8.043	121.907	173.491	100000000.000
124	7.993	121.630	168.735	100000000.000
125	8.290	122.054	171.337	100000000.000
126	7.100	116.847	172.729	100000000.000
127	8.411	117.786	173.606	100000000.000
128	10.849	123.240	172.572	100000000.000
129	6.856	111.368	174.119	100000000.000
130	6.787	114.327	172.333	100000000.000
131	9.379	119.116	173.353	100000000.000
132	7.312	119.306	173.439	100000000.000
133	7.394	124.190	171.345	100000000.000
134	7.117	119.884	172.575	100000000.000
135	9.378	127.446	171.915	100000000.000
200	8.310	115.000	168.441	100000000.000
201	7.804	125.159	168.441	100000000.000
203	6.920	113.750	168.441	100000000.000

E.2 HNCA

ID	HN	N	CO	CA	Volumen
1001	8.601	106.471	100000000.000	54.882	131800000.000
1002	7.745	107.069	100000000.000	64.568	68820000.000
1003	7.743	107.260	100000000.000	59.633	13180000.000
1004	7.294	108.507	100000000.000	45.485	62680000.000
1005	7.295	108.571	100000000.000	64.473	24130000.000
1006	7.083	109.162	100000000.000	57.306	135300000.000
1007	7.085	109.076	100000000.000	49.833	12930000.000
1008	6.858	111.497	100000000.000	57.714	52600000.000
1009	6.857	111.475	100000000.000	60.238	37470000.000
1010	8.474	112.739	100000000.000	59.150	38540000.000
1011	8.475	112.755	100000000.000	53.997	12200000.000
1012	8.716	112.727	100000000.000	44.786	34420000.000
1013	8.716	112.784	100000000.000	59.001	13310000.000
1014	7.686	113.363	100000000.000	53.051	24070000.000
1015	7.689	113.426	100000000.000	61.084	102000000.000
1016	8.480	113.667	100000000.000	44.728	37980000.000
1017	8.480	113.628	100000000.000	55.242	32650000.000
1018	8.940	113.747	100000000.000	52.299	28780000.000
1019	8.941	113.769	100000000.000	55.492	132200000.000
1020	8.080	114.453	100000000.000	55.807	37510000.000
1021	8.073	114.490	100000000.000	61.644	235100000.000
1022	9.267	114.752	100000000.000	58.704	135300000.000
1023	8.175	115.051	100000000.000	61.794	117500000.000
1024	7.930	115.191	100000000.000	44.935	19650000.000
1025	7.931	115.077	100000000.000	59.326	94300000.000
1026	8.313	115.089	100000000.000	58.316	83670000.000
1027	8.971	114.852	100000000.000	60.237	79870000.000
1028	8.233	115.457	100000000.000	53.373	33970000.000
1029	8.451	115.857	100000000.000	55.403	82800000.000
1030	8.450	115.804	100000000.000	49.794	82020000.000
1031	8.239	116.077	100000000.000	53.450	89380000.000
1032	8.002	116.170	100000000.000	51.322	117400000.000
1033	7.998	116.325	100000000.000	60.510	21740000.000
1034	8.502	116.420	100000000.000	45.385	63730000.000
1035	8.502	116.432	100000000.000	61.138	23490000.000
1036	8.091	116.340	100000000.000	50.939	-981700.000
1037	7.321	116.478	100000000.000	53.378	86940000.000
1038	7.316	116.545	100000000.000	58.411	20390000.000
1039	7.755	116.571	100000000.000	53.649	63280000.000
1040	7.755	116.529	100000000.000	57.726	26340000.000
1041	6.701	116.669	100000000.000	54.555	9884000.000
1042	6.703	116.550	100000000.000	63.921	68170000.000
1043	7.991	116.884	100000000.000	55.277	12350000.000
1044	8.001	116.387	100000000.000	48.386	3075000.000
1045	7.098	116.650	100000000.000	58.153	134700000.000
1046	7.100	116.750	100000000.000	53.368	24840000.000
1047	7.199	117.140	100000000.000	54.316	88510000.000
1048	7.198	116.952	100000000.000	45.407	18260000.000
1049	9.356	117.246	100000000.000	55.655	54740000.000
1050	9.356	117.249	100000000.000	65.716	55240000.000
1051	8.413	117.790	100000000.000	59.287	290500000.000
1052	8.413	117.771	100000000.000	62.249	85770000.000
1053	8.839	117.776	100000000.000	55.533	32540000.000
1054	8.841	117.851	100000000.000	61.123	55610000.000
1055	9.139	117.748	100000000.000	60.170	15820000.000
1056	9.140	117.548	100000000.000	62.059	13930000.000
1057	7.873	118.097	100000000.000	54.313	29150000.000
1058	7.874	118.070	100000000.000	57.491	195000000.000
1059	8.134	118.435	100000000.000	52.013	43930000.000
1060	8.135	118.388	100000000.000	61.089	70570000.000
1061	8.471	118.460	100000000.000	52.293	58990000.000
1062	8.470	118.387	100000000.000	58.871	33750000.000
1063	8.567	118.733	100000000.000	54.534	38280000.000
1064	8.567	118.816	100000000.000	61.360	52050000.000
1065	9.380	119.076	100000000.000	58.846	72630000.000
1066	7.751	119.309	100000000.000	54.479	154100000.000
1067	7.752	119.347	100000000.000	44.701	20830000.000
1068	7.313	119.301	100000000.000	59.483	151800000.000
1069	7.789	119.592	100000000.000	54.607	283500000.000
1070	8.850	119.463	100000000.000	53.480	31280000.000
1071	8.851	119.569	100000000.000	65.738	104200000.000
1072	8.408	119.562	100000000.000	55.554	45870000.000
1073	8.408	119.573	100000000.000	61.003	80620000.000
1074	9.714	119.894	100000000.000	60.759	27460000.000
1075	9.715	119.919	100000000.000	45.293	25460000.000
1076	5.407	119.797	100000000.000	58.398	21580000.000
1077	5.412	119.672	100000000.000	61.395	124900000.000
1078	8.228	120.081	100000000.000	56.234	235000000.000
1079	7.941	120.402	100000000.000	53.361	254400000.000
1080	7.941	120.380	100000000.000	60.676	12100000.000
1081	8.066	120.437	100000000.000	54.984	116600000.000
1082	8.066	120.456	100000000.000	62.052	334800000.000
1083	8.121	120.727	100000000.000	61.840	-997900.000
1084	8.108	120.491	100000000.000	53.602	79030000.000
1085	8.309	120.821	100000000.000	49.718	30230000.000
1086	8.315	120.820	100000000.000	59.773	33030000.000
1087	8.413	120.904	100000000.000	58.229	78990000.000
1088	8.413	120.867	100000000.000	62.596	53110000.000
1089	7.596	121.159	100000000.000	53.746	29020000.000
1090	7.596	121.094	100000000.000	56.778	142200000.000
1091	9.750	121.141	100000000.000	53.800	108400000.000
1092	9.748	121.135	100000000.000	50.753	724400.000
1094	8.230	121.649	100000000.000	53.517	161000000.000
1095	8.231	121.744	100000000.000	59.599	41710000.000
1096	7.994	121.681	100000000.000	52.145	160600000.000
1097	7.993	121.668	100000000.000	57.351	29460000.000
1098	8.045	121.840	100000000.000	54.965	382200000.000
1099	9.303	122.081	100000000.000	58.174	106600000.000
1100	9.305	122.070	100000000.000	54.220	13740000.000
1101	8.288	122.051	100000000.000	61.294	66960000.000
1102	8.292	122.007	100000000.000	55.368	197800000.000
1103	9.414	122.483	100000000.000	55.924	99250000.000
1104	9.417	122.515	100000000.000	60.498	29610000.000
1105	8.189	122.064	100000000.000	53.729	115300000.000
1106	8.188	121.824	100000000.000	56.907	39070000.000
1107	8.919	122.096	100000000.000	62.736	40310000.000
1108	8.918	122.073	100000000.000	53.371	24320000.000
1109	9.123	122.199	100000000.000	54.159	23280000.000
1110	9.120	122.240	100000000.000	63.940	12680000.000
1111	8.619	122.332	100000000.000	56.942	148900000.000
1112	8.632	122.553	100000000.000	50.890	85410.000
1113	8.128	122.636	100000000.000	60.685	65730000.000
1114	8.128	122.677	100000000.000	44.706	30560000.000
1115	8.466	122.662	100000000.000	56.774	91810000.000
1116	8.467	122.755	100000000.000	54.577	40350000.000
1117	8.767	122.886	100000000.000	55.936	88580000.000
1118	8.769	122.953	100000000.000	53.304	32750000.000
1119	8.346	122.986	100000000.000	55.940	130700000.000
1120	8.354	122.843	100000000.000	61.086	25590000.000
1121	10.862	123.113	100000000.000	53.620	11980000.000
1122	8.627	123.628	100000000.000	52.490	92260000.000
1123	8.620	123.644	100000000.000	49.957	4873000.000
1124	8.456	123.853	100000000.000	62.040	139000000.000
1125	8.456	123.875	100000000.000	53.815	352800000.000
1126	8.157	123.971	100000000.000	52.204	342500000.000
1127	8.159	123.823	100000000.000	54.966	104700000.000
1128	7.399	123.837	100000000.000	59.167	28500000.000
1129	7.399	123.928	100000000.000	50.188	100700000.000
1130	8.839	124.438	100000000.000	60.010	30700000.000
1131	8.836	124.412	100000000.000	45.304	54610000.000
1132	9.079	124.736	100000000.000	60.556	121600000.000
1133	9.078	124.618	100000000.000	63.443	1847000.000
1134	9.142	125.155	100000000.000	55.776	58780000.000
1135	9.143	125.245	100000000.000	53.423	26430000.000
1136	8.692	125.435	100000000.000	58.319	121400000.000
1137	8.691	125.423	100000000.000	61.200	32570000.000
1138	8.829	125.837	100000000.000	58.625	72190000.000
1139	8.828	125.793	100000000.000	61.594	32720000.000
1140	9.306	126.096	100000000.000	55.419	23840000.000
1141	9.306	126.102	100000000.000	60.695	59840000.000
1142	9.381	126.187	100000000.000	53.068	80420000.000
1143	9.381	126.167	100000000.000	61.017	39090000.000
1144	9.046	126.188	100000000.000	57.048	369000.000
1145	7.782	126.442	100000000.000	54.178	78340000.000
1146	7.782	126.260	100000000.000	60.343	39640000.000
1147	8.668	126.408	100000000.000	52.813	77400000.000
1148	8.668	126.469	100000000.000	56.690	43480000.000
1149	8.606	126.900	100000000.000	60.465	86880000.000
1150	8.606	126.935	100000000.000	63.428	1200000.000
1151	8.511	127.358	100000000.000	51.102	92280000.000
1152	8.512	127.348	100000000.000	53.241	58750000.000
1153	9.384	127.348	100000000.000	53.608	22610000.000
1154	9.384	127.444	100000000.000	55.746	36080000.000
1155	7.924	127.327	100000000.000	57.875	2450000.000
1156	7.915	127.315	100000000.000	60.707	544100.000
1157	9.608	128.188	100000000.000	55.600	91880000.000
1158	9.608	128.191	100000000.000	60.753	54890000.000
1159	9.210	128.051	100000000.000	53.378	36450000.000
1160	8.997	128.895	100000000.000	61.434	25630000.000
1161	8.998	128.898	100000000.000	49.664	72950000.000
1162	8.521	129.165	100000000.000	60.423	37330000.000
1163	9.692	131.930	100000000.000	49.881	74570000.000
1164	9.690	131.728	100000000.000	55.921	19360000.000
1165	9.402	131.770	100000000.000	60.797	41390000.000
1166	9.403	131.698	100000000.000	63.053	8810000.000
1167	9.145	133.220	100000000.000	57.925	61150000.000
1168	9.147	133.241	100000000.000	60.885	29740000.000
1169	8.248	119.944	100000000.000	53.746	77450000.000
1170	8.248	120.030	100000000.000	62.290	239100000.000
1171	8.166	122.614	100000000.000	55.271	111500000.000
1172	8.114	120.083	100000000.000	56.317	48470000.000

E.3 HNCACB (CBCANH Typ)

ID	HN	N	CO	CA	Volumen
3001	8.991	128.907	100000000.000	60.945	16490000.000
3002	8.992	129.070	100000000.000	49.658	39290000.000
3003	8.993	129.034	100000000.000	37.802	-19420000.000
3004	8.991	128.900	100000000.000	22.175	-27100000.000
3005	9.207	127.927	100000000.000	29.860	-20650000.000
3006	9.205	128.245	100000000.000	45.683	-30700000.000
3007	9.206	127.934	100000000.000	52.841	8901000.000
3008	9.206	127.936	100000000.000	55.104	29380000.000
3009	9.608	128.363	100000000.000	60.495	42810000.000
3010	9.609	128.397	100000000.000	55.752	34340000.000
3011	9.608	128.400	100000000.000	38.929	-63350000.000
3012	9.608	128.372	100000000.000	34.746	-24670000.000
3013	9.379	127.437	100000000.000	55.625	15370000.000
3014	9.378	126.628	100000000.000	53.289	57870000.000
3015	9.378	127.522	100000000.000	41.358	-15240000.000
3016	9.380	127.923	100000000.000	31.229	-19890000.000
3017	9.378	126.358	100000000.000	69.338	-28220000.000
3018	9.378	126.420	100000000.000	60.837	26830000.000
3019	9.378	126.393	100000000.000	53.266	63290000.000
3020	9.378	126.404	100000000.000	44.883	-51550000.000
3021	9.304	126.626	100000000.000	60.633	65990000.000
3022	9.304	126.644	100000000.000	55.325	28520000.000
3023	9.304	126.638	100000000.000	34.488	-44020000.000
3024	9.304	126.611	100000000.000	30.617	-22280000.000
3025	8.518	129.403	100000000.000	61.602	24680000.000
3026	8.518	129.414	100000000.000	59.397	36520000.000
3027	8.520	129.408	100000000.000	32.300	-38990000.000
3028	8.519	129.407	100000000.000	30.849	-18400000.000
3029	8.506	127.529	100000000.000	53.595	13380000.000
3030	8.504	127.532	100000000.000	51.448	45860000.000
3031	8.506	127.498	100000000.000	44.874	-39250000.000
3032	8.506	127.539	100000000.000	18.606	-46250000.000
3033	8.596	127.322	100000000.000	60.358	80250000.000
3034	8.597	127.298	100000000.000	39.154	-18250000.000
3035	8.598	127.338	100000000.000	34.745	-37670000.000
3036	8.665	126.604	100000000.000	56.556	40330000.000
3037	8.665	126.629	100000000.000	53.015	43260000.000
3038	8.664	126.653	100000000.000	42.996	-35910000.000
3039	8.662	126.610	100000000.000	31.982	-25410000.000
3040	8.831	126.241	100000000.000	61.301	23530000.000
3041	8.831	126.272	100000000.000	58.731	62290000.000
3042	8.831	125.335	100000000.000	39.307	-14620000.000
3043	8.831	126.287	100000000.000	32.013	-38400000.000
3044	8.693	125.679	100000000.000	71.268	-18340000.000
3045	8.692	125.648	100000000.000	61.071	18660000.000
3046	8.693	125.702	100000000.000	58.307	74050000.000
3047	8.692	125.939	100000000.000	38.438	-39610000.000
3048	8.832	124.623	100000000.000	60.063	39070000.000
3049	8.832	124.563	100000000.000	45.269	44240000.000
3050	8.832	124.660	100000000.000	39.202	-27450000.000
3051	7.895	126.305	100000000.000	57.405	18610000.000
3052	7.896	126.304	100000000.000	33.531	-16060000.000
3053	7.781	126.555	100000000.000	60.281	30210000.000
3054	7.782	126.553	100000000.000	54.300	57040000.000
3055	7.781	126.562	100000000.000	41.303	-29420000.000
3056	7.781	126.559	100000000.000	35.018	-26490000.000
3057	7.707	126.945	100000000.000	58.040	34480000.000
3058	7.708	127.004	100000000.000	54.884	70650000.000
3059	7.706	126.970	100000000.000	40.277	-28980000.000
3060	7.707	127.010	100000000.000	29.892	-48560000.000
3061	7.581	121.172	100000000.000	56.617	232800000.000
3062	7.582	121.171	100000000.000	53.478	43870000.000
3063	7.581	121.149	100000000.000	41.079	-51990000.000
3064	7.581	121.146	100000000.000	33.241	-100000000.000
3065	8.161	123.561	100000000.000	54.948	64800000.000
3066	8.160	124.167	100000000.000	52.128	365600000.000
3067	8.160	124.173	100000000.000	42.030	-80300000.000
3068	8.161	124.174	100000000.000	19.149	-100000000.000
3069	8.455	124.018	100000000.000	61.875	112600000.000
3070	8.455	124.010	100000000.000	53.798	271600000.000
3071	8.456	124.030	100000000.000	41.055	-100000000.000
3073	9.076	124.909	100000000.000	60.409	89100000.000
3074	9.076	124.928	100000000.000	34.856	-48460000.000
3075	9.140	125.133	100000000.000	55.358	81460000.000
3076	9.137	124.901	100000000.000	54.291	1858000.000
3077	9.140	125.149	100000000.000	37.620	-62510000.000
3078	9.137	125.334	100000000.000	34.911	-17680000.000
3079	9.745	121.161	100000000.000	53.810	62470000.000
3080	9.746	121.201	100000000.000	36.887	-35270000.000
3081	9.746	121.163	100000000.000	32.047	-28260000.000
3082	9.711	119.968	100000000.000	60.623	15000000.000
3083	9.713	119.946	100000000.000	45.046	29690000.000
3084	9.712	119.973	100000000.000	33.657	-10680000.000
3085	9.299	122.146	100000000.000	58.046	66880000.000
3086	9.299	122.120	100000000.000	54.108	13890000.000
3087	9.299	122.170	100000000.000	40.289	-53560000.000
3088	9.298	122.142	100000000.000	36.803	-18440000.000
3089	8.977	122.072	100000000.000	55.628	59900000.000
3090	8.975	122.047	100000000.000	42.403	-20680000.000
3091	8.976	122.123	100000000.000	30.593	-41330000.000
3092	8.763	122.943	100000000.000	56.020	56980000.000
3093	8.765	122.977	100000000.000	53.421	19380000.000
3094	8.764	122.967	100000000.000	42.592	-39640000.000
3095	8.763	122.946	100000000.000	34.865	-22480000.000
3096	8.626	122.572	100000000.000	63.329	-55330000.000
3097	8.625	122.577	100000000.000	56.844	182000000.000
3098	8.625	122.589	100000000.000	58.701	18560000.000
3099	8.626	122.595	100000000.000	29.507	-100000000.000
3100	8.349	123.056	100000000.000	69.392	-50350000.000
3101	8.349	123.112	100000000.000	61.548	53170000.000
3102	8.349	123.075	100000000.000	56.167	168500000.000
3103	8.349	123.076	100000000.000	32.712	-100000000.000
3104	8.123	123.124	100000000.000	60.544	77260000.000
3105	8.123	123.128	100000000.000	44.957	26410000.000
3106	8.127	123.102	100000000.000	40.033	-38000000.000
3107	8.162	122.730	100000000.000	55.567	61460000.000
3108	8.165	122.707	100000000.000	40.664	-61550000.000
3109	8.227	121.547	100000000.000	41.044	-100000000.000
3110	8.238	121.339	100000000.000	53.768	69730000.000
3111	8.228	121.882	100000000.000	59.281	27050000.000
3112	8.230	121.463	100000000.000	63.295	-45790000.000
3113	7.938	120.441	100000000.000	53.426	162200000.000
3114	7.938	120.453	100000000.000	31.377	-85570000.000
3115	7.786	119.575	100000000.000	54.709	166400000.000
3116	7.788	119.606	100000000.000	34.904	-100000000.000
3117	7.787	119.570	100000000.000	37.813	-27090000.000
3118	9.357	117.367	100000000.000	65.620	37650000.000
3119	9.357	117.373	100000000.000	55.627	31850000.000
3120	9.357	117.390	100000000.000	36.660	-42670000.000
3121	9.358	117.362	100000000.000	30.888	-23220000.000
3122	8.973	114.808	100000000.000	60.184	46310000.000
3123	8.974	114.796	100000000.000	39.874	-10510000.000
3124	8.973	114.819	100000000.000	29.908	-25430000.000
3125	8.713	112.509	100000000.000	58.704	19860000.000
3126	8.714	112.480	100000000.000	44.952	28390000.000
3127	8.714	112.483	100000000.000	32.001	-13240000.000
3128	8.485	113.481	100000000.000	55.260	21740000.000
3129	8.486	113.552	100000000.000	44.938	38320000.000
3130	8.487	113.554	100000000.000	38.139	-21470000.000
3131	8.458	112.637	100000000.000	64.153	-56060000.000
3132	8.459	112.631	100000000.000	59.059	40410000.000
3133	8.463	112.908	100000000.000	54.557	15650000.000
3134	8.461	112.689	100000000.000	36.497	-19380000.000
3135	7.294	108.514	100000000.000	45.941	95610000.000
3136	7.296	108.519	100000000.000	64.653	8722000.000
3137	7.294	108.519	100000000.000	67.836	-13520000.000
3138	7.083	109.166	100000000.000	66.217	-67540000.000
3139	7.083	109.121	100000000.000	57.588	82930000.000
3140	7.090	109.166	100000000.000	50.555	12880000.000
3141	7.083	109.166	100000000.000	19.692	-17930000.000
3142	6.700	116.487	100000000.000	63.495	67080000.000
3143	6.698	116.437	100000000.000	54.208	11170000.000
3144	6.700	116.494	100000000.000	36.518	-46690000.000
3145	6.701	116.998	100000000.000	41.294	-3380000.000
3146	7.200	116.980	100000000.000	54.260	57800000.000
3147	7.200	116.946	100000000.000	45.904	28450000.000
3148	7.201	116.987	100000000.000	41.088	-42180000.000
3149	7.320	116.645	100000000.000	58.522	18950000.000
3150	7.321	116.669	100000000.000	53.486	63110000.000
3151	7.321	116.668	100000000.000	42.159	-52600000.000
3152	7.320	116.633	100000000.000	28.107	-14920000.000
3153	7.750	116.458	100000000.000	57.494	27470000.000
3154	7.751	116.488	100000000.000	53.589	62080000.000
3155	7.750	116.491	100000000.000	35.111	-56180000.000
3156	7.872	118.086	100000000.000	57.466	137200000.000
3157	7.873	118.135	100000000.000	54.240	18210000.000
3158	7.873	118.097	100000000.000	35.345	-79540000.000
3159	7.873	118.116	100000000.000	16.859	-21310000.000
3160	8.131	118.415	100000000.000	60.793	42850000.000
3161	8.131	118.384	100000000.000	51.992	34490000.000
3162	8.130	118.406	100000000.000	40.440	-38190000.000
3163	8.129	118.441	100000000.000	21.363	-20050000.000
3164	8.110	120.161	100000000.000	56.305	29930000.000
3165	8.109	120.160	100000000.000	53.558	54150000.000
3166	8.110	120.160	100000000.000	35.064	-45640000.000
3167	8.110	120.160	100000000.000	29.621	-25590000.000
3168	8.244	120.786	100000000.000	29.321	-82630000.000
3169	8.248	120.441	100000000.000	32.194	-100000000.000
3170	8.242	120.634	100000000.000	56.245	257300000.000
3171	8.312	120.509	100000000.000	22.046	-21270000.000
3172	8.312	120.557	100000000.000	36.080	-31240000.000
3173	8.312	120.514	100000000.000	49.660	26900000.000
3174	8.312	120.536	100000000.000	59.765	34650000.000
3175	8.414	120.964	100000000.000	31.808	-26430000.000
3176	8.411	120.898	100000000.000	33.724	-41830000.000
3177	8.412	120.914	100000000.000	58.130	72940000.000
3178	8.412	120.952	100000000.000	62.430	27650000.000
3179	8.402	119.587	100000000.000	37.620	-32820000.000
3180	8.402	119.645	100000000.000	55.392	41030000.000
3181	8.402	119.625	100000000.000	60.907	51220000.000
3182	8.402	119.609	100000000.000	69.340	-77270000.000
3183	8.850	119.640	100000000.000	30.886	-39190000.000
3184	8.851	119.617	100000000.000	35.054	-22780000.000
3185	8.850	119.603	100000000.000	53.597	27600000.000
3186	8.850	119.641	100000000.000	65.634	61030000.000
3187	8.838	117.811	100000000.000	42.900	-14040000.000
3188	8.839	117.803	100000000.000	55.543	26680000.000
3189	8.840	117.851	100000000.000	60.905	33520000.000
3190	8.839	117.854	100000000.000	71.243	-41580000.000
3191	9.131	117.628	100000000.000	32.940	-15080000.000
3192	9.130	117.655	100000000.000	60.326	23670000.000
3193	9.128	117.716	100000000.000	61.988	10030000.000
3194	9.131	117.696	100000000.000	62.945	-12120000.000
3195	8.568	118.870	100000000.000	37.787	-27060000.000
3196	8.568	118.865	100000000.000	42.268	-18450000.000
3197	8.568	118.814	100000000.000	54.650	20760000.000
3198	8.569	118.867	100000000.000	61.011	37050000.000
3199	8.466	118.433	100000000.000	58.749	25560000.000
3200	8.467	118.469	100000000.000	52.379	37700000.000
3201	8.468	118.471	100000000.000	42.238	-66140000.000
3202	8.495	116.519	100000000.000	71.259	-20300000.000
3203	8.494	116.541	100000000.000	60.857	18560000.000
3204	8.494	116.552	100000000.000	45.285	74030000.000
3205	8.242	116.048	100000000.000	53.739	69810000.000
3206	8.244	116.032	100000000.000	52.234	47690000.000
3207	8.241	116.068	100000000.000	32.070	-67040000.000
3208	8.241	116.040	100000000.000	23.178	-24160000.000
3209	7.998	116.248	100000000.000	62.511	-14560000.000
3210	7.997	116.139	100000000.000	60.323	12260000.000
3211	7.992	116.651	100000000.000	51.883	88070000.000
3212	7.997	116.133	100000000.000	39.280	-78010000.000
3213	7.990	116.788	100000000.000	55.987	14990000.000

ID	HN	N	CO	CA	Volumen
3214	7.990	116.841	100000000.000	21.382	-60690000.000
3215	7.991	116.664	100000000.000	51.890	108700000.000
3216	8.066	114.491	100000000.000	32.526	-45380000.000
3217	8.066	114.496	100000000.000	56.102	70750000.000
3218	8.065	114.504	100000000.000	61.530	172200000.000
3219	8.066	114.493	100000000.000	69.385	-100000000.000
3220	8.177	114.859	100000000.000	57.951	27200000.000
3221	8.170	114.956	100000000.000	61.540	87950000.000
3222	8.179	115.837	100000000.000	63.363	-3881000.000
3223	8.169	114.935	100000000.000	69.171	-100000000.000
3224	7.925	114.933	100000000.000	45.049	25790000.000
3225	7.925	114.971	100000000.000	59.137	58770000.000
3226	7.926	114.945	100000000.000	64.328	-75490000.000
3227	7.686	113.304	100000000.000	45.698	-17250000.000
3228	7.686	113.302	100000000.000	52.835	23660000.000
3229	7.686	113.326	100000000.000	61.099	63220000.000
3230	7.686	113.337	100000000.000	71.276	-59640000.000
3231	7.801	125.237	100000000.000	19.171	-4356000.000
3232	7.804	125.212	100000000.000	33.583	-10300000.000
3233	7.804	125.159	100000000.000	52.080	6283000.000
3234	7.803	125.186	100000000.000	57.398	15490000.000
3235	8.314	114.981	100000000.000	45.085	63270000.000
3236	8.313	114.988	100000000.000	58.372	55420000.000
3237	8.313	114.991	100000000.000	63.345	-76500000.000
3238	8.317	115.087	100000000.000	70.465	6083000.000
3239	9.140	132.659	100000000.000	33.747	-18830000.000
3240	9.139	132.681	100000000.000	40.412	-5119000.000
3241	9.140	133.095	100000000.000	57.647	50160000.000
3242	9.141	132.658	100000000.000	60.866	7842000.000
3243	9.400	131.964	100000000.000	33.583	-19430000.000
3244	9.397	132.011	100000000.000	62.577	2480000.000
3247	9.400	132.112	100000000.000	67.661	-15530000.000
3248	9.402	131.799	100000000.000	60.523	27730000.000
3249	9.691	132.125	100000000.000	22.656	-38430000.000
3251	9.691	132.125	100000000.000	50.021	60940000.000
3252	9.694	131.942	100000000.000	56.222	11790000.000
3253	9.692	131.687	100000000.000	42.616	-7309000.000
3254	7.760	119.437	100000000.000	54.649	64410000.000
3255	7.748	119.279	100000000.000	45.765	9746000.000
3256	7.749	119.313	100000000.000	42.248	-79860000.000
3257	8.230	120.110	100000000.000	36.801	-14350000.000
3258	8.253	120.216	100000000.000	41.046	-100000000.000
3259	8.251	120.007	100000000.000	51.840	42140000.000
3260	8.253	120.286	100000000.000	54.198	34580000.000
3261	8.253	120.097	100000000.000	62.184	281400000.000
3262	8.458	122.941	100000000.000	32.126	-65150000.000
3263	8.459	122.675	100000000.000	34.927	-49530000.000
3264	8.460	122.992	100000000.000	54.656	51820000.000
3265	8.460	122.709	100000000.000	56.489	120800000.000
3266	8.203	122.210	100000000.000	70.172	-43620000.000
3267	8.200	122.122	100000000.000	60.155	30030000.000
3268	8.203	121.742	100000000.000	58.322	23370000.000
3269	8.189	122.117	100000000.000	33.183	-35030000.000
3270	8.188	122.118	100000000.000	35.082	-61790000.000
3271	8.195	121.987	100000000.000	53.636	65000000.000
3272	7.991	121.676	100000000.000	23.164	-73570000.000
3273	7.991	121.674	100000000.000	52.120	135100000.000
3274	7.991	121.639	100000000.000	57.601	20380000.000
3275	7.991	121.629	100000000.000	66.259	-15750000.000
3276	8.061	120.728	100000000.000	32.547	-100000000.000
3277	8.053	121.131	100000000.000	55.011	136400000.000
3278	8.061	120.692	100000000.000	61.864	274000000.000
3279	9.273	114.671	100000000.000	28.100	-70050000.000
3280	9.273	114.660	100000000.000	31.959	-25470000.000
3281	9.273	114.679	100000000.000	58.494	103900000.000
3282	9.276	114.736	100000000.000	59.958	45060000.000
3283	7.099	116.812	100000000.000	27.418	-58330000.000
3284	7.100	116.763	100000000.000	42.190	-11970000.000
3285	7.099	116.784	100000000.000	53.442	18960000.000
3286	7.098	116.805	100000000.000	57.783	96770000.000
3287	8.412	117.760	100000000.000	59.140	156700000.000
3288	8.412	117.736	100000000.000	31.951	-41670000.000
3289	8.412	117.897	100000000.000	61.907	34310000.000
3290	8.412	117.742	100000000.000	63.266	-100000000.000
3291	8.444	115.820	100000000.000	55.269	77970000.000
3292	8.446	115.802	100000000.000	50.086	59710000.000
3293	8.444	115.827	100000000.000	38.154	-84880000.000
3294	8.444	115.799	100000000.000	22.619	-45160000.000
3295	7.739	106.971	100000000.000	29.888	-13360000.000
3296	7.739	106.955	100000000.000	60.048	22240000.000
3297	7.738	107.070	100000000.000	64.660	32570000.000
3298	7.739	106.966	100000000.000	67.831	-52400000.000
3299	10.854	123.213	100000000.000	54.087	9531000.000
3300	10.854	123.270	100000000.000	36.548	-17220000.000
3301	10.846	122.861	100000000.000	59.309	2392000.000
3302	10.854	123.182	100000000.000	64.672	-2323000.000
3303	8.627	123.732	100000000.000	18.625	-19930000.000
3304	8.627	123.776	100000000.000	45.697	-55160000.000
3305	8.627	123.591	100000000.000	51.251	25220000.000
3306	8.627	123.729	100000000.000	52.752	74600000.000
3307	8.625	122.749	100000000.000	56.796	181500000.000
3308	9.120	122.283	100000000.000	16.770	-13890000.000
3309	9.120	122.314	100000000.000	36.513	-9556000.000
3310	9.121	122.263	100000000.000	54.190	12780000.000
3311	9.122	122.266	100000000.000	63.440	13540000.000
3312	9.413	122.510	100000000.000	42.423	-35830000.000
3313	9.413	122.500	100000000.000	56.110	68540000.000
3314	9.413	122.474	100000000.000	60.204	17580000.000
3315	9.413	122.475	100000000.000	70.201	-21030000.000
3316	6.855	111.148	100000000.000	27.424	-23410000.000
3317	6.855	111.139	100000000.000	57.863	38530000.000
3318	6.854	111.770	100000000.000	58.780	-3758000.000
3319	6.854	111.147	100000000.000	60.937	-22760000.000
3320	6.852	111.770	100000000.000	60.155	3479000.000
3321	6.915	113.643	100000000.000	36.848	6146000.000
3322	6.916	113.724	100000000.000	53.903	-9238000.000
3323	8.935	113.697	100000000.000	42.763	-62020000.000
3324	8.935	113.700	100000000.000	52.254	16560000.000
3325	8.935	113.710	100000000.000	55.562	91950000.000
3326	9.375	118.811	100000000.000	36.255	-11770000.000
3327	9.378	119.118	100000000.000	42.137	-37200000.000
3328	9.380	119.129	100000000.000	59.762	14650000.000
3329	9.378	119.129	100000000.000	58.875	50390000.000
3330	7.313	119.284	100000000.000	51.355	8776000.000
3331	7.314	119.313	100000000.000	59.218	107400000.000
3332	7.316	119.319	100000000.000	38.027	-29850000.000
3333	7.317	119.320	100000000.000	39.599	-16300000.000
3334	5.415	119.622	100000000.000	28.928	-12870000.000
3335	5.413	119.611	100000000.000	30.644	-65330000.000
3336	5.412	119.631	100000000.000	58.484	21500000.000
3337	5.414	119.634	100000000.000	61.250	53900000.000
3338	9.378	127.445	100000000.000	31.088	-21940000.000
3339	7.896	126.005	100000000.000	29.755	-7322000.000
3340	7.155	119.777	100000000.000	58.710	15340000.000
3341	7.158	119.659	100000000.000	38.732	-4375000.000
3342	7.156	119.750	100000000.000	28.959	-8614000.000
3343	11.797	101.896	100000000.000	65.513	-19730000.000
3344	11.797	101.916	100000000.000	52.301	20110000.000
3345	11.797	101.829	100000000.000	48.416	-55930000.000
3346	11.797	101.916	100000000.000	51.380	48130000.000

E.4 HN(CO)CACB (CBCA(CO)NH Typ)

ID	HN	N	CO	CA	Volumen
2001	9.144	133.255	100000000.000	40.137	1.000
2002	9.142	133.276	100000000.000	60.683	1.000
2003	9.695	131.857	100000000.000	42.329	1.000
2004	9.695	131.879	100000000.000	55.538	1.000
2005	9.402	131.703	100000000.000	62.043	1.000
2006	9.403	131.699	100000000.000	67.785	1.000
2007	8.261	129.750	100000000.000	55.625	1.000
2008	8.261	129.748	100000000.000	32.425	1.000
2009	8.517	129.184	100000000.000	60.940	1.000
2010	8.518	129.180	100000000.000	30.366	1.000
2011	8.321	129.063	100000000.000	61.283	1.000
2012	8.329	129.052	100000000.000	68.815	1.000
2013	8.997	128.872	100000000.000	61.028	1.000
2014	8.999	128.850	100000000.000	37.418	1.000
2015	7.852	129.197	100000000.000	55.747	1.000
2016	7.855	129.235	100000000.000	31.936	1.000
2017	7.805	129.199	100000000.000	41.887	1.000
2018	7.801	129.233	100000000.000	54.306	1.000
2019	9.606	128.183	100000000.000	59.803	1.000
2020	9.607	128.193	100000000.000	34.266	1.000
2021	8.032	128.031	100000000.000	61.764	1.000
2022	8.030	128.108	100000000.000	69.030	1.000
2023	9.212	127.711	100000000.000	29.763	1.000
2024	9.212	127.696	100000000.000	54.565	1.000
2025	9.381	127.441	100000000.000	41.110	1.000
2026	9.380	127.375	100000000.000	54.423	1.000
2027	8.509	127.311	100000000.000	44.550	1.000
2028	8.508	127.284	100000000.000	52.828	1.000
2029	8.605	127.048	100000000.000	39.073	1.000
2030	8.601	127.061	100000000.000	59.629	1.000
2031	7.705	126.759	100000000.000	39.998	1.000
2032	7.705	126.741	100000000.000	58.032	1.000
2033	9.307	126.329	100000000.000	30.212	1.000
2034	9.304	126.357	100000000.000	55.259	1.000
2035	8.667	126.382	100000000.000	56.479	1.000
2036	8.666	126.396	100000000.000	31.653	1.000
2037	7.780	126.304	100000000.000	35.305	1.000
2038	7.780	126.296	100000000.000	59.677	1.000
2039	9.381	126.058	100000000.000	60.540	1.000
2040	9.381	126.120	100000000.000	69.254	1.000
2041	8.827	126.030	100000000.000	61.255	1.000
2042	8.827	126.043	100000000.000	39.262	1.000
2043	8.361	125.920	100000000.000	57.232	1.000
2044	8.358	125.951	100000000.000	62.547	1.000
2045	7.894	125.880	100000000.000	56.277	1.000
2046	7.897	125.978	100000000.000	29.165	1.000
2047	8.695	125.478	100000000.000	60.963	1.000
2048	8.695	125.467	100000000.000	70.756	1.000
2049	8.185	125.105	100000000.000	51.510	1.000
2050	8.169	125.184	100000000.000	36.032	1.000
2052	8.452	125.174	100000000.000	33.204	1.000
2053	8.453	125.167	100000000.000	61.262	1.000
2054	9.143	124.952	100000000.000	53.416	1.000
2055	9.143	124.955	100000000.000	35.035	1.000
2056	9.079	124.738	100000000.000	60.211	1.000
2057	9.082	124.750	100000000.000	33.863	1.000
2058	8.834	124.467	100000000.000	45.032	1.000
2059	8.551	124.440	100000000.000	61.173	1.000
2060	8.149	124.217	100000000.000	41.348	1.000
2061	8.154	124.231	100000000.000	53.891	1.000
2062	7.391	124.247	100000000.000	58.534	1.000
2063	7.397	124.217	100000000.000	61.855	1.000
2064	8.628	123.644	100000000.000	18.235	1.000
2065	8.630	123.633	100000000.000	51.296	1.000
2066	8.448	123.912	100000000.000	32.496	1.000
2067	8.450	123.833	100000000.000	42.526	1.000
2068	8.446	123.815	100000000.000	50.837	1.000
2069	8.447	123.892	100000000.000	61.357	1.000
2070	8.252	123.843	100000000.000	55.831	1.000
2071	8.245	123.867	100000000.000	33.292	1.000
2072	8.128	123.015	100000000.000	44.271	1.000
2073	10.853	123.171	100000000.000	58.703	1.000
2074	10.851	123.141	100000000.000	64.129	1.000
2075	8.765	122.844	100000000.000	34.639	1.000
2076	8.766	122.836	100000000.000	53.459	1.000
2077	9.414	122.405	100000000.000	60.009	1.000
2078	9.417	122.425	100000000.000	69.652	1.000
2079	8.169	122.496	100000000.000	58.141	1.000
2080	8.169	122.552	100000000.000	63.971	1.000
2081	7.598	122.500	100000000.000	53.003	1.000
2082	7.597	122.468	100000000.000	40.799	1.000
2083	8.474	122.673	100000000.000	69.070	1.000
2084	8.472	122.636	100000000.000	60.796	1.000
2085	8.464	122.640	100000000.000	54.316	1.000
2086	8.464	122.619	100000000.000	34.707	1.000
2087	8.618	122.406	100000000.000	58.533	1.000
2088	8.619	122.392	100000000.000	62.841	1.000
2089	8.917	122.341	100000000.000	52.371	1.000
2090	8.915	122.355	100000000.000	45.264	1.000
2091	8.189	122.100	100000000.000	32.516	1.000
2092	8.189	122.110	100000000.000	56.905	1.000
2093	9.122	122.213	100000000.000	62.989	1.000
2094	9.123	122.182	100000000.000	36.394	1.000
2095	9.304	122.020	100000000.000	36.535	1.000
2096	9.305	122.020	100000000.000	54.524	1.000
2097	8.981	121.967	100000000.000	55.334	1.000
2098	8.980	121.965	100000000.000	41.959	1.000
2099	8.303	121.982	100000000.000	60.973	1.000
2100	8.298	121.980	100000000.000	68.675	1.000
2101	8.042	121.930	100000000.000	55.728	1.000
2102	8.041	121.885	100000000.000	32.460	1.000
2103	8.288	122.005	100000000.000	56.325	1.000
2104	8.308	121.940	100000000.000	49.840	1.000
2105	8.537	122.060	100000000.000	58.083	1.000
2106	8.541	122.017	100000000.000	63.036	1.000
2107	7.995	121.526	100000000.000	66.106	1.000
2108	7.994	121.580	100000000.000	56.670	1.000
2109	7.598	121.058	100000000.000	53.148	1.000
2110	7.598	121.063	100000000.000	40.770	1.000
2111	9.752	121.077	100000000.000	53.264	1.000
2112	9.750	121.095	100000000.000	31.649	1.000
2113	8.230	121.497	100000000.000	58.378	1.000
2114	8.229	121.500	100000000.000	62.999	1.000
2115	8.375	122.210	100000000.000	61.184	1.000
2116	8.374	122.157	100000000.000	68.840	1.000
2117	8.114	121.026	100000000.000	55.234	1.000
2118	8.119	121.021	100000000.000	31.936	1.000
2119	8.296	120.940	100000000.000	43.022	1.000
2121	8.232	120.982	100000000.000	29.529	1.000
2122	8.231	121.065	100000000.000	57.980	1.000
2123	8.412	120.902	100000000.000	62.058	1.000
2124	8.413	120.892	100000000.000	31.324	1.000
2125	8.309	120.523	100000000.000	21.376	1.000
2126	8.315	120.495	100000000.000	49.644	1.000
2127	7.940	120.470	100000000.000	60.745	1.000
2128	7.970	120.678	100000000.000	44.789	1.000
2129	8.065	120.789	100000000.000	32.417	1.000
2130	8.065	120.791	100000000.000	55.240	1.000
2131	8.066	120.663	100000000.000	61.589	1.000
2132	8.059	120.609	100000000.000	51.851	1.000
2133	7.145	119.854	100000000.000	38.490	1.000
2134	7.142	119.833	100000000.000	59.215	1.000
2135	7.607	119.814	100000000.000	53.093	1.000
2136	7.603	119.801	100000000.000	40.722	1.000
2137	8.111	120.123	100000000.000	55.691	1.000
2138	8.112	120.096	100000000.000	29.686	1.000
2139	8.065	120.323	100000000.000	55.252	1.000
2140	8.072	120.190	100000000.000	29.813	1.000
2141	8.226	120.069	100000000.000	36.355	1.000
2142	8.244	120.008	100000000.000	40.828	1.000
2143	8.224	119.982	100000000.000	55.842	1.000
2144	8.233	120.084	100000000.000	62.515	1.000
2145	8.238	119.723	100000000.000	18.394	1.000
2146	8.243	119.970	100000000.000	53.532	1.000
2147	7.787	119.693	100000000.000	37.490	1.000
2148	7.787	119.684	100000000.000	54.141	1.000
2149	7.746	119.404	100000000.000	44.147	1.000
2150	8.404	119.555	100000000.000	37.404	1.000
2151	8.404	119.575	100000000.000	55.437	1.000
2152	8.848	119.608	100000000.000	53.451	1.000
2153	8.848	119.603	100000000.000	35.163	1.000
2154	7.313	119.305	100000000.000	51.308	1.000
2155	7.313	119.302	100000000.000	39.076	1.000
2156	8.570	118.825	100000000.000	42.205	1.000
2157	8.570	118.817	100000000.000	54.336	1.000
2158	5.408	119.680	100000000.000	28.678	1.000
2159	5.408	119.704	100000000.000	58.317	1.000
2160	9.378	119.063	100000000.000	59.801	1.000
2161	9.378	119.063	100000000.000	35.648	1.000
2162	8.473	118.437	100000000.000	58.047	1.000
2163	8.467	118.496	100000000.000	53.146	1.000
2164	8.476	118.494	100000000.000	42.029	1.000
2165	8.133	118.377	100000000.000	51.486	1.000
2166	8.131	118.318	100000000.000	21.447	1.000
2167	7.873	118.023	100000000.000	54.041	1.000
2168	7.873	118.055	100000000.000	16.514	1.000
2169	8.411	117.824	100000000.000	31.403	1.000
2170	8.411	117.811	100000000.000	61.862	1.000
2171	8.480	117.899	100000000.000	55.755	1.000
2172	8.475	117.938	100000000.000	29.238	1.000
2173	8.839	117.881	100000000.000	42.488	1.000
2174	8.841	117.826	100000000.000	55.440	1.000
2175	9.139	117.660	100000000.000	61.867	1.000
2176	9.139	117.577	100000000.000	32.436	1.000
2177	9.359	117.426	100000000.000	65.212	1.000
2178	9.363	117.301	100000000.000	30.841	1.000
2179	7.199	116.963	100000000.000	45.435	1.000
2180	7.320	116.690	100000000.000	28.353	1.000
2181	7.320	116.673	100000000.000	58.417	1.000
2182	6.703	116.546	100000000.000	54.168	1.000
2183	6.701	116.542	100000000.000	40.734	1.000
2184	7.098	116.790	100000000.000	52.729	1.000
2185	7.098	116.810	100000000.000	41.933	1.000
2186	7.756	116.536	100000000.000	35.130	1.000
2187	7.756	116.596	100000000.000	57.002	1.000
2188	7.993	116.834	100000000.000	31.020	1.000
2189	7.994	116.862	100000000.000	55.737	1.000
2190	8.003	116.219	100000000.000	59.904	1.000
2191	8.406	116.882	100000000.000	55.432	1.000
2192	8.452	117.011	100000000.000	32.478	1.000
2193	8.502	116.743	100000000.000	60.790	1.000
2194	8.496	116.714	100000000.000	70.750	1.000
2195	8.237	116.096	100000000.000	52.295	1.000
2196	8.238	116.104	100000000.000	22.645	1.000
2197	8.447	115.862	100000000.000	22.647	1.000
2198	8.447	115.852	100000000.000	50.060	1.000
2199	8.349	115.779	100000000.000	57.494	1.000
2200	8.348	115.776	100000000.000	62.950	1.000
2201	8.446	115.837	100000000.000	55.190	1.000
2202	8.432	115.820	100000000.000	33.847	1.000
2203	7.931	115.051	100000000.000	44.661	1.000
2204	8.290	115.029	100000000.000	44.190	1.000
2205	8.167	115.055	100000000.000	57.653	1.000
2206	8.173	115.016	100000000.000	63.374	1.000
2207	9.267	114.858	100000000.000	31.379	1.000

ID	HN	N	CO	CA	Volumen
2208	9.267	114.867	100000000.000	59.676	1.000
2209	8.968	114.920	100000000.000	59.836	1.000
2210	8.969	114.842	100000000.000	39.857	1.000
2211	6.781	114.281	100000000.000	40.384	1.000
2212	6.780	114.298	100000000.000	55.128	1.000
2213	6.920	113.820	100000000.000	36.501	1.000
2214	6.921	113.688	100000000.000	53.718	1.000
2215	8.482	113.609	100000000.000	55.311	1.000
2216	8.481	113.629	100000000.000	37.640	1.000
2217	8.940	113.740	100000000.000	42.040	1.000
2218	8.938	113.738	100000000.000	52.400	1.000
2219	7.689	113.515	100000000.000	45.263	1.000
2220	7.683	113.499	100000000.000	52.460	1.000
2221	7.647	113.615	100000000.000	53.450	1.000
2222	7.645	113.732	100000000.000	36.453	1.000
2223	7.592	114.027	100000000.000	53.632	1.000
2224	7.591	113.952	100000000.000	36.780	1.000
2225	8.067	114.694	100000000.000	55.701	1.000
2226	8.069	114.585	100000000.000	32.525	1.000
2227	8.478	112.699	100000000.000	36.540	1.000
2228	8.475	112.656	100000000.000	54.367	1.000
2229	8.716	112.617	100000000.000	31.820	1.000
2230	8.717	112.613	100000000.000	58.315	1.000
2231	6.787	112.935	100000000.000	40.823	1.000
2232	6.781	112.884	100000000.000	55.028	1.000
2233	7.632	112.598	100000000.000	36.479	1.000
2234	7.631	112.594	100000000.000	55.539	1.000
2235	6.962	112.624	100000000.000	55.316	1.000
2236	6.961	112.576	100000000.000	36.472	1.000
2237	7.460	111.799	100000000.000	37.983	1.000
2238	7.460	111.792	100000000.000	54.091	1.000
2239	6.781	111.766	100000000.000	38.192	1.000
2240	6.805	111.809	100000000.000	54.184	1.000
2241	6.858	111.328	100000000.000	57.575	1.000
2242	6.854	111.388	100000000.000	26.719	1.000
2243	6.772	110.399	100000000.000	38.066	1.000
2244	6.772	110.432	100000000.000	55.194	1.000
2245	7.462	110.496	100000000.000	54.714	1.000
2246	7.461	110.455	100000000.000	38.052	1.000
2247	7.083	109.144	100000000.000	19.485	1.000
2248	7.083	109.198	100000000.000	49.855	1.000
2250	7.296	108.406	100000000.000	55.373	1.000
2251	7.295	108.402	100000000.000	63.572	1.000
2252	8.130	108.319	100000000.000	55.350	1.000
2253	8.138	108.313	100000000.000	18.032	1.000
2254	7.742	107.265	100000000.000	29.709	1.000
2255	7.739	107.254	100000000.000	59.989	1.000
2256	8.600	106.470	100000000.000	38.441	1.000
2257	8.600	106.460	100000000.000	55.668	1.000
2258	7.295	108.398	100000000.000	67.889	1.000
2259	9.712	119.860	100000000.000	33.485	1.000
2260	9.714	119.876	100000000.000	60.301	1.000
2261	8.634	123.657	100000000.000	21.181	0.000
2262	8.627	123.653	100000000.000	49.004	0.000

Anhang F Parameter der Optimierungsprogramme

F.1 Netzbildung

`distanzen` `[positive Zahl]` `[abhängig vom Isotop]`

CA/CB/CO	0.5 – 1.0	ppm
H	0.02–0.05	ppm
NH	0.5 – 1.0	ppm

F.2 Problemstellung

`protein` `[dateiname]` `"Protein"`

enthält die Proteindefinition (Länge,Sequenz, ...)

`pool` `[dateiname]` `"Poolfile"`

Liste der Peaklisten und NMR−Experiment−Definitionen.

`netfile` `[dateiname]` `"netfile.2_0"`

Name der Peaknet Datei, bestimmt die Nachbarschaftsbeziehung zwischen den Peaks.

`cache` `[verzeichnisname]` `"cache"`

Das Verzeichnis, in dem die Context Listen für peaks und Aminosäuren abgelegt werden.

`indepFile` `[dateiname]` `"OptimizerInfo.<<extension>>"`

Die Definition des Cache−Ausschnittes der geladen wird.

`optiFileName` `[dateiname]` `"RandomDump.#<arenasize>.empty"`

Der Anfangszustand des Optimierers.

`dumpFile` `[dateiname]` `"RandomDump.<<extension>>"`

Der Endzustand des Optimierers. Das Ergebnis! Kann als Anfangszustand verwendet werden.

F.3 Parameter des RANDOM Algorithmus

Abbruchbedingung:

`maxLoops` `[loop]`

Die Anzahl der Zyklen, die RANDOM maximal durchlaufen darf. Teil der Abbruchbedingung.

`maxTime` `[min]`

Reale Zeit, die RANDOM maximal durchlaufen darf. Teil der Abbruchbedingung.

`maxIdleLoops` `[loop]`

Maximale Anzahl der Zyklen, in denen keine Verbesserung erreicht wird, bevor RANDOM abbricht. Teil der Abbruchbedingung.

Konfiguration:

`qThreshold` `[0.0 – 1.0]` `[0.9]`

Faktor für die Qualitätsschwelle.

`qThresholdDecayRate` `[0.0 – 1.0]` `[0.0005]`

Zerfallsrate für `qThreshold`.

`minTTL` `[loops]` `[2]`

minimale TTL (Time to Live). Zustände mit einer Qualität unterhalb der Qualitätsschwelle werden nicht ersetzt, wenn sie $\leq$ minTTL Zykeln alt sind.

`maxTTL` `[loops]` `[3]`

maximale TTL (Time to Live). Zustände mit einer TTL $\geq$ maxTTL Zykeln werden ersetzt, auch wenn sie eine Qualität oberhalb der Qualitätsschwelle haben.

`randArenaSize` `[anzahl]` `[4]`

Anzahl der gleichzeitig gerechneten Zustände **S** im RANDOM−Algorithmus.

`profile` `[dateiname]` `"profil.mix1_2"`

Wahrscheinlichkeitstabelle für die einzelnen Aktionen.

`random` `[zahl | "time" ]` `"time"`

Startwert für den Pseudorandomzahlengenerator. Mit "time" wird die Systemzeit in Sekunden als Startwert verwendet.

Geschwindigkeit / Speicher:

`calc` `[bezeichner]` `"as_link"`

Wählt den Berechnungsalgorithmus aus. Bestimmt wieviel Speicher der Algorithmus verbraucht.

`none`	kein Berechnungscache.
`as`	Berechnungscache für AS−Berechnung.
`as_link`	Berechnungscache für AS− und Link−Berechnung.

F.4 Genetischer Optimierer

Konfiguration:

`optiOp` `[bezeichner]` `[Gs2p2]`

Der Name des Crossover Operators.

`crossOverPct` `[0.0–1.0]` `[0.15]`

Parameter des Crossover Operators. Anteil der Zustandsdifferenz, der in den neuen Zustand übertragen wird.

`genArenaSize` `[anzahl]` `[8]`

Anzahl der parallel berechneten Individuen.

`seedPoolSize` `[anzahl]` `[4]`

Anzahl der Eltern.

`survivingSeeds` `[anzahl]` `[4]`

Anzahl der garantiert überlebenden Individuen.

`genQThreshold` `[0.0–1.0]` `[0.9]`

Mindestqualitätfaktor, relativ zum aktuell besten Individuen.

`genQThresholdDecayRate` `[0.0–1.0]` `[0.05]`

Analog zu `qThresholdDecayRate.`

`genCrossoverThreshold` `[0.0–+∞]` `[0.5]`

Mindestrate die die gleitende Crossover−Rate überschreiten muß. Wird die Schwelle unterschritten, wird `genQThreshold` mittels `genQThresholdDecayRate`angehoben. (Formel: 2−15)

`genMaxEqualLoops` `[loops]` `[5]`

Maximale Zyklenanzahl, wenn alle Individuen die gleiche Qualität haben.

`genMaxIdleLoops` `[loops]` `[15]`

Analog zu `maxIdleLoops`.

Suboptimierer (RANDOM als Mutationsalgorithmus):

`subMaxTime` `[min]` `[1]`

`subMaxLoops` `[loops]` `[4000]`

`subMaxIdleLoops` `[loops]` `[4001]`

Analog zu den gleichnamigen Parametern ohne sub−Präfix. Diese Parameter überschreiben die Voreinstellungen für RANDOM.

Nachgeschalteter Optimierer (RANDOM):

`finalRunTime` `[min]` `[−]`

`finalMaxLoops` `[loops]` `[20000]`

`finalMaxIdle` `[loops]` `[5000]`

Analog zu den gleichnamigen Parametern ohne sub−Präfix. Diese Parameter überschreiben die Voreinstellungen für RANDOM.

Anhang G Verwendete Symbole und Abkürzungen

G.1 Abkürzungen

NMR	Nuclear Magnetic Resonance.
HNCO	NMR Experimente
HNCA	
CBCANH	
CBCA(CO)NH	
ACX	AS−Context, AS−Protospinsystem.
LCX	Link−Context, Link−Protospinsystem.
CX	Context.
Context	Context, Repräsentation eines Protospinsystems im Programm.

G.2 Symbole

N	Node, Knoten im PeakNet, repräsentiert einen Peak.
E K e_i	Edge, ungerichtete Kante zwischen zwei Knoten im PeakNet mit Qualität.
E	Element im Zustand.
a_p	Anzahl der belegbaren Positionen im Zustand.
a_k	Anzahl der verteilbaren Elemente.
P	die Menge aller Peaks.
$\boldsymbol{S}_i$	Zustand, eine Lösung im Lösungsraum
$\boldsymbol{S}_b$	die Menge aller Zustände, die genau b Elemente enthalten.
$\boldsymbol{S}_p$	der Lösungsraum mit Zuständen mit p Positionen.
$\boldsymbol{S}$	State, der Zustand im Lösungsraum, das Individuum im Optimierer.
$\boldsymbol{S}_{x,p}$	State mit x Elementen und p Positionen.
$\boldsymbol{S}_{NULL}$	der leere Zustand. Einziges Element in $\boldsymbol{S}_0$.
$\boldsymbol{S}_{NULL}$	der leere Zustand. Einziges Element in $\boldsymbol{M_0}$.
$\boldsymbol{S}_b^{\subset}$	Menge aller $\boldsymbol{S}_{b,p}$ mit *b* belegten Positionen, die die gleichen Elemente enthalten.
$\boldsymbol{S}_{b,norm}^{\subset}$	Das erste Element der Menge $\boldsymbol{S}_b^{\subset}$, jedes Element der Menge kann durch Normierung in $\boldsymbol{S}_{b,norm}^{\subset}$ überführt werden.
N	Nachbarschaft
Mb	Menge aller $S_{b,p}$ mit b belegen Positionen ohne Wiederholung eines Elements.
$T^{op}_{\boldsymbol{S}1,\,\boldsymbol{S}2}$	Operation, die $\boldsymbol{S}_1$ in $\boldsymbol{S}_2$ umwandelt. $\boldsymbol{S}_1 \in M_i \wedge \boldsymbol{S}_2 \in M_j$
$T^{op}_{S1,S2}$	Transition, die Operation op, die Zustand S1 in S2 umwandelt.
T^{+}	Transition, die Operation op, die Zustand S1, durch hinzufügen eines Elements, in S2 umwandelt. Die inverse Operation zu T^{-} .
T^{-}	Transition, die Operation op die Zustand S1, durch Entfernen eines Elements, in S2 umwandelt. Die inverse Operation zu T^{+} .
T^{NULL}	Null−Transition, die Operation die Zustand S1 unverändert läßt.
a_{amino}	Anzahl der Aminosäuren im Protein.
a_{AS}	Anzahl der AS−Protospinsysteme
a_{Link}	Anzahl der Link−Protospinsysteme
$a_{ca,xx}$	Anzahl der Kanten mit CA Frequenz im der aktuellen Knoten
$a_{cb,xx}$	Anzahl der Kanten mit CB Frequenz im der aktuellen Knoten
a_{Pro}	Anzahl der Proline im Protein. (analog für alle anderen Aminosäuren)
$a_{Pro,innen}$	Anzahl der Proline die nicht am Rand des Proteins liegen.

$a_{AS,max}$	theror. Anzahl der AS−Protospinsysteme.
$a_{Pro,innen}$	Anzahl der Proline die nicht am Rand des Proteins liegen.
Q_{AS}	Qualität eines ACX ohne Bewertung der Aminosäureposition. (analog für LCX)
Q_{AS+Typ}	Qualität eines ACX mit Bewertung der Aminosäureposition.
$\mathrm{O}(\ )$	Komplexitatsfunktion
$\mathrm{P}_{i,j}(T^{op})$	Wahrscheinlichkeit der Akzeptanz der Aktion T^{op} im kombinatorischen Optimierer
$\mathrm{POP}(t)$	Population der Individuen im genetischen Optimierer
V_1 , V_2	Volumen des ersten Peaks, ..

Anhang H Kombinatorik

H.1 Permutationen und Variation

Verteile *p* unterschiedliche Elemente aus einer *n*−elementigen Menge auf *i* Positionen

	ohne Wiederholung	**mit Wiederholung**
Permutation **i = p = n**	$A_i = i!$	$A_i = \frac{i!}{\prod g_{k!}!}$
Variation **i= p n != p**	$V_{n,i} = \frac{n!}{(n-i)!} = n*(n-1)*(n-2)*...*(n-i+1)$	$\overline{V}_{n,i} = n^i$

g_k ist die Anzahl der Elemente in der *k* te Gruppe von Elementen, die man nicht von einander unterscheiden kann.

H.2 Kombinationen

Wieviel unterschiedliche Kombinationen (ohne Beachtung der Reihenfolge) von *i* aus *n* unter−schiedlichen Elementen gibt es?

ohne Wiederholung.	wie vorher, aber mit Wiederholung.
$C_{n,i} = \frac{n!}{i!(n-i)!} = \binom{n}{i}$	$\bar{C}_{n,i} = \binom{n+i-1}{i}$

n = Anzahl der Elemente i = Positionen	Permutationen Reihenfolge wesentlich!	Kombinationen Reihenfolge unwesentlich.
ohne Wiederholung	$V_{n,i} = \frac{n!}{(n-i)!}$	$C_{n,i} = \frac{n!}{i!(n-i)!} = \binom{n}{i}$
mit Wiederholung (doppelte Elemente)	$\overline{V}_{n,i} = n^i$	$\bar{C}_{n,i} = \binom{n+i-1}{i}$

Auf wieviel verschiedene Arten kann man *i* von *k* verschiedenen Elementen auf *p* Positionen verteilen?

Nebenbedingungen: *p>=i* und die Reihenfolge wird beachtet.

$$a(M_i) = \frac{k!}{(k-i)!} * \frac{p!}{i!(p-i)!}$$

p ist die Anzahl der *möglichen* Positionen.

i ist die Anzahl der *belegten* Positionen und damit gleich Anzahl der ausgewählten Elemente.

n ist die Anzahl der Elemente

Quelle: [Zachmann84, S.27++]

Anhang I Definitionen der Graphen Theorie

Aus [Reinelt94, S.4+, Bronstein91, S.157++]

Definitionen: Ein *ungerichteter Graph* **G** = (V, E) besteht aus einem finiten Set *Knoten* **V** und einem finiten Set Kanten **E**. Jede *Kante* e hat zwei Endknoten u,v aus V und wird als e={u,v} notiert. Ist der Graph ungerichtet, kann man e={u,v} nicht von e={v,u} unterscheiden, andernfalls ist der Graph *gerichtet*. Ein gerichteter Graph **D** ist *vollständig,* wenn es für jede Kante e={u,v} eine Kante e={v,u} gibt.

Zwei Knoten sind *benachbart,* wenn es mindestens eine Kante gibt, die sie verknüpft. Wenn für eine Folge benachbarter Knoten $e_{1,} e_{2,} \ldots e_{k}$ gilt $\{e_{i,} e_{i+1}\} \in E$ mit i = 1,...k−1, dann ist die zugehörige Kantenfolge ein *Weg.* Die Kantenanzahl k−1 ist seine *Weglänge,* e_1 der *Anfangs*− und e_2 der *Endknoten*. Ein geschlossener Weg, bei dem der Anfangsknoten gleich dem Endknoten ist, ist ein *Rundweg (Zyklus)*. Der *Grad* eines Knotens ist die Anzahl der Kanten die an diesem Knoten ankommen und von diesem ausgehen.

Ein Graph ist *zusammenhängend*, wenn jedes Eckenpaar aus V über einen Weg in E verbunden ist. In einem azyklischen Graph gibt es keine Zyklen, ein solcher Graph ist ein *Baum*.

Ein *gewichteter* Graph G besteht aus einem Knotenset V und einem Kantenset E, indem jeder Kante ein Gewicht **w** zugeordnet ist. Das Gewicht eines Weges ist die Summe der Gewichte entlang des Weges. Diese Summe wird auch die *Länge des Weges* genannt. Zusätzlich können auch den Knoten Gewichte zugeordnet sein. Das Gewicht des Weges enthält dann auch die Knotengewichte. Der Weg mit der niedrigsten Summe ist der *kürzeste Weg*.

Anhang J Notwendige Computerresourcen

J.1 Programmierumgebung

Das Programm wurde in einer UNIX Umgebung auf Linux (Kernel 2.2.14, SuSE 5.2 − 6.4) entwickelt. Die Rechnungen der Parametrisierung wurden auf einem Intel PII 400 mit 256 MB Speicher durchgeführt.

Voraussetzung für die Erzeugung der Programme ist eine vollständige GNU Programmierumgebung aus GCC C++ (größer v 2.8), GDB, libg++, Perl 5.005 mit TK Perl v8.0, CVS, flex, bison, GNU make, nm, ar und den üblichen UNIX Werkzeugen. Für die Shellscripte im Projekt werden außerdem tar, bzip2 und sed benötigt.

Das Verzeichnis mit den *shared libraries* des ASSIGN Quellbaumes (**`${ROOTDIR}/lib`**) muß in den Pfad für die *shared libraries* des Betriebssystems aufgenommen werden!

Anhang K Verzeichnisstrukturen

K.1 Verzeichnisse und Dateien eines Zuweisungsprojektes

Verzeichnis / Datei	Erklärung
../Trigger	das Hauptverzeichnis des Projekts "Trigger"
../Trigger/bin	Link zu den Programmen –> $ASS/bin
../Trigger/cache /.GlyInfo	ein Cache Basisverzeichnis die gesammelte Glycin Information
/.globalPage /.maxNames /.selectedEntries	Dateien für viewCache.pl
../Trigger/cache /p0001 /p0002 /... /p0203	Peak Cache je ein Verzeichnis pro Basispeak (d.h. Peak im HNCO)
../Trigger/cache/p0023/conditions /listCA.complete /.filtered.19991119-01	die lokalen Bedingungen für BP–23 eine Protospinsystemliste für die lokalen Bedingungen eine gefilterte Protospinsystemliste
/as001 /as002 /... /as113	Aminosäure Cache. je ein Verzeichnis pro Aminosäure im Protein.
../Trigger/pattern	die Protospinsystemmuster $ASS/pattern
../Trigger/orginale	enthält die Peaklisten
/orginale/**Poolfile.pool**	faßt die Peaklisten zusammen.
/orginale/**hnco.plf**	die Definition einer Peakliste mit den Teilen .xpk, .vol, .ppm, .list100, .axis
../Trigger/sequences	die Optimierungsergebnisse
../Trigger/sequences/macros	die Konfigurationsmakros $ASS/Assign/Optimi–zerScripts
../Trigger/sequences/<<optimierungsstufe>> /best	die Optimierungsergebnisse einer Stufe Verzeichnis für die Referenzzuweisung bei bekannter theoretischer Zuweisung
../Trigger/profile	die Aktionsprofile $ASS/profile
/**.profile.local** /**setEnv**	setzt die Umgebungsvariablen für das Projekt in die–sem Verzeichnis, sollte von der Shell ge*sourced* wer–den
/**Protein**	die Proteindefinition
/netfile.xxxx	ein Netzfile (PeakNet)
/**NetLog**	das Logfile für dumpNode
/**RandomDump.32.empty**	eine leere Zuweisung mit 32 Individuen
/**asQFactor.noHisTag**	die Gewichtung der Aminosäurepositionen
/NetDB.<<ext>>	eine Perl Datenbank

*Tabelle K–1 Dateisystemstruktur eines Projektes anhand von "Trigger". **Fett** gesetzte Dateien müssen von Hand erzeugt werden. Blau markiert Verzeichnisse, die mit dem $ASS Projekt geteilt werden.*

K.2 Der Quellcode

Die Programme des ASSIGN Projekts befinden sich in einem gemeinsamen Verzeichnisbaum (`.../assign`). Der Pfad zu diesem Verzeichnis muß in der Umgebungsvariablen $ASS gespeichert sein. Der Quellcode ist in verschiedene Bibliotheken in einzelnen Unterverzeichnissen aufgeteilt. Jedes Bibliotheksverzeichnis enthält den Quellcode für eine Bibliothek und für Programme, die die Bibliothek nutzen. Die Bibliotheksverzeichnisse haben eine einheitliche Struktur und verwenden gemeinsame Makefiles aus dem zentralen Quellverzeichnis. Die Objektdateien jeder Bibliothek werden in einem einzelnen Verzeichnis des Build−Verzeichnisses gesammelt, während die der lokalen Programme im Quellverzeichnis erzeugt werden.

Verzeichnis / Datei	**Erklärung**
`.../assign`	das Hauptverzeichnis des Quellcodes enthält die zentralen Makefiles und alle Bibliotheksverzeichnisse. `${ROOTDIR}` und `$ASS`.
`${ROOTDIR}/bin`	`collectBin.pl` erzeugt hier einen Symlink für jedes ausführbare Programm.
`${ROOTDIR}/lib`	enthält alle Bibliotheken. Die `lib` Verzeichnisse in den Bibliotheksverzeichnissen zeigen hierher.
`${ROOTDIR}/pattern`	die Protospinsystemmuster.
`${ROOTDIR}/profile`	die Aktionsprofile.
`${ROOTDIR}/Tools`	Werkzeuge für die Programmerzeugung.
`${ROOTDIR}/<<bibliothek>>`	ein Bibliotheksverzeichnis. Enthält auch die zugehörigen Programme.
`${ROOTDIR}/AssignNet/OptimizerScripts`	die Makros für die Optimiererstartscripte. (`../macros`)
`${ROOTDIR}/Viewer/*`	die Perl−Werkzeuge und −Programme, auch Bibliotheksverzeichnis für die Perl−Programme.

Tabelle K−2 Struktur der Verzeichnisse des Quellbaumes. **Blau** *markiert Verzeichnisse, die mit den Projekten geteilt werden.* **`${ROOTDIR}/lib`** *muß in den Pfad für die* shared libs *des Betriebssystems aufgenommen werden!*

Verzeichnis / Bibliothek	Erklärung
${ROOTDIR}/Base	Basisklassen, zum Teil aus libg++, Array, Map, ...
${ROOTDIR}/Basic	Basisklassen, die auf Base aufbauen. Peak, Axis, Accounting, ...
${ROOTDIR}/Lex	der Lexer (kaum benutzt, nur für Protein)
${ROOTDIR}/Fuzzy	die Fuzzy − Zahlen Klasse (nur Fuzzy.h benutzt)
${ROOTDIR}/LongOpt	der Kommandozeilenparameterparser
${ROOTDIR}/Obj	Debugging und Allocator−Klassen
${ROOTDIR}/Peak	Klassen Peak, Peakliste, Peakpool
${ROOTDIR}/PeakNet	Klassen PeakNet, Condition.., LinFunc,
${ROOTDIR}/BBNet	Programme
${ROOTDIR}/Gr	Modul rund um die Grzesiek−Aminosäurebewertungsfunktionen.
${ROOTDIR}/Matrix	Routinen um Felix .mat Dateien zu lesen.
${ROOTDIR}/Expression	die Bewertung der Protospinsysteme. Klassen Expression ...
${ROOTDIR}/AssignNet	alles für den Optimierer

Tabelle K−3 Die Bibliotheksverzeichnisse in ansteigender Komplexität der Bibliotheken.

Verzeichnis / Datei	Erklärung
${ROOTDIR}/AssignNet	das Quellverzeichnis für die Bibliothek AssignNet innerhalb des lokalen Makefiles gleich ${SRCDIR}
${SRCDIR}/lib	Symlink zu ${ROOTDIR}/lib
${SRCDIR}/obj	ein Symlink zu dem aktuellen Build Verzeichnis Ist innerhalb des lokalen Makefiles gleich ${OBJDIR}
${SRCDIR}/.depend/*.d	die Abhängigkeiten der Quellen, eine Datei je .cc oder .c
${SRCDIR}/.depend/makefile.dependencies.0	alle temporäre Datei für die Abhängigkeiten
${SRCDIR}/makefile.dependencies	Include−Datei für makefile
${SRCDIR}/makefile	das Makefile für diese Bibliothek und alle seine Pro−gramme
${SRCDIR}/.cc_exe	Liste der Programme, die aus c++ Quellen in diesem Verzeichnis erzeugt werden.
${SRCDIR}/.cvsignore ${SRCDIR}/CVS	Teil der CVS Versionsverwaltung

Tabelle K−4 Struktur eines einzelnen Bibliotheksverzeichnisses.

K.2.1 Die Make Dateien

DIe Make−Dateien des Projektes bilden eine Hierarchie. Jedes Verzeichnis enthält eine Datei namens *makefile* die alle weiteren Makefiles einbindet.

Datei	Erklärung
${ROOTDIR}/makefile	das zentrale Makefile, startet globale Aktionen bindet ${ROOTDIR}/makefile.tools und make-file.root ein.
${ROOTDIR}/makefile.root	zentrale Definition des aktuellen Orts im Dateisystem. definiert die Wurzel des Quell− und des Buildbaumes.
${ROOTDIR}/makefile.common	zentrale Definition wichtiger Regeln, wird nur von den Makefiles der Bibliotheken eingebunden. bindet makefile.root, makefile.tools, make-file.build und makefile.os ein.
${ROOTDIR}/makefile.tools	zentrale Definition wichtiger Werkzeuge.
${ROOTDIR}/makefile.os	zentrale Anpassung der Makefiles an das Betriebssy−stem definiert auch den verwendeten Kompiler.
${SRCDIR}/<<bibliothek>>/makefile	das Makefile für eine Bibliothek. bindet ${ROOTDIR}/makefile.common ein.
${OBJBASE}/<<build>>/makefile.build	definiert den Namen des aktiven Builds als Makevariable

Tabelle K−5 Hierarchie der Makefiles

Verzeichnis / Datei	Erklärung
../assign.build/	die Build Hierarchie ${OBJBASE}, enthält die parallelen Buildverzeichnisse
../assign.build/makefile	das zentrale Makefile der Build Hierarchie, startet globale Aktionen innerhalb der Build Hierarchie ${OBJBASE}
../assign.obj/	ein symbolischer Link zum aktiven Buildverzeichnis. −> ${OBJBASE}/<<build>>/
${OBJBASE}/makefile.build_root	zentrale Definition des aktuellen Orts im Dateisystem.
${OBJBASE}/makefile	definiert die Aktionen zum Verändern des aktiven Builds.
${OBJBASE}/makefile.common	gemeinsame Definitionen innerhalb der Build Hierarchie.
${OBJBASE}/<<build>>/	die Wurzel für einen Build. (Ziel für einen Symlink) enthält je ein Verzeichnis je Bibliothek.
${OBJBASE}/<<build>>/makefile	für die Build−Hierarchie.
${OBJBASE}/<<build>>/makefile.build	wird von ${ROOTDIR}/makefile.common eingebun−den. definiert BUILD_FLAG == Name des aktiven Builds
${OBJBASE}/<<build>>/lib	zentrales Bibliotheksverzeichnis (Ziel für einen Symlink)
${OBJBASE}/<<build>>/<<bibliothek>>	obj −Verzeichnis einer Bibliothek (Ziel für einen Symlink)

Tabelle K−6 Struktur des Build−Verzeichnisbaumes

Die Struktur der Makefiles erlaubt es Bibliotheken und Programme mit verschiedenen Optimierungen und Debuggingoptionen zu erzeugen. Dazu muß man den jeweiligen Build <<build>> in assign.obj aktivieren.

```
make  <<build>>
```

Dabei ändert sich der Link für assign.obj. Danach müssen zuerst die Bibliotheken und dann die Programme neu erzeugt werden.

Anhang L Datenformate der Eingabedateien

L.1 Peakpool

Die Peaks eines Datensatzes / Experimentsets werden in einem Peakpool zusammengefaßt. Die `.pool` Datei benennt die `.plf` Dateien, die für einen Pool eingelesen werden sollen.

```
BEGIN POOLFILE
   Length 4
   PeakList "/simul/va/Trigger/orginale/hnco_reorder_19991005.plf"
   PeakList "/simul/va/Trigger/orginale/hnca_test.plf"
   PeakList "/simul/va/Trigger/orginale/cbcanh_test.plf"
   PeakList "/simul/va/Trigger/orginale/cbca_conh_test.plf"
END  POOLFILE
```

L.2 Die `.plf` Datei

Die `.plf` Datei verbindet die einzelnen Dateien eines Experiments zu einer Peakliste. Sie kann unterschiedlich viele Teile (**`PARTS`**) zusammenfassen und sich Dateien mit anderen `.plf` Dateien teilen (zB. die Referenzierung in der `.axis` Datei). Die Teile werden in der in `PARTS` angegebenen Reihenfolge geladen und können sich daher überlagern! Das gilt insbesondere für die `.xpk` und `.list1xx` Dateien!

```
BEGIN PEAKLISTFILE
   SymbName  "hnca"
   Parts     "axis,xpk,vol"
   XPK     "/data/simul/va/Trigger/orginale/hnca_reorder_19991005.xpk"
   Axis    "/data/simul/va/Trigger/orginale/hnca_test.axis"
   Vol     "/data/simul/va/Trigger/orginale/hnca_reorder_19991005.vol"
END PEAKLISTFILE
```

L.3 Die `.axis` Datei

Die `.axis` Dateien enthalten die Referenzierung und Benennung der Achsen. `.axis` Dateien können mit `dumpAxis` aus den Felix `.mat` Dateien extrahiert werden. Dabei müssen die ersten 16 KByte der .mat Datei erhalten sein. Der Rest kann mit Nullen auf die volle Größe aufgefüllt werden. Andernfalls muß man sie aus dem Referenzierungsdialog in FELIX übernehmen. Allerdings stellt FELIX die Zahlenwerte gerundet dar.

```
BEGIN AXISVEC
   VERSION 1
   NAME    "hnca"
   HASDIM  "110100000"
   STARTID 1001
   LASTID  1500
   LENGTH  4
BEGIN AXIS
    AxisName      "HN"
    FelixTitle    "HN"
    Sf            800.234
    Sw            5686.124
    PointsInDim   1024
    RefPoint      926.850
    RefFreq       4339.153
    Isotop        H1
    Unit          Points
END AXIS
BEGIN AXIS
    AxisName      "N"
    FelixTitle    "N"
    Sf            81.096
    Sw            2778.930
    PointsInDim   128
    RefPoint      61.205
    RefFreq       9693.935
    Isotop        N15
    Unit          Points
END AXIS
BEGIN AXIS
    AxisName      ""
    FelixTitle    ""
    Sf            1.000
    Sw            1.000
    PointsInDim   1
    RefPoint      1.000
    RefFreq       1.000
    Isotop        H1
    Unit          Notref
END AXIS
BEGIN AXIS
    AxisName      "CA"
    FelixTitle    "CA"
    Sf            201.229
    Sw            5882.350
    PointsInDim   128
    RefPoint      45.977
    RefFreq       11899.677
    Isotop        C13
    Unit          Points
END AXIS
END AXISVEC
```

L.4 Die `.xpk` Datei

Die `.xpk` Dateien werden von FELIX exportiert. Sie enthalten die Position der Peaks in *points* und die Benennung der Frequenzen, falls sie schon vorhanden ist. Außerdem kann sie das Vorzeichen und die Höhe des Peaks enthalten.

Da FELIX sie mit FORTRAN Routinen schreibt und liest, muß man bei einer manuellen Nachbearbeitung peinlich auf die Formatierung (Leerzeichen vs. Tabulator u.ä.) der Datei achten, wenn man sie in FELIX reimportieren will.

```
c**/home/tnp/felix97/text/hncapoints.txt
xpk       3
  1001   468.742     1.812     0  null              74.412     3.828     0  null             110.005     2.019     0  null                1.00
  1002   592.039     1.688     0  null              32.000     3.346     0  null             107.773     1.991     0  null                1.00
  1003   592.406     2.934     0  null              53.609     3.088     0  null             107.061     1.799     0  null                1.00
  1004   657.097     1.831     0  null             115.561     3.895     0  null             102.401     1.976     0  null                1.00

 ... und so weiter.
```

L.5 Die `.list100` Datei

Die `.list100` Dateien werden auch von FELIX exportiert. Man kann sie zur Definition des Volumens der Peaks verwenden.

```
c**
list100      4
     1  1001    8.594  106.994 100000000.000   52.641   131787496.00
     2  1002    7.739  107.592 100000000.000   62.327    68824496.00
     3  1003    7.737  107.782 100000000.000   57.392    13178810.00
     4  1004    7.288  109.029 100000000.000   43.244    62676944.00

... usw.
```

L.6 Die `.vol` Datei

Die `.vol` Dateien werden auch von FELIX exportiert. Man kann sie zur Definition des Volumens der Peaks verwenden. Die Leerzeilen zwischen den Zeilen mit Daten gehören zum Datenformat !

```
c**volumelist written: Sun 10-10-1999 18:46:57
vol        1
  1001     1.318e+08

  1002     6.882e+07

  1003     1.318e+07

  1004     6.268e+07

... usw.
```

L.7 Protospinsystemlisten

Protospinsysteme werden in der Implementierung als Contexte bezeichnet. Protospinsystemlisten sind daher Contextlisten.

```
BEGIN ContextList
LENGTH 2
BEGIN CONTEXT
 DOUBLE "CA #1" 59.3971
 DOUBLE "CA-1 #1" 60.9885
 DOUBLE "CB #1" 32.3001
 DOUBLE "CB-1 #1" 30.6076
 DOUBLE "ExternFuzzyMap CA #1" 0.961458
 DOUBLE "ExternFuzzyMap CA-1 #1" 0.816397
 DOUBLE "ExternFuzzyMap CB #1" 1
 DOUBLE "ExternFuzzyMap CB-1 #1" 1
 DOUBLE "GRIE #1 Arg" 0.175695
 DOUBLE "GRIE #1 Gln" 0.00905253
 DOUBLE "GRIE #1 Glu" 0.146354
 DOUBLE "GRIE #1 His" 0.00472225
 DOUBLE "GRIE #1 Lys" 0.16855
 DOUBLE "GRIE #1 Met" 0.0369936
 DOUBLE "GRIE #1 Pro" 0.188264
 DOUBLE "GRIE #1 Val" 0.270369
 NODE "cbca_conh CA-1 #1" 2009
 NODE "cbca_conh CB-1 #1" 2010
 NODE "cbcanh CA #1" 3026
 NODE "cbcanh CA-1 #1" 3025
 NODE "cbcanh CB #1" 3027
 NODE "cbcanh CB-1 #1" 3028
 NODE "hnca CA-1 #1" 1162
 NODE "hnco CO-1 #1" 5
END CONTEXT
BEGIN CONTEXT
   ... usw
END CONTEXT
END ContextList
```

L.8 `AS_QFACTOR`

In der `AS_QFACTOR` Datei wird das Gewicht einer Aminosäureposition in der Optimierung definiert.

```
BEGIN AS_QFACTOR
    VERSION "1"
    LENGTH 113
    1, 0.0
    2, 0.1
    3, 0.1
    4, 0.1
    5, 0.2
    6, 0.3
    7, 0.3
    8, 0.3
    9, 0.5
    10, 0.8
    11, 1
    12, 1
   ... usw
    112, 1
    113, 1
END AS_QFACTOR
```

L.9 OptimizerInfo

`OptimizerInfo` Dateien definieren die Wahl der Protospinsystemliste für die einzelne Aminosäure in der Optimierung. Außerdem bestimmen sie die Bewertungsfunktion für die Protospinsysteme, die einer Aminosäure zugewiesen werden, und das Muster, dem die Nachbarschaftsverknüpfungen (Brücken) zwischen den Protospinsystemen der Aminosäuren genügen müssen.

```
BEGIN RandomSequenceOptimizerInfo
   VERSION "3"
   PROTEIN "/simul/va/Trigger/Protein"
   CACHEDIR "/simul/va/Trigger/cache.neu"
   USE_ALLCX_LIST 0
   ALLCX_LIST "/simul/va/Trigger/cache.neu/.filtered.27101999-
   02.reeval.allPeakCX"
   AS_QFACTOR_LIST "/simul/va/Trigger/asQFactor.noHisTag"
BEGIN ASINFO
   LENGTH 113
   1, "leer" , "leer"
   2,  "all.19991011-1900" , "c21w.hf:AS:XX"
   3,  "all.19991011-1900" , "c21w.hf:AS:XGly"
   4,  "all.19991011-1900" , "c21w.hf:AS:GlyX"
   5,  "all.19991011-1900" , "c21w.hf:AS:XX"
   6,  "all.19991011-1900" , "c21w.hf:AS:XX"
   7,  "all.19991011-1900" , "c21w.hf:AS:XX"
       ..... usw.
   10, "all.19991011-1900" , "c21w.hf:AS:XX"
   11, "all.19991011-1900" , "c21w.hf:AS:XGly"
   12, "all.19991011-1900" , "c21w.hf:AS:GlyX"
   13, "all.19991011-1900" , "c21w.hf:AS:XX"
   14, "all.19991011-1900" , "c21w.hf:AS:XX"
       ..... usw.
   31, "all.19991011-1900" , "c21w.hf:AS:XX"
   32, "all.19991011-1900" , "c21w.hf:AS:XX"
   33, "all.19991011-1900" , "c21w.hf:AS:XGly"
   34, "all.19991011-1900" , "c21w.hf:AS:GlyX"
   35, "all.19991011-1900" , "c21w.hf:AS:XX"
   36, "all.19991011-1900" , "c21w.hf:AS:XX"
       ..... usw.
   41, "all.19991011-1900" , "c21w.hf:AS:XX"
   42, "all.19991011-1900" , "c21w.hf:AS:XGly"
   43, "all.19991011-1900" , "c21w.hf:AS:GlyX"
   44, "all.19991011-1900" , "c21w.hf:AS:XX"
   45, "all.19991011-1900" , "c21w.hf:AS:XX"
   46, "all.19991011-1900" , "c21w.hf:AS:XX"
       ..... usw.
   60, "all.19991011-1900" , "c21w.hf:AS:XX"
   61, "all.19991011-1900" , "c21w.hf:AS:XX"
   62, "all.19991011-1900" , "c21w.hf:AS:XGly"
   63, "all.19991011-1900" , "c21w.hf:AS:GlyX"
   64, "all.19991011-1900" , "c21w.hf:AS:XX"
   65, "all.19991011-1900" , "c21w.hf:AS:XX"
   66, "all.19991011-1900" , "c21w.hf:AS:XX"
   67, "all.19991011-1900" , "c21w.hf:AS:XX"
   68, "all.19991011-1900" , "c21w.hf:AS:XX"
   69, "all.19991011-1900" , "c21w.hf:AS:XGly"
   70, "all.19991011-1900" , "c21w.hf:AS:GlyX"
   71, "all.19991011-1900" , "c21w.hf:AS:XX"
   72, "all.19991011-1900" , "c21w.hf:AS:XX"
   73, "all.19991011-1900" , "c21w.hf:AS:XGly"
   74, "all.19991011-1900" , "c21w.hf:AS:GlyX"
   75, "all.19991011-1900" , "c21w.hf:AS:XX"
   76, "all.19991011-1900" , "c21w.hf:AS:XX"
       ..... usw.
   88, "all.19991011-1900" , "c21w.hf:AS:XX"
   89, "all.19991011-1900" , "c21w.hf:AS:XX"
   90, "leer" , "leer"
   91, "all.19991011-1900" , "c21w.hf:AS:XX"
   92, "all.19991011-1900" , "c21w.hf:AS:XX"
   93, "all.19991011-1900" , "c21w.hf:AS:XX"
       ..... usw.
   100,"all.19991011-1900" , "c21w.hf:AS:XX"
   101,"all.19991011-1900" , "c21w.hf:AS:XX"
   102,"leer" , "leer"
   103,"all.19991011-1900" , "c21w.hf:AS:XX"
   104,"all.19991011-1900" , "c21w.hf:AS:XX"
       ..... usw.
   112,"all.19991011-1900" , "c21w.hf:AS:XX"
   113,"all.19991011-1900" , "c21w.hf:AS:XX"
END ASINFO
BEGIN LINKINFO
   LENGTH 112
   1,  "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   2,  "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   3,  "c21w.hf:LINK:GLY", "pattern/link/testBridge.pat"
   4,  "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   5,  "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   6,  "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   7,  "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
       ..... usw.
   10, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   11, "c21w.hf:LINK:GLY", "pattern/link/testBridge.pat"
   12, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   13, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   14, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
       ..... usw.
   31, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   32, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   33, "c21w.hf:LINK:GLY", "pattern/link/testBridge.pat"
   34, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   35, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   36, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
       ..... usw.
   41, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   42, "c21w.hf:LINK:GLY", "pattern/link/testBridge.pat"
   43, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   44, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   45, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   46, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
       ..... usw.
   60, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   61, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   62, "c21w.hf:LINK:GLY", "pattern/link/testBridge.pat"
   63, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   64, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   65, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   66, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   67, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   68, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   69, "c21w.hf:LINK:GLY", "pattern/link/testBridge.pat"
   70, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   71, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   72, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   73, "c21w.hf:LINK:GLY", "pattern/link/testBridge.pat"
   74, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   75, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   76, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
       ..... usw.
   88, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   89, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   90, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   91, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   92, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   93, "c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
       ..... usw.
   100,"c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   101,"c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   102,"c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   103,"c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   104,"c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
       ..... usw.
   111,"c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
   112,"c21w.hf:LINK:AS", "pattern/link/testBridge.pat"
END LINKINFO
END RandomSequenceOptimizerInfo
```

L.10 RandomDump

```
BEGIN RandomSequenceOptimizerARENA
   VERSION    "1.03"
   DUMPTIME   "Mon 15-3-1999 12:25:18"
   DUMPREASON "empty"
       BEGIN CONFIG
       DumpTimeInterval  900
       LoopCount         0
       MaxLoopCount      10000
       LastQ             0
       LastQChanged      0
       QTolerance        0.8
       toleranceDecayRate  0.0005
       maxUnchangedLoops 2000
       END CONFIG
       BEGIN ACTIONCONFIG
           BEGIN PROFILES
               VERSION "1.01"
               LENGTH 4
               NAMES  upperLimit swapItems singleExchange crossOver randomGenerate mergeFromBest doNothing moveLinkedRange
   removeRange
               PROFILE    0.6          5          92.5          0.5          0          0          0          2          0.01
               PROFILE    0.8          70         26.5          0.5          0          0          0          3          0.01
               PROFILE   0.95          86          8.5          0.5          0          0          0          5          0.01
               PROFILE 1.0001          86          4.5          0.5          0          0          0          9          0.01
           END PROFILES
       END ACTIONCONFIG
       BEGIN BEST
           BEGIN RandomSequenceOptimizerState
               VERSION  "1.01"
               PROTEINNAME  "Trigger"
               QUALITY    0
               USED_AS     0
               USED_LINKS  0
               BEGIN AS
                   LENGTH  113
                   1,[E],0,
                   2,[E],0,
           ... usw.
                   112,[E],0,
                   113,[E],0,
               END AS
               BEGIN LINK
                   LENGTH  112
                   1,[E],0,
                   2,[E],0,
           ... usw.
                   111,[E],0,
                   112,[E],0,
                   END LINK
           END RandomSequenceOptimizerState
       END BEST
   BEGIN ARENA
      LENGTH 8
    STATE 1
      BEGIN RandomSequenceOptimizerState
                   ... usw. siehe oben
      END RandomSequenceOptimizerState
   STATE 2
                   ... usw. siehe oben
   STATE 8
   END ARENA
   END RandomSequenceOptimizerARENA
```

`RandomDump` Dateien enthalten die Zustandsdefinition des Optimierers. Als Eingabedatei definiert sie die Arena und das Aktionsprofil der Optimierung. Die einzelnen **`RandomSequenceOptimizerState`** Objekte definieren die Startzustände der Individuen S_i. Das **`BEST`** Objekt dient nur zur Dokumentation und wird nach dem Programmstart neu berechnet.

Als Ausgabedatei dokumentieren sie das aktuelle Ergebnis der Optimierung. Für jede leere Aminosäureposition gibt es eine **`"<as-nr>,[E],0,"`** Zeile. Wenn sie zugewiesen ist , enthält sie dann das zugewiesene Protospinsystem.

L.11 Aktionsprofil

Die Profildateien definieren den Anteil jeder definierten Aktion an den Veränderungen, die ein Zustand S_i in **`randomOptimize`** erfahren kann. (Kap. 2.8.1)

L.12 PeakNet

Die PeakNet Dateien (`netfile`) beschreiben die Verknüpfung zwischen Peaks und die Stärke der Verknüpfung.

```
Begin PeakNet
  additionalDim = 0
  NrOfNodes = 794
  nrOfEdges = 1414

Begin NodeList
Begin Node   0
2 4
0 8 2, 3, 0, 1, 4, 5, 6, 7
1 8 10, 11, 8, 9, 12, 13, 14, 15
2 0
3 0
End Node   0
Begin Node   1
2003 4
0 1 0
1 1 8
2 0
3 0
End Node   1
Begin Node   2
       ... usw.
End Node 793
End NodeList
Begin EdgeList
Edge 0 0 0 1 0 0.001
Edge 1 0 0 2 0 0.001
Edge 2 0 0 3 0 0.001
       ... usw.
Edge 1412 785 1 793 1 0.001
Edge 1413 274 1 279 1 0.001
End EdgeList
END PeakNet
```

Anhang M Scripte

Die folgenden Abschnitte und Absätze sind Auszüge aus der Mitschrift während der Zuweisung des Proteins Trigger M (Kap. 5). Eingerückte Zeilen im Fettdruck sind die Fortsetzung der vorhergehenden Zeile. Die `bash` Kommandos wurden in Abschnitte aufgeteilt, die jeweils einer Teilaufgabe während der Zuweisung entsprechen. Sie sollen als Beispiele für die Benutzung der Hilfsprogramme dienen, die die Datenumgebung für die Optimierer erzeugen. Im Anschluß an die Mitschrift zur Zuweisung sind einige Hilfsscripte aufgeführt, die im Hauptteil angesprochen wurden.

M.1 Ein minimales Projekt

Bereite das Projektverzeichnis vor

```
## Wahl eines Projektnamens, der auch als Verzeichnisname in UNIX und als Variablenname in bash
## geeignet ist. d.h. nur Buchstaben und Zahlen und '_', keine Leerzeichen! (hier "CheY25")
mkdir CheY25
cd CheY25
## Editor: setze die Environmentvariablen in .profile.local und setEnv
## binde das Projekt an das Programmverzeichnis an.
## erzeuge die restlichen Verzeichnisse und Links.
## erzeuge CheY25 Protein Datei aus 1 Char oder 3 Char Sequenz.
## Editor:  as1char.txt erzeugen.
##    alle 1 char Codes in einer Zeile.
##    füge die Sequenz 1char in das Modul lib/Protein.pl ein, dann:
##
prepareProjekt.sh
##
. .profile.local
```

- **Import der Daten aus FELIX**

```
cd CheY25; . .profile.local
## Es werden die Peaks der Experimente HNCO, HNCA, CBCA(CO)NH und CBCANH
## benötigt.
# export der .xpk, .vol Dateien aus FELIX.
# extrahieren der .axis Datei aus dem FELIX .mat File
dumpAxis pfad/hnco.mat > orginale/hnco.axis
## ... usw für alle .mat
## Editor: Anlegen einer .plf Datei je Experiment in orginale/
## Reihenfolge der Achsen, Nummerierung der Peaks usw, anpassen.
reorderAchsen.sh
## Editor: Poolfile definieren. -> orginale/Poolfile.pool
#
# lege den Cache für die Protospinsystemlisten an.
genCache -baseplf orginale/hnco -ignore -proteinfile Protein
```

- **Verknüpfen der Netze**

```
cd CheY25; . .profile.local
## Verknüpfung zum Basisknoten. erzeugt netfile
buildC21WNet -p orginale/Poolfile -n netfile -accept
## innere Verknüpfung erzeugen. erzeugt netfile.inner
buildInnerLink -n netfile
## äußere Verknüpfung anlegen.erzeugt netfile.inner.outer
buildInnerLink -n netfile.inner
##
```

- **Suche der Protospinsysteme**

```
cd CheY25; . .profile.local
#
# legt die Protospinsystemlisten je Basispeak im zugehörigen Verzeichnis im Cache an.
#
export CX_EXT_OUT=19990101-01
export NETFILE=netfile.inner.outer
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
export PEAKS_FOR_NODE=9
genContextList_merged -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE -assContext
   completeAS.$CX_EXT_OUT -baseExp hnco  pattern/completeAS.noSub.small.pat
##
```

- **Verteilen der Protospinsysteme auf die Aminosäuren**

```
cd CheY25; . .profile.local
# genCx2AsMapping trägt die CX in den CXListen der Aminosäuren ein,
#   die aufgund der Frequenzen für sie in Frage kommen.
#
export CX_EXT_IN=19990101-01
export CX_EXT_OUT=19990102-01
export MAX_LIST_LEN=400
export MAX_AS_PRO_CX=60
export MIN_STRONG_Q=0.05
export MIN_WEAK_Q=0.002
export MIN_Q=0.0001
genCx2AsMapping -joker 0 -ignoredAS 1 -maxASPosProCXCount $MAX_AS_PRO_CX -maxAsListLen $MAX_LIST_LEN
   -assContext completeAS.$CX_EXT_IN -outName all.$CX_EXT_OUT -minQ $MIN_Q -minWeakQ $MIN_WEAK_Q
   -minStrongQ $MIN_STRONG_Q
## Verteilung der Protospinsysteme auf die Aminosäuren untersuchen.
##   welchen Aminosäuren wurden die CX des BP #n zugewiesen? (Tabelle: BP->AS)
##   welche BPs wurden der Aminosäure zugewiesen? (Tabelle: AS->BP)
## die Ausgabedatei enthält beide Tabellen
findBasePeakinASCache.pl -asIn completeAS.$CX_EXT_OUT > base2as.map.$CX_EXT_OUT
#
#erzeuge die allCX Liste für den Optimierer
#
makeAllCXList -iMode peaks -list completeAS.${CX_EXT_IN}  -assContext
   cache/all.${CX_EXT_OUT}.allPeakCX
```

- **Optimieren**

```
cd CheY25; . .profile.local
## Editor:
##   anlegen der OptimizerInfo Datei.
##   anlegen der Gewichtungsfaktoren für die Aminosäuren.
##
## Anlegen der Optimierungsmakros in sequences/macros für dieses Projekt.
##   kopieren der Dateien mit einem Projektnamen als Erweiterung (zB. .Trigger)
##   in Dateien mit dem eigenen Projektnamen (d.h. .CheY25),
##   und anpassen an das Projekt, dabei sollten die default Werte des Projekts
##   definiert werden.
##
## Lege das Script für diese spezielle Optimierung an. (Kap.M.4)
##   editiere run_optimize_gen
##   Darin kann man die default Werte wieder überschreiben
##   und das Archivverzeichnis definieren.
##
## Aufrufen des Scripts für den Optimiererstart.
```

```
batch < run_optimize_gen
## Dabei entsteht ein .tar.bz2 file das alle Teile des Projektes und die Ausgabe
## der Optimierung enthält.
##
bunzip2  < tar.extension.tar.bz2 | tar tvf -
-rwxr--r-- va/users       1067 1999-11-07 02:11 callscript
-rw-r--r-- va/users         25 1999-11-07 02:11 copyScript.26140
-rw-r--r-- va/users       8449 1999-11-07 15:39 log.extension
-r-------- va/users       4065 1999-11-07 15:39 proc_environ
-rw-r--r-- va/users   11148455 1999-11-07 15:39 RandomDump.extension
-rw-r--r-- va/users      10376 1999-11-07 15:39 RandomDump.extension.best.byAS
-rw-r--r-- va/users        556 1999-11-07 15:39 RandomDump.extension.best.byAS.stat
-rw-r--r-- va/users      10570 1999-11-07 15:39 RandomDump.extension.best.byPeak
-rwx------ va/users       8186 1999-11-07 02:11 run_script
-rw-r--r-- va/users     864427 1999-11-07 15:39 trace.CheY25.extension
```

- **Auswertung**

```
##  Die Dateien im .tar.bz2 sind Rohdaten. Sie können mit den
##  Tools in $ASS/Viewer/graph_tools aufbereitet werden.
cd CheY25; . .profile.local
cd sequences/gen/<<wo_auch_immer>>
## auspacken des .tar.bz2 im lokalen Verzeichnis
bunzip2  < tar.extension.tar.bz2 | tar xvf -
# Um die RandomDump Datei, das log, die .byAS und .byPeak Tabellen
# in eine präsentable Form zu bringen, kann man die folgenden Scripte und Makfiles
# verwenden.
#
## kopieren von makfile.conf (siehe Kap. M.7)
cp ../makefile.cfg makefile.conf
ln -s ../makefile makefile
# Editor:
#    Anpassen makfile.conf an die benutzte Konfiguration der Optimiererdaten
#    und das Protein.
#
## Erzeugt die graphische Aufbereitung des Optimiererlogs.
##  (siehe Kap. M.6)
make
## es entstehen eine Reihe von .ps Dateien, die :
##    einen Graph des zeitlichen Verlaufs der Optimierung,
##  und
##    eine Auswertung der Statistik über die Arena
##  darstellen.
##  (siehe Kap. 3)
##
## Die Zuweisung kann in StarOffice importiert werden, und dann mit der Vorlage "DiffDerZuweisungen"
## als Tabelle aufbereitet. (siehe Tab(5-22 .. 5-25) in Kap 5 Trigger M)
```

M.2 Zuweisung von Trigger

- **Bereite das Projektverzeichnis vor**

```
# siehe auch prepareProjekt.sh
mkdir Trigger
cd    Triger
ln -s $ASS/bin
ln -s $ASS/profile
ln -s $ASS/pattern
touch 00README
#####
#editiere .profile.local
#editiere setEnv.neu
######
mkdir orginale
touch  NetLog
######
### create Trigger Protein from 1 Char or 3 Char Sequence.
#edit as1char.txt
#fuege as1char in lib/Protein.pl ein, dann
genProteinFrom1Char.pl Trigger > ../../../Trigger/Protein
```

- **Berechne die theoretische Qualität einer Trigger Zuweisung**

```
#
# AS    = 113  Proline = 2   Anfang= 1  =>
# zuweisbare AS = Protein Laenge - Pro - Anfang
#               = 110 zuweisbare mit his tag  = 100 ohne his tag
#
# LINKS = AS - (1 Anfang) - (1 ende) - (pro * 2)
#       = 107 mit his tag    = 97 ohne
#
# (siehe Tab. 2-13)
```

- **Analyse: Anzahl der einzelnen Aminosäuren im Protein**

```
#######
# Anzahl 113 Vorspann 13
#
# AS   Anzahl  Im Vorspann(13)  ohne Vorspann (Vorspann bis incl. 10)
#
# Ala  9   0                9
# Arg  1   1                0
# Asn  5   0                5
# Asp  6   0                6
# Cys   -       -   fehlt
# Gln  3   0                3
# Glu  5   1                4
# Gly  7   2   ?            5 (6)
# His  7   6                1
# Ile  8   0                8
# Leu  9   0                9
# Lys  15  0                15
# Met  3   1                2
# Phe  5   0                5
# Pro  2   0                2
# Ser  10  2   ?            8 (9)
# Thr  8   0                8
# Tyr  2   0                2
# Val  8   0                8
#  0   0
#
# SUMME113    13
#
# 11   Gly                       *
# 12   Ser
# 13   Glu      *
# 14   Lys
```

- **Importiere die FELIX Tabellen**

```
###########
### .axis files extrahieren mit   dumpAxis
###
##  hnco                                  aus  tgfhnco.mat
##  hnca                                  aus  tgfhnca.mat
##  cbca_conh  in ref als CBCANH benannt aus  cbcaconh.mat
##  cbcanh     in ref als HNCACB benannt aus  hncacb.mat
############
##   erzeuge den RAW Pool  der Peaks aus den Felix .xpk + .vol Dateien
# Daten vom 19991005
# typ         .xpk file
#                                              Peaks IDs
# hnco        hncopoints       HN    CO    N   106  1-135
# hnca        hncapoints       NH    CA    N   171  1-172
# cbcanh      hncbcapoints     HN    C     N   338  1-342
# cbca_conh   cbcanhpoints     HN    C     N   259  1-262
#
##Axis files editiert
##.plf files erzeugt
#############                                            neue IDs
## hnco   bleibt                                   1    -  999
## hnca                                            1001 - 1999
## cbca_conh                                       2001 - 2999
## cbcanh                                          3001 - 3999
############
```

```
##Peaks neu nummeriert und die Achsen vertauscht.
##neue Achsenreihenfolge:
##HN   N    CO    CA,CAB
##vorher                          nachher
## hnco        HN    CO    N    ==>  HN   N    CO     --
## hnca        NH    CA    N    ==>  HN   N    --     CA
## cbcanh      HN    CAB   N    ==>  HN   N    --     CAB
## cbca_conh   HN    CAB   N    ==>  HN   N    --     CAB
# form:
#reorderXPK  -plf <plf-old>  -vec <new-pos>  -form xpk -o <new-xpk>
# siehe reorderAchsen.sh
reorderAchsen.sh
############
# list100 files für hncacb == cbcanh und hnca
# hncacb.list100
# einige Peaks enthalten ****** statt eines Volumens.
# die meisten Peaks sind CB, d.h. das Vorzeichen ist eindeutig
# unsichere Kanidaten sind 219 (66.7), 223 (66.5), 290 (60.6)
# sind diese CA oder CB ?
# alle **** wurden durch -99999999.99 ersetzt   !!         hnco ids
# << snip --- liste der Peaks mit -99999999.99 >>
##########
# erzeuge orginale/Poolfile.pool
##########
genCache -baseplf orginale/hnco_reorder_19991005 -ignore -proteinfile Protein
##########
```

- **HNCA, HN und N an HNCO anpassen (neu ausrichten)**

```
# optimizeMatch kann die Peaks eines Experiments mittels linearer Projektion so
# verschieben, daß sie mit den Peaks eines Basisexperiments besser übereinstimmen!
# Das Experiment kann dadurch gestaucht, verschoben und gestreckt werden.
# Dazu werden vorher in einer Projektion der Peaks in die [HN/N]-Ebene
# Peakpaare aus beiden Listen gesucht, die eindeutig zusammen gehören,
# (d.h. irgendwo in den Randbereichen des Spektrums) und eine Liste
# mit den Peakidentifikationsnummern paarweise jeweils auf einer Zeile
# angelegt.
# (Kap. 5.2)
#
# Paar files für HNCO + (HNCA,CBCANH,CBCA_CONH)
#
#  hnco_hnca_hn_n_pair.txt
#  hnco_cbcanh_hn_n_pair.txt
#  hnco_cbca_conh_hn_n_pair.txt
#
#Fehlerquelle: die .list100 Files überschreiben die xpk Werte !
#Fix   : statt .list100 Files werden .vol Files verwendet.
#
optimizeMatch -o orginale/hnca_test.axis -maxDistance 0.03,0.5 orginale/hnco_reorder_19991005
   orginale/hnca_reorder_19991005 hnco_hnca_hn_n_pair.txt 0,1,2,3 0,1
#
optimizeMatch -o orginale/cbcanh_test.axis -maxDistance 0.03,0.5 orginale/hnco_reorder_19991005
   orginale/cbcanh_reorder_19991005 hnco_cbcanh_hn_n_pair.txt 0,1,2,3 0,1
#
optimizeMatch -o orginale/cbca_conh_test.axis -maxDistance 0.03,0.5 orginale/hnco_reorder_19991005
   orginale/cbca_conh_reorder_19991005 hnco_cbca_conh_hn_n_pair.txt 0,1,2,3 0,1
###
# jeweils mehrfach optimiert und den besten Lauf genommen.
# gnuplot orginale/ppm_/multi_ ... reorder/test  zum Ansehen
#
# siehe Gl. 5-1 in Kap, 5.2.1
#
# SUMME:            VORHER      NACHHER         PEAKPAARE
# HNCO - HNCA       1.0904      0.25041          23
# HNCO - CBCA_CONH  0.502393    0.209824         40
# HNCO - CBCANH     1.21803     0.438924         58
```

- **Ein PeakNet anlegen. (n−BP mit dem Basispeak verknüpfen.)**

```
#########
# Peak Netz anlegen, Ausgabe ist: netfile_test
buildC21WNet -p orginale/Poolfile_test -n netfile_test -accept -b /data/simul/va/Trigger
## Gewicht in Gl. 5.1
# H    0.05
# N    1.2
# CA   2.0
# CA   1.0 innerhalb des HNCA
#
## << snip -- BP analyse >>
```

- **Suche die Konkurrenzgruppen**

```
#
#
mergeASCX -poolfile orginale/Poolfile_test -netfile netfile_test -pattern
   pattern/completeAS.noSub.pat -glyPattern pattern/listGly.pat
#
#Konkurrenzgruppen:
#Anzahl der Gruppen = 5
#Gruppe[0] = {24, 31, 36, 37, 41, 49, 51, 53, 103}
# 103  eindeutig abtrennbar
# 24   eindeutig abtrennbar
# 31   eindeutig abtrennbar
# 36 + 37  extra Gruppe (extra1),  nur 37 hat genug Peaks
# 41 + 49 + 51 + 53    extra Gruppe (extra2),
# sub gruppen:
#   49+41   zwei gute Spinsysteme, schwach mit (51,53) gekoppelt
#   53+51   nur ein gutes Spinsystem (->51),
#
#Gruppe[1] = {57, 106}         eindeutig trennbar
#Gruppe[2] = {65, 74}          eindeutig trennbar
#Gruppe[3] = {83, 87}          eindeutig trennbar
#Gruppe[4] = {111, 113}        eindeutig trennbar
```

- **Peaklisten in Perl Peak−DB importieren.**

```
##erzeuge NetDB File (perl datenbankfiles)
##name:   NetDB_test
# addiere HNCO , HNCA, CBCANH, CBCA_CONH
# Achsen H,CO,N,CA
# vorher die Header für die Experimente erzeugen
#
cp  NetDB_empty  NetDB_test
addPPMToDB.pl -db NetDB_test -exp hnco -vor 0,1,2,3 -nach 0,2,1,3 <
   orginale/hnco_reorder_19991005.ppm
addPPMToDB.pl -db NetDB_test -exp cbca_conh -vor 0,1,2,3 -nach 0,2,1,3 < orginale/cbca_conh_test.ppm
addPPMToDB.pl -db NetDB_test -exp cbcanh -vor 0,1,2,3 -nach 0,2,1,3 < orginale/cbcanh_test.ppm
addPPMToDB.pl -db NetDB_test -exp hnca -vor 0,1,2,3 -nach 0,2,1,3 < orginale/hnca_test.ppm
```

- **Erzeuge eine druckbare graphische Darstellung der Peakknoten (.ps Dateien)**

```
# erzeugt eine .ps Datei je BP mit einem NodeView Ausdruck (siehe viewCache.pl).
# Die Dateien entstehen im Verzeichnis -dir .....
# die Dateien kann man mit lpr -Ppostscript dateiauswahl   ausdrucken. (Kap. 2.3.7 Abb. 2-3)
printAllNodeGraphs.pl -db NetDB_reorder_dumped.save2  -dir nodeGraphs/NetDB_reorder_dumped.save2
```

- **Betrachten der Protospinsysteme der BP**

```
# die BP-Spinsysteme in der Perl-DB können mit main.pl  betrachtet werden.
# In main.pl können auch die Verknüpfungen zwischen BP und n-BP und die Konkurrenz zwischen den BP
# analysiert und verändert werden. Wenn man die Verknüpfung verändert, muß man die Veränderungen
# entweder in den PeakNet Datenbanken mit dumpNode nachtragen oder die Perl-DB in die PeakNet-DB
# exportieren.
main.pl
```

- **Re−export der Peaks in eine PeakNet−DB, Neuberechnung der Kanten**

```
perlNet2PeakNet -poolfile orginale/Poolfile_test \
   -netDBfile NetDB_test_dumped.save2 \
   -out netfile_test_dumped.save2.p2p \
   -filter -drop 0,0,1,1
#
# output:  netfile_test_dumped.save2.p2p.new
```

```
# entferne alle CA Kanten
dropEdges -n netfile_test_dumped.save2.p2p.new  -batch 1 < dropEdges.txt
# output   netfile_test_dumped.save2.p2p.new.new
########
recalcPNEdgeQuality -poolfile orginale/Poolfile_test  \
   -netfile netfile_test_dumped.save2.p2p.new -cleanup 0 \
   -out netfile_test_dumped.save2.p2p.recalc
#
# H    0.2
# N    1.2
#    die CA werden in buildInnerLink und buildOuterLink
# CA   0
# CA   0 innerhalb des HNCA
#
# wenn mit CA/CB Edges, dann
# CA   1
# CA   1 innerhalb des HNCA
#                 hnca      cbcanh   cbca_conh     hnca
# hnco  ,NH        170       329       208
# cbcanh,N         170       329       208
# hnca,CA         3514      4333      5233        2084
#
# mit dumpNode "filter net 0.01" alle CA Knoten gelöscht
# out: netfile_test_dumped.save2.p2p.recalc
```

- **Anlegen der restlichen PeakNet Konfigurationen.**

```
buildInnerLink -n netfile_test_dumped.save2.p2p.recalc
# alle CA 1.7
# nr of connects: { 628 320 646}
# ohne cleanup , entfernt alle von 101 -> (2045, 2046,3051,3052,3339)
#
# out: netfile_test_dumped.save2.p2p.recalc.inner
```

- **Mit dumpNode kann man die PeakNet−DB direkt editieren.**

```
# mit dumpNode neue Verknüpfung für p101 angelegt.Q immer 0.6
dumpNode < relink.p101.txt
```

- **Analyse der Aminosäure erzeuge die .GlyInfo files**

```
export NETFILE=netfile_test_dumped.save2.p2p.recalc.inner
makeGlyInfo
#
# 7 Gly im Protein -> 14 HNCO Peaks / Protospinsysteme
# gefunden:
#  30 mögliche Gly Basis Peaks (hnco)
#  58 moegliche nicht-HNCO Peaks
#
#  9 komplette G_X (hnca,cbcanh,cbca_conh)
#                                21, 24, 48, 64, 97, 104, 110, 119, 121,
#
#  1 teil G_X      cbcanh,cbca_conh  125
#  5 teil G_X      cbca_conh         23, 34, 59, 86, 132
#
#  8 mögliche X_G  (hnca, cbcanh)    2, 7, 68, 83, 88, 93, 105, 106(hat cbca_conh in der Nähe)
#
#  2 mögliche G_X oder X_G (hnca)    30, 43, 133
#
#  1 ??   defekt?? (zwei hnca)       74
#
# Manueller Test an den Ausdrucken: ##############
#
# FALSCH:    2, 7,  23(?), 30, 43, 48, 59, 86, (?97), 110, 119, ?125 (dreck?),
# G_X:   8:     21, 24, 64, (?97 CB-), 121, 34, 132, 106(?)
# X_G:   8:     68, 74(?), 83, 88, 93, 105, 106(?),  133
#
# unlink 48,2119,    ? 125,2104
#
# aus cont1: zugewiesene Gly BPs (11):
# Gly BP: 21,24,64,68,83,88,93,105,106,121,200
```

- **Testweise die Protospinsystemlisten für die Basispeaks (Peak−Cache) erzeugen.**

```
export NETFILE=netfile_test_dumped.save2.p2p.recalc.inner
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
export PEAKS_FOR_NODE=9
genContextList_merged  -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE -assContext
   completeAS.19991012-1945 -baseExp hnco  pattern/completeAS.noSub.small.pat
```

- **Neubewerten der existierenden Protospinsysteme im Peak−Cache.**

```
# eine Neuberechnung ist dann notwendig, wenn ein Werkzeug nachträglich die Protospinsysteme in
# einer Weise verändert hat, die die Qualitätsberechnung ungültig macht.
reevalContextList  -doFilterGrieName 0 -inContext completeAS.noSub.small  -outContext
   completeAS.noSub.small.out  pattern/reevalASType.pat 2
#
#
export MAX_LIST_LEN=400
export MAX_AS_PRO_CX=60
export MIN_STRONG_Q=0.05
export MIN_WEAK_Q=0.002
export MIN_Q=0.0001
genCx2AsMapping -joker 0 -ignoredAS 1 -maxASPosProCXCount $MAX_AS_PRO_CX -maxAsListLen $MAX_LIST_LEN
   -assContext completeAS.noSub.small.out  -outName all.19991011-1900 -minQ $MIN_Q -minWeakQ
   $MIN_WEAK_Q -minStrongQ  $MIN_STRONG_Q
```

- **Erzeuge die lokalen Perlbedingungen für die manuelle Kontrolle der Protospinsysteme im Peak−Cache.**

```
# Nachdem die Protospinsysteme im Peak-Cache erzeugt wurden, können sie manuell mit viewCache.pl
# überprüft werden und die Auswahlbedingungen für jeden BP eingestellt werden. Damit man die
# offensichtlichsten lokalen Bedingungen nicht alle von Protospinsystemlisten von Hand eingeben muß
# kann man eine Vorauswahl automatisch erzeugen lassen. Die alten Bedingungen werden dabei zertört!
makeAllLocals.pl

# die Auswahl der Protospinsysteme je BP kann dann mittels viewCache.pl
# manuell kontrolliert werden.
viewCache.pl  -proj ~/.ProjectFile
```

- **Re−export der Peaks in eine PeakNet−DB, Neuberechnung der Kanten.**

```
perlNet2PeakNet -poolfile orginale/Poolfile_test \
   -netDBfile NetDB_test_dumped.save2 \
   -out netfile_test_dumped.save2.p2p \
   -filter -drop 0,0,1,1
#
# out:  netfile_test_dumped.save2.p2p.noCACO
#
recalcPNEdgeQuality -poolfile orginale/Poolfile_test  \
   -netfile netfile_test_dumped.save2.p2p.noCACO -cleanup 0 \
   -out netfile_test_dumped.save2.p2p.noCACO.recalc
#
# H    0.3
# N    1.5    trotzdem (101->p3339,p3052,p3051, 2045,2046) low 0.03 + 2* 0.23
#    die CA werden in buildInnerLink und buildOuterLink angelegt.
# CA   0
# CA   0 innerhalb des HNCA
#
#############
# existierende Netfiles:
# netfile_test_dumped.save2.p2p.noCACO.recalc
#    version mit den beiden virtuellen Peaks 44' und 200
#    recalc + ohne CA Edges
#############
```

- **Anlegen der PeakNet−DB Dateien (inner und outer)**

```
buildInnerLink -n netfile_test_dumped.save2.p2p.noCACO.recalc
#
# alle CA mit 2
# Kanten nicht gelöscht,
# nr of connects: { 632 320 652}
#
# out: netfile_test_dumped.save2.p2p.noCACO.recalc.inner
#
############
#
buildOuterLink -n netfile_test_dumped.save2.p2p.noCACO.recalc.inner
#
# nr of connects: { 3880 4333 3514 5233 6318 0}
# bad links nicht entfernt!
# out: netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
```

- **Erzeuge die Protospinsysteme für XX Spinsysteme**

```
############
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
export PEAKS_FOR_NODE=9
genContextList_merged  -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE -assContext
   completeAS.19991021-1916 -baseExp hnco  pattern/completeAS.noSub.small.pat
#
# 34 und 36 sind extrem schlechte Spinsysteme, daher mit anderen Parametern neu berechnen.
#
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.55"
genContextList_merged  -startID 34,36 -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.19991021-1916 -baseExp hnco  pattern/completeAS.noSub.small.pat
#
#########
```

- **Erzeuge lokale Regeln, die die Verknüpfung zwischen den Peaks eines HNCA Paares berücksichtigt.**

```
#
genPeakPairs.pl -cxFile listCA_m1.complete -name "hnca CA-1 #1" -name "cbcanh CA-1 #1" -pairFile
   pair_ca_1
#
genPeakPairs.pl -cxFile listCA_m1.complete -name "hnca CA-1 #1" -name "cbca_conh CA-1 #1" -pairFile
   pair_ca_1
#
#
######### Externe information von Tanja:
#
# CA Freq im CBCANH sind 2.3 ppm höher als im HNCA.
#
# orginale/cbcanh_plus_2ppm.axis  aus    orginale/cbcanh_test.axis   erzeugt
#
# daher Protospinsysteme neu berechnen.
```

- **Die Protospinsysteme werden mit höherer Toleranz neuberechnet.**

```
export POOLFILE=orginale/Poolfile_plus_2ppm
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
export PEAKS_FOR_NODE=9
genContextList_merged  -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE -assContext
   completeAS.plus2ppm.19991023-1334 -baseExp hnco  pattern/completeAS.noSub.small.pat
#
```

- **Die lokalen Bedingungen werden mit viewCache.pl auf die Protospinsysteme angewandt.**

```
# innerhalb der graphischen Oberfläche können die lokalen Nebenbedingungen kontolliert und
# eingestellt werden
viewCache.pl -proj .ProjectFile.Trigger.19991027

# exportiert alle .ProjectFile.Trigger.19991027 (mit p0200)
# nach .filtered.27101999-02
#
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
reevalContextList  -doFilterGrieName 0 -inContext .filtered.27101999-02   -outContext
   .filtered.27101999-02.reeval  pattern/reevalASType.pat
```

- **Verteile die Protospinsysteme auf die Aminosäuren.**

```
#
export MAX_LIST_LEN=400
export MAX_AS_PRO_CX=60
export MIN_STRONG_Q=0.05
export MIN_WEAK_Q=0.002
export MIN_Q=0.0001
genCx2AsMapping -joker 0 -ignoredAS 1 -maxASPosProCXCount $MAX_AS_PRO_CX -maxAsListLen $MAX_LIST_LEN
   -assContext .filtered.27101999-02.reeval  -outName .filtered.27101999-02.reeval -minQ $MIN_Q
   -minWeakQ $MIN_WEAK_Q -minStrongQ  $MIN_STRONG_Q
##
cp OptimizerInfo.all19991011-1900.noHisTag   OptimizeInfo.filtered.27101999-02.reeval
##
## alle listen auf .filtered.27101999-02.reeval umgestellt.
##
```

- **Erzeuge eine Protospinsystemliste, die alle Protospinsysteme umfaßt.**

```
# legt eine Protospinsystemliste an, die die Protospinsysteme von allen zu einem
# AS-Protospinsystemcache gehörenden Listen enthält. Dies ist die so genannte AllCX Liste.
makeAllCXList -iMode peaks -list .filtered.27101999-02.reeval -assContext
   cache.neu/.filtered.27101999-02.reeval
##
## enthält 174 Protospinsysteme
##
cp OptimizerInfo.all19991011-1900.noHisTag   OptimizeInfo.filtered.27101999-02.reeval.allCX
##  alle listen auf "all" umgestellt.
##  allCX Liste ist cache.neu/.filtered.27101999-02.reeval

#######################################
# exportierte alle .ProjectFile.Trigger.19991027_plus2ppm (mit p0200)
# umgestellt  auf  completeAS.plus2ppm.19991023-1334
# exportiert  nach .filtered.27101999-03
```

- **Für besonders schlechte Spinsysteme werden die Protospinsysteme mit angepaßten Parametern erneut berechnet.**

```
# schlechte spinsysteme (d.h. für die die Protospinsystemsuche mit normalen Parametern nur eine
# leere Liste ergibt. Dies sind gleichzeitig die Protospinsysteme, die auch manuell Probleme
# bereiten):
#  p34 (Fuzzy Map CB bei 45 !),   pct=40 ,_wideCB, link 1108->2089
#  p41 neu,                       pct=85
#  p53 neu, wenige Peaks          pct=40,  no filters.
#  p86   cb im gly bereich,       _wideCB pattern
#  p106  cb im gly bereich,       _wideCB pattern, 65% (gly)
#  p17                            _wideCB
#  p12                            _wideCB
#  p97                            _wideCB
export POOLFILE=orginale/Poolfile_plus_2ppm
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
export PEAKS_FOR_NODE=9
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.40"
genContextList_merged -startID 34 -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.plus2ppm.pct40.19991023-1334 -baseExp hnco
```

```
  pattern/completeAS.noSub_wideCB.small.pat
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
genContextList_merged -startID 41 -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.plus2ppm.pct40.19991023-1334 -baseExp hnco
   pattern/completeAS.noSub.small.pat
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.30"
export PEAKS_FOR_NODE=9
genContextList_merged -useFilter 0 -startID 53 -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern
   $PEAKS_FOR_NODE -assContext completeAS.plus2ppm.pct40.19991023-1334 -baseExp hnco
   pattern/completeAS.noSub.small.pat
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
export PEAKS_FOR_NODE=9
genContextList_merged -startID 86 -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.plus2ppm.pct40.19991023-1334 -baseExp hnco
   pattern/completeAS.noSub.small_wideCB.pat
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
export PEAKS_FOR_NODE=9
genContextList_merged -startID 106 -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.plus2ppm.pct85.19991023-1334 -baseExp hnco
   pattern/completeAS.noSub.small_wideCB.pat
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
export PEAKS_FOR_NODE=9
genContextList_merged -startID "17,12,97" -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern
   $PEAKS_FOR_NODE -assContext completeAS.plus2ppm.pct40.19991023-1334 -baseExp hnco
   pattern/completeAS.noSub.small_wideCB.pat
```

- **Danach müssen die Protospinsysteme neu verteilt und die AllCX Liste neuberechnet werden.**

```
# in viewCache.pl die CX erneut mit den lokalen Bedingungen filtern.
# dann ...
export MAX_LIST_LEN=400
export MAX_AS_PRO_CX=60
export MIN_STRONG_Q=0.05
export MIN_WEAK_Q=0.002
export MIN_Q=0.0001
genCx2AsMapping -joker 0 -ignoredAS 1 -maxASPosProCXCount $MAX_AS_PRO_CX -maxAsListLen $MAX_LIST_LEN
   -assContext .filtered.27101999-02_plus2ppm.reeval  -outName .filtered.27101999-02_plus2ppm.reeval
   -minQ $MIN_Q -minWeakQ $MIN_WEAK_Q -minStrongQ  $MIN_STRONG_Q
##
cp OptimizerInfo.all19991011-1900.noHisTag   OptimizeInfo.filtered.27101999-02.reeval
##  alle listen auf .filtered.27101999-02.reeval umgestellt.
##
makeAllCXList -iMode peaks -list .filtered.27101999-02_plus2ppm.reeval -assContext
   cache.neu/.filtered.27101999-02_plus2ppm.reeval
######################
# für die Optimierung der Zuweisung wird bei einigen Peaks die Entartung hoch gesetzt
# peaks 119 , 117, und 112 auf Entartung 2 gesetzt.
#
export POOLFILE=orginale/Poolfile_plus_2ppm_mult
#
########
# Neubewertung der exportierten spinsysteme.
# (mit 2ppm, mit mult)
#
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
export POOLFILE=orginale/Poolfile_plus_2ppm_mult
reevalContextList  -doFilterGrieName 0 -inContext .filtered.27101999-03   -outContext
   .filtered.27101999-03.reeval  pattern/reevalASType.pat
##
##
export MAX_LIST_LEN=400
export MAX_AS_PRO_CX=60
export MIN_STRONG_Q=0.05
export MIN_WEAK_Q=0.002
export MIN_Q=0.0001
genCx2AsMapping -joker 0 -ignoredAS 1 -maxASPosProCXCount $MAX_AS_PRO_CX -maxAsListLen $MAX_LIST_LEN
   -assContext .filtered.27101999-03.reeval  -outName .filtered.27101999-03.reeval -minQ $MIN_Q
   -minWeakQ $MIN_WEAK_Q -minStrongQ  $MIN_STRONG_Q
##
##
makeAllCXList -iMode peaks -list .filtered.27101999-03.reeval -assContext
   cache.neu/.filtered.27101999-03.reeval
##
############
buildOuterLink -n netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
#
# nr of connects: { 7168 8119 6518 9925 12368 0}
# bad links nicht entfernt!
# out: netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.4_0
```

- **Bilde eine Tabelle der möglichen Positionen für jeden BP im Protein**

```
# mapping as -> peak und inverse
findBasePeakinASCache.pl -asIn .filtered.27101999-03.reeval >base2as.map.filtered.27101999-03.reeval
```

- **Optimiere die Zuweisung**

```
# Starte eine Optimierung mit GENETIC oder RANDOM
# dazu wird ein Optimiererscript konfiguriert und an die Aufgabe angepasst.
# (siehe Script für cont 1 weiter hinten)
# das Script nutzt die macros in sequences/macros um die Defaultwerte für die
# Parameter zu definieren.
cd sequences/gen
at now + 1  optimierer_script
# nach ende der optimierung erhält man ein .tar.gz Archiv, daß alle erzeugten Dateien
# und eine Kopie des Scripts sowie eine Kopie des effektiven Scripts (mit den expandierten
#  Variablen) enthält.
# die Zuweisung wird in StarOffice importiert, und dann mit der Vorlage "DiffDerZuweisungen"
# als .sdc File graphisch aufbereitet. (siehe Tabelle bei Trigger)
```

- **Welcher BP kann welchen AS zugeordnet werden ?**

```
#################
## test mapping
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
export POOLFILE=orginale/Poolfile_plus_2ppm_mult
export MAX_LIST_LEN=400
export MAX_AS_PRO_CX=60
export MIN_STRONG_Q=0.05
export MIN_WEAK_Q=0.002
export MIN_Q=0.0001
export IN_LIST=.filtered.27101999-03.reeval
export OUT_LIST=.filtered.27101999-03.reeval.test
# mit joker
genCx2AsMapping -joker 1 -ignoredAS 1 -maxASPosProCXCount $MAX_AS_PRO_CX -maxAsListLen $MAX_LIST_LEN
   -assContext $IN_LIST -outName $OUT_LIST -minQ $MIN_Q -minWeakQ $MIN_WEAK_Q -minStrongQ
   $MIN_STRONG_Q
# suche die möglichen Zuordnungen
findBasePeakinASCache.pl -asIn .filtered.27101999-03.reeval.test > base2as.map.filtered.27101999-
   03.reeval.test
#
# als .sdc File darstellen.
```

- **Suche Nachbarn für die manuelle Qualitätskontrolle, aufgrund der PPM Frequenzen.**

```
# build PPM Lists for unused Peaks
#
printSelectedPeaks.pl -db NetDB_test_dumped.save2 -list unused.plus2ppm_n4_cont1 -ignore 75 >
   unused.plus2ppm_n4_cont1.ppm
printSelectedPeaks.pl -db NetDB_test_dumped.save2 -list unused.plus2ppm_n4_cont2 -ignore 75 >
   unused.plus2ppm_n4_cont2.ppm
printSelectedPeaks.pl -db NetDB_test_dumped.save2 -list unused.plus2ppm_n4_fast -ignore 75 >
   unused.plus2ppm_n4_fast.ppm
#
# vergleich TNP assignment und VA hnco
findBestNeighbourPeak.pl -sparse -order "ppm,ppm" -maxWanted 4  orginale/hnco_reorder_19991005.ppm
   data/ftp/tgftab_hn_sorted.ppm > va_TO_tnp.txt
#
findBestNeighbourPeak.pl -sparse -reverse -order "ppm,ppm" -maxWanted 4
   orginale/hnco_reorder_19991005.ppm data/ftp/tgftab_hn_sorted.ppm > tnp_TO_va.txt
```

- **Das neu gefundene Protospinsystem BP201 hinzufügen**

```
# HNCO Peak 201 hinzugefügt. in NetDB_test_dumped.save3
# bei ppm(7.804 125.159 175.5) == points(583.35252 40.29852 127.60132)
#
# in hnco_reorder_19991005.xpk
# und in den netfiles + cache dir !
# Eintraege in .globalPage.19991027_plus2ppm
# hnco_reorder_19991005.ppm  erneuert.
#  hnco ....ppm !! erste Zeile manuell korrigieren (space)!
#
Xpk2PPM orginale/hnco_reorder_19991005 > orginale/hnco_reorder_19991005.ppm
#
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
export POOLFILE=orginale/Poolfile_plus_2ppm_mult
#
dumpNode
# in dumpNode
make node 201
link 0 3231 0 1
link 0 3232 0 1
link 0 3233 0 1
link 0 3234 0 1
link 1 3231 1 1
link 1 3232 1 1
link 1 3233 1 1
link 1 3234 1 1
#
# Der neue BP muß auch in alle anderen Netze eingefügt werden.
#
# Die Protospinsystemlisten für BP-201 muß angelegt werden.
export POOLFILE=orginale/Poolfile_plus_2ppm_mult
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.4_0
export PEAKS_FOR_NODE=9
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.95"
######### neuer Peak 201
genContextList_merged -startID "201" -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.plus2ppm.19991105-2316 -baseExp hnco  pattern/completeAS.noSub.pat
```

- **Erster Vergleich der Zuweisungen: auto (cont) <–> manuell**

```
#  auto    Anzahl  Kommentare
#  missing: (1) ;  66, manchmal mit P47 (cont2) oder P31(fast)
#                    (wie tnp) zugewiesen.
#
#  manuell
#  missing:  6  ;  12, 20, 50, 51(auto=sehr schwach), 82, 106,
#
#  verschieden:
#           10  ;  13, 14, 16, 17, 19, 22, 25, 27, 60, 61, 67, (92), 103
#                    (92 mit vertauschtem HNCO Peak, siehe as111)
#
#################################
# viewCache.pl  p0202 konfiguriert. (liste angewählt, keine Filter)
#
# .filtered.05111999-01  ausgecheckt.
#################################
export CX_EXPORT=.filtered.05111999-01
##
##
reevalContextList  -doFilterGrieName 0 -inContext ${CX_EXPORT}  -outContext ${CX_EXPORT}.reeval
   pattern/reevalASType.pat
##
##
export MAX_LIST_LEN=400
export MAX_AS_PRO_CX=60
export MIN_STRONG_Q=0.05
export MIN_WEAK_Q=0.002
export MIN_Q=0.0001
genCx2AsMapping  -joker 1 -ignoredAS 1 -maxASPosProCXCount $MAX_AS_PRO_CX -maxAsListLen
   $MAX_LIST_LEN -assContext ${CX_EXPORT}.reeval  -outName ${CX_EXPORT}.reeval -minQ $MIN_Q
   -minWeakQ $MIN_WEAK_Q -minStrongQ  $MIN_STRONG_Q
##
findBasePeakinASCache.pl -asIn ${CX_EXPORT}.reeval > base2as.map${CX_EXPORT}.reeval
##
##
```

```
makeAllCXList -iMode peaks -list ${CX_EXPORT}.reeval -assContext cache.neu/${CX_EXPORT}.reeval
##
##
cp OptimizeInfo.filtered.27101999-02.reeval OptimizeInfo${CX_EXPORT}.reeval.temp
sed  -e "/\.filtered\.27101999-02/s/\.filtered\.27101999-02/${CX_EXPORT}/"
   OptimizeInfo${CX_EXPORT}.reeval.temp > OptimizeInfo${CX_EXPORT}.reeval
rm OptimizeInfo${CX_EXPORT}.reeval.temp
#
#
cp OptimizeInfo.filtered.27101999-02.reeval.allCX OptimizeInfo${CX_EXPORT}.reeval.allCX.temp
sed  -e "/\.filtered\.27101999-02/s/\.filtered\.27101999-02/${CX_EXPORT}/"
   OptimizeInfo${CX_EXPORT}.reeval.allCX.temp > OptimizeInfo${CX_EXPORT}.reeval.allCX
rm OptimizeInfo${CX_EXPORT}.reeval.allCX.temp
#
#
### g_plus2ppm
#
# 0511199-01  mit subLoop 32000 : kein fast-Lauf! (trotz Extension)
# fast == nur kurze SubLoops
#
```

- **Bilde aus den Peaks, die nicht in der Zuweisung verwendet wurden, eine Perl−DB.**

```
# mit collectUnusedPeaks eine DB mit den unbenutzten Peaks erstellt.
#
collectUnusedPeaks.pl -db NetDB_test_dumped.save3 -byAS
   sequences/gen/RandomDump.plus2ppm_n4_cont.27101999-03.allCX.05.best.byAS -action
   "reduceDB=NetDB.red_cont1"
#
# out:  NetDB.red_cont1
###
# Inspektion der ungenutzten Peaks in NetDB.red_cont1
#
# ein neuer virtueller Basispeak (203)   in NetDB.red_cont1 und NeDB...saveed3
#   xpk: 710.84358 82.93285 127.60132
# 134 + 3340, 3341, 33432  Basispeak in
#
# NODE;203;v1;1;hnco;6.92,175.0,113.75,0;2213|2214|3321|3322,,2213|2214|3321|3322,
#
# mit main.pl ansehen, als Postscript ausdrucken oder im Editor kontrollieren.
```

- **Baue die neu gefundenen Spinsysteme in die Peaknetze ein.**

```
# Vorgehen wie vorher bei BP-201
# in dumpNode
make node 203
link 0 2213 0 1
link 1 2213 1 1
link 0 2214 0 1
link 1 2214 1 1
link 0 3321 0 1
link 1 3321 1 1
link 0 3322 0 1
link 1 3322 1 1
#
#
134
link 0 3340 0 1
link 0 3341 0 1
link 0 3342 0 1
link 1 3340 1 1
link 1 3341 1 1
link 1 3342 1 1
# und in ein neues PeakNet ausgeben.
# out: netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.4_0
#
```

- **Danach müssen die Protospinsysteme usw. neu berechnet werden.**

```
########################
```

```
export POOLFILE=orginale/Poolfile_plus_2ppm_mult
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.4_0
export PEAKS_FOR_NODE=9
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.95"
######### neuer Peak 203
genContextList_merged -startID "203" -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.plus2ppm.19991106-2318 -baseExp hnco  pattern/completeAS.noSub.pat
## ergebnis: 0 cx, da spinsystem sehr schlecht (neg. CA und pos. CB Peaks in cbcanh)
#
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 1.0"
genContextList_merged -startID "134" -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.plus2ppm.19991106-2318 -baseExp hnco  pattern/completeAS.noSub.pat
# ergebnis: 1 cx
#
## viewCache.pl  p0202 konfiguriert. (liste angewählt, keine Filter)
#
# .filtered.06111999-01  ausgecheckt.
#################################
export CX_EXPORT=.filtered.06111999-01
##
##
reevalContextList  -doFilterGrieName 0 -inContext ${CX_EXPORT}  -outContext ${CX_EXPORT}.reeval
   pattern/reevalASType.pat
##
##
export MAX_LIST_LEN=400
export MAX_AS_PRO_CX=60
export MIN_STRONG_Q=0.05
export MIN_WEAK_Q=0.002
export MIN_Q=0.0001
genCx2AsMapping  -joker 0 -ignoredAS 1 -maxASPosProCXCount $MAX_AS_PRO_CX -maxAsListLen
   $MAX_LIST_LEN -assContext ${CX_EXPORT}.reeval  -outName ${CX_EXPORT}.reeval -minQ $MIN_Q
   -minWeakQ $MIN_WEAK_Q -minStrongQ  $MIN_STRONG_Q
##
findBasePeakinASCache.pl -asIn ${CX_EXPORT}.reeval > base2as.map${CX_EXPORT}.reeval
##
##
makeAllCXList -iMode peaks -list ${CX_EXPORT}.reeval -assContext cache.neu/${CX_EXPORT}.reeval
##
##
cp OptimizeInfo.filtered.27101999-02.reeval OptimizeInfo${CX_EXPORT}.reeval.temp
sed  -e "/\.filtered\.27101999-02/s/\.filtered\.27101999-02/${CX_EXPORT}/"
   OptimizeInfo${CX_EXPORT}.reeval.temp > OptimizeInfo${CX_EXPORT}.reeval
rm OptimizeInfo${CX_EXPORT}.reeval.temp
#
#
cp OptimizeInfo.filtered.27101999-02.reeval.allCX OptimizeInfo${CX_EXPORT}.reeval.allCX.temp
sed  -e "/\.filtered\.27101999-02/s/\.filtered\.27101999-02/${CX_EXPORT}/"
   OptimizeInfo${CX_EXPORT}.reeval.allCX.temp > OptimizeInfo${CX_EXPORT}.reeval.allCX
rm OptimizeInfo${CX_EXPORT}.reeval.allCX.temp
#
#
####################
#
#  g_plus2ppm_4h_06111999-01
#
#  mit bad1-20  &  4 h
#
### Optimieren ..
######################
###################### 1999-11-19
######################
#  test, wie sehen die CX Listen aus, wenn PCT=100% ist und CB == wide
#
export POOLFILE=orginale/Poolfile_plus_2ppm_mult
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.4_0
export PEAKS_FOR_NODE=9
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 1.0"
genContextList_merged  -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE -assContext
   completeAS.plus2ppm.19991119-1144 -baseExp hnco  pattern/completeAS.noSub.pat
## zähle die CX in den Listen.
#
grep LEN cache.neu/p*/completeAS.plus2ppm.19991119-1144 >cache.neu/LEN.completeAS.plus2ppm.19991119-
   1144
#
# Basispeaks mit length = 0 (ohne wideCB) , d.h. ohne Protospinsysteme
#
# BP: 9,17,20,21,23,24,30,34,43,47,49,65,74,86,91,93,97,98,
#     101,104,105,112,117,125,129,130,135,200,203
#
# ! gleiche CX liste !
genContextList_merged  -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE -assContext
   completeAS.plus2ppm.19991119-1144 -baseExp hnco  pattern/completeAS.noSub.wideCB.pat
grep LEN cache.neu/p*/completeAS.plus2ppm.19991119-1144 >cache.neu/LEN.completeAS.plus2ppm.19991119-
   1144.wideCB
#
# Basispeaks mit length = 0 (mit wideCB)
#
# BP: 20,23,24,30,43,49,65,74,91,93,98,101,105,112,117,125,129,130,135,200,203
#
# DIFF: 9,17,21,34,47,86,97,104
#
export EXT=19991119-1559
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 1.0"
genContextList_merged  -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE -assContext
   completeAS.plus2ppm.${EXT} -baseExp hnco  pattern/completeAS.noSub.wideCB.small.pat
## zähle die CX in den Listen.
#
grep LEN cache.neu/p*/completeAS.plus2ppm.${EXT} >cache.neu/LEN.completeAS.plus2ppm.${EXT}
#
# Basispeaks mit length = 0 (ohne wideCB)
#9,17,20,21,23,24,30,34,43,47,49,65,74,86,91,93,97,98,101,104,105,112,117,125,129,130,135,200,203
# BP:
#
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.95"
genContextList_merged  -startID
   "9,17,20,21,23,24,30,34,43,47,49,65,74,86,91,93,97,98,101,104,105,112,117,125,129,130,135,200,203
   " -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE -assContext
   completeAS.plus2ppm.19991119-1214 -baseExp hnco  pattern/completeAS.noSub.wideCB.pat
grep LEN cache.neu/p*/completeAS.plus2ppm.19991119-1214 >cache.neu/LEN.completeAS.plus2ppm.19991119-
   1214.wideCB
#
# mit length = 0, ohne wide CB
#
# BP: 34,86,91,97,125,130,203
#
# mit length = 0, mit wide CB
#
# BP: 91,125,130,203
#
######### alle mit Gly Mustern
#
export EXT=19991119-1559
export FILTER=" -allowDefects 0 -doFilterGrieName 1 -minPeaksPct 0.85"
genContextList_merged  -startID "21" -verbose 1 ${DEBUG} ${FILTER} -nodesForPattern $PEAKS_FOR_NODE
   -assContext completeAS.plus2ppm.${EXT} -baseExp hnco  pattern/completeAS.noSub.wideCB.small.pat
grep LEN cache.neu/p*/completeAS.plus2ppm.${EXT} >cache.neu/LEN.completeAS.plus2ppm.${EXT}.wideCB
#
```

- **Erzeuge einen AS−Cache, der einer AS nur die BP zuweist, die in der manuellen Zuweisung gefunden wurden.**

```
# Erzeuge die automatische Zuweisung die maximal der manuellen gleicht.
# Um die Zuweisungen zu vergleichen, braucht man eine Zuordnung der Peaks zu den Positionen im
# Protein. Diese entsteht nicht bei der manuellen Zuweisung und muß daher künstlich erzeugt werden.
#
# export endgueltige TNP Zuweisung in /simul/va/DOC/Doktorarbeit/Trigger/tnpZuweisung.txt
#
# alle temporären löschen mit
#  find cache.neu/ -name tnp_only.20000116   -exec rm {} \;
#
# erzeuge den "HF-ONLY" Cache (TNP-ONLY)
genHFPeaksOnlyASCache.pl -noOverwrite -hfCXFormat simple -symlink -as2peak
   /simul/va/DOC/Doktorarbeit/Trigger/tnpZuweisung.txt -out tnp_only.20000116 -in
   .filtered.19991119-01.reeval -hfCXOrder "as,peak" > script_TNP_Only.20000118.sh
sh script_TNP_Only.20000118.sh
#
# p0101 doppelt zugewiesen fehlt daher in as14
#
#
cp OptimizeInfo.filtered.19991119-01.reeval OptimizeInfo.filtered.TNP_ONLY.20000116-01
#
# damit kann dann eine pseudo Optimierung ablaufen, die dem Optimierer nur die Wahl des
```

```
# Protospinsystems, aber nicht die Wahl der Basispeaks offen läßt.
## optimierung tnp_nach
######################
# export TNP Zuweisung vorher > /simul/va/DOC/Doktorarbeit/Trigger/tnpZuweisung_prev.txt
#
# alle löschen mit
#  find cache.neu/ -name tnp_only.20000116   -exec rm {} \;
#
# erzeuge den "HF-ONLY" Cache (TNP-ONLY)
#  nutzt tnp_only.20000116 + as081 -> p0049 als tnp_only_pre.20000116 gelinkt
#
cp OptimizeInfo.filtered.TNP_ONLY_PRE.20000116-01 OptimizeInfo.filtered.TNP_ONLY.20000116-01
# mit manueller Korrektur
#
##### Optimierungen tnp und tnp_pre
## tnp optimierung in seq/gen/tnp
#
## tnp optimierung in seq/gen/tnp_pre
```

- **Bilde eine PerlDB aus den ungenutzten Peaks der Zuweisung**

```
# Ungenutzte Peaks können graphisch dargestellt werden, wenn sie in einer PerlDB vorliegen.
# dazu werden die Peaks aus dem PeakPool ausgewählt, die in einer Zuweisung nicht auftauchen.
#
collectUnusedPeaks.pl -db NetDB_test_dumped.save3 -byAS
   sequences/gen/tnp_pre/RandomDump.plus2ppm_n4_tnp_pre.20000116-01.01.best.byAS -action
   "reduceDB=NetDB.red_tnp_pre"  | grep -v DEBUG  > unused.plus2ppm_n4_tnp_pre
## ungenutzte 14 Basispeaks mit N-BP, insgesammt 169 Peaks = 155 n-BP
## 34,49,53,60,91,111,113,118,123,125,130,200,201,203 (Basispeaks mit N-BP)
## davon 21 mit  >= 2 peaks
## davon 14 mit  >= 3 peaks
## davon 10 mit  >= 4 peaks
##
collectUnusedPeaks.pl -db NetDB_test_dumped.save3 -byAS
   sequences/gen/tnp/RandomDump.plus2ppm_n4_tnp.20000116-01.01.best.byAS -action
   "reduceDB=NetDB.red_tnp"  | grep -v DEBUG  > unused.plus2ppm_n4_tnp
## ungenutzte 10 Basispeaks mit N-BP, insgesamt 146 Peaks = 136 n-BP
## 34,53,60,91,111,118,123,125,200,203 (Basispeaks mit N-BP)
## davon 17 mit  >= 2 peaks
## davon 11 mit  >= 3 peaks
## davon 6  mit  >= 4 peaks
```

M.3 reorderAchsen.sh

```
#!/bin/sh

#
# ordnet die Achsen der rohen Spektren neu an.
#
##neue Achsenreihenfolge:
#HN   N    CO    CA,CAB
#vorher                          nachher
# hnco        HN    CO    N    ==>  HN   N    CO     --
# hnca        NH    CA    N    ==>  HN   N    --     CA
# cbcanh      HN    CAB   N    ==>  HN   N    --     CAB
# cbca_conh   HN    CAB   N    ==>  HN   N    --     CAB
#
# form:
#reorderXPK  -plf <plf-old>  -vec <new-pos>  -form xpk -o <new-xpk>
#
# VEC = index der spalte aus der spalte 0 stammt,...
# eg:
#   hnco NH vorher in col=0 nachher in col=0 ==>  0
#   hnco N  vorher in col=2 nachher in col=1 ==>  2
#   hnco CO vorher in col=1 nachher in col=2 ==>  1
#   hnco CA vorher (nicht vorhanden) nachher in col=3 ==>  -1
#   VEC=0,2,1,-1
# leere spalten  == -1
#
DIR=$BASEPATH/orginale

## INPUT Extension
RAW=_raw_19991005
## OUTPUT Extension
REORDERED=_reorder_19991005

BASE=hnco
VEC=0,2,1,-1
SRC_PLF=$DIR/$BASE$RAW
DEST_PLF=$DIR/$BASE$REORDERED
sed -e /$RAW/s//$REORDERED/  $SRC_PLF.plf > $DEST_PLF.plf
reorderXPK -o $DIR/$BASE$REORDERED.axis $SRC_PLF $VEC axis
reorderXPK -o $DIR/$BASE$REORDERED.xpk  $SRC_PLF $VEC xpk
test -e $DIR/$BASE$RAW.list100 && reorderXPK -o $DIR/$BASE$REORDERED.list100 $SRC_PLF $VEC list100
Xpk2PPM $DEST_PLF > $DEST_PLF.ppm


BASE=cbca_conh
VEC=0,2,-1,1
IDSHIFT=2000
SRC_PLF=$DIR/$BASE$RAW
DEST_PLF=$DIR/$BASE$REORDERED
sed -e /$RAW/s//$REORDERED/  $SRC_PLF.plf > $DEST_PLF.plf
reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.axis $SRC_PLF $VEC axis
reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.xpk  $SRC_PLF $VEC xpk
test -e $DIR/$BASE$RAW.list100 && reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.vol  $SRC_PLF $VEC
   vol
test -e $DIR/$BASE$RAW.list100 && reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.list100 $SRC_PLF
   $VEC list100
Xpk2PPM $DEST_PLF > $DEST_PLF.ppm


BASE=cbcanh
VEC=0,2,-1,1
IDSHIFT=3000
SRC_PLF=$DIR/$BASE$RAW
DEST_PLF=$DIR/$BASE$REORDERED
sed -e /$RAW/s//$REORDERED/  $SRC_PLF.plf > $DEST_PLF.plf
reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.axis $SRC_PLF $VEC axis
reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.xpk  $SRC_PLF $VEC xpk
test -e $DIR/$BASE$RAW.list100 && reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.vol  $SRC_PLF $VEC
   vol
test -e $DIR/$BASE$RAW.list100 && reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.list100 $SRC_PLF
   $VEC list100
Xpk2PPM $DEST_PLF > $DEST_PLF.ppm


BASE=hnca
VEC=0,2,-1,1
IDSHIFT=1000
SRC_PLF=$DIR/$BASE$RAW
DEST_PLF=$DIR/$BASE$REORDERED
sed -e /$RAW/s//$REORDERED/  $SRC_PLF.plf > $DEST_PLF.plf
reorderXPK  -i $IDSHIFT -o $DIR/$BASE$REORDERED.axis $SRC_PLF $VEC axis
reorderXPK  -i $IDSHIFT -o $DIR/$BASE$REORDERED.xpk  $SRC_PLF $VEC xpk
test -e $DIR/$BASE$RAW.list100 && reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.vol  $SRC_PLF $VEC
   vol
test -e $DIR/$BASE$RAW.list100 && reorderXPK -i $IDSHIFT -o $DIR/$BASE$REORDERED.list100 $SRC_PLF
   $VEC list100
Xpk2PPM $DEST_PLF > $DEST_PLF.ppm
```

M.4 Optimierer Start Script für cont01

```
#!/bin/sh

# STUFE 1 Cont 01
# setzt die Optimierung von g_plus2ppm_fast fort.
# startet mit RandomDump.plus2ppm_n4_fast.2710199-03.10
#

. ~/.profile
cd /simul/va/Trigger
. .profile.local

export VER_EXT=default
    #export RAND=rand1
export RAND=gen1
    #export WITH_STAT=YES
```

```
export WITH_STAT=NO
    #export USE_PERSISTENT=YES
export USE_PERSISTENT=NO

export ARCH_DIR=${BASEPATH}/sequences/gen/

. ${RUN_MACROS}/quality
. ${RUN_MACROS}/files
. ${RUN_MACROS}/parameter

## SUPERSEED PARAMETER
export POOLFILE=orginale/Poolfile_plus_2ppm_mult
    #export  NETFILE=$NETFILE_2
export NETFILE=netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0
    #export  NETFILE=$NETFILE_4
    #export  MAX_Q=${MAX_Q_N}
    # _oN2
    #export CONFIG_NAME=OptimizerInfo.nur_hfPeaks.merged.19990314-02
    # _oM83
    #export CONFIG_NAME=OptimizerInfo.merged.19990314-03.no83-85
    #xport CONFIG_NAME=OptimizerInfo.filtered.14071999-01
export CONFIG_NAME=OptimizeInfo.filtered.27101999-03.reeval.allCX
    #_oM3
    #export CONFIG_NAME=OptimizerInfo.merged.19990314-03
export PROFILE_NAME=profile/profile.mix1_2
export  MAX_TIME=3600
    #export  STAT_TIME=480

export  GEN_NROFSEQUENCES=32
    #export  IN_NAME=RandomDump.${GEN_NROFSEQUENCES}.empty
# setzt die Optimierung von g_plus2ppm_fast fort.
# war:   Q=473.317    AS=95    Links=81
#
export  IN_NAME=${ARCH_DIR}/RandomDump.plus2ppm_n4_fast.27101999-03.01

export  SEEDPOOLSIZE=16
export  SURVIVINGSEEDS=8

export GEN_CROSSOP=Gs2p2
export CROSSOVER_PCT=0.5

export SUB_MAX_LOOPS=32000
export SUB_MAX_IDLE_LOOPS=700
export SUB_MAX_TIME=10
export GEN_G_TOL=0.999
export GEN_Q_THRESHOLD_DECAY_RATE=0.05
export GEN_CROSSOVER_RATE=0.9

export Q_TOL=0.9
export THRESHOLD_DECAY_RATE=0.005
export MIN_TTL=2
export MAX_TTL=5

export MAX_IDLE=10000

export TRACE_INTERVAL=100
    #export MULTIRUN=10

export LINK_CACHE_SIZE=400000

###
. ${RUN_MACROS}/max_quality
. ${RUN_MACROS}/extension
## EXTENSION
export VER_EXT=plus2ppm_n4_cont.27101999-03.allCX.05
    #export VER_EXT=${VER_EXT}_qT09_nurHf

echo "#*** NACH EXTENSION"
. ${RUN_MACROS}/output_files
## SUPERSEED FILES
    #export  PCT=85
    #export REF_STATE=${BASEPATH}/sequences/best/R1.n_0.drop${PCT}-01
    #. ${ARCH_DIR}/drop_q

##
echo "#*** VOR TRACE LOGFILE "
echo > ${TRACE_NAME}
traceLogFile ${TRACE_NAME}

#################################
echo "#*** VOR TEMP_DIR"
export TEMP_DIR=/tmp/run.$$
mkdir ${TEMP_DIR}
trap "rm -rf ${TEMP_DIR} 2>/dev/null" EXIT
. ${RUN_MACROS}/copyScript

export TEMP_SCRIPT=${TEMP_DIR}/callscript
echo "#*** VOR BUILD CALL"
. ${RUN_MACROS}/buildCall

##############
echo -n "# "                    >  ${LOG_NAME}
date                            >> ${LOG_NAME}
echo  "#!/bin/sh "              >> ${LOG_NAME}
echo "#SCRIPT is $0        "    >> ${LOG_NAME}
echo "#ARG1 is $1          "    >> ${LOG_NAME}
echo "#ARG2 is $2          "    >> ${LOG_NAME}
echo "#optimizer call was: "    >> ${LOG_NAME}
echo                            >> ${LOG_NAME}
cat ${TEMP_SCRIPT}              >> ${LOG_NAME}
echo                            >> ${LOG_NAME}
chmod u+x ${TEMP_SCRIPT}
echo "#*** VOR call TEMP SCRIPT"

(
    time ${TEMP_SCRIPT} 2>&1
)  >> ${LOG_NAME} 2>&1
#################################
. ${RUN_MACROS}/buildTar
```

M.5 Die resultierende Kommandozeile für cont01

```
# Das oben angeführte Script für cont01 erzeugt diese Kommandozeile.
# (alles in einer Zeile, Umbruch fand nur an Space statt! Die ersten 2 Zeilen mit # fallen weg.)
/simul/va/assign/bin/geneticOptimize -net
   netfile_test_dumped.save2.p2p.noCACO.recalc.inner.outer.2_0 -indepFile
   OptimizeInfo.filtered.27101999-03.reeval.allCX -optiFileName
   /simul/va/Trigger//sequences/gen//RandomDump.plus2ppm_n4_fast.27101999-03.01 -dumpInterval 10
   -dumpFileName /simul/va/Trigger//sequences/gen//RandomDump.plus2ppm_n4_cont.27101999-03.allCX.05
   -maxTime 3600 -random time -maxLoops 1000000 -thresholdDecayRate 0.005 -maxIdleLoops 10000
   -minTTL 2 -maxTTL 5 -qThreshold 0.9 -profile profile/profile.mix1_2 -traceInterval 100 -traceFile
   /simul/va/Trigger//sequences/gen//trace.Trigger.plus2ppm_n4_cont.27101999-03.allCX.05 -debug
   traceEachQ,TRACEDUMPS,TracePopulation -calc as_link  -endQ 808.1685  -crossOverPct 0.5
   -genArenaSize 32 -seedPoolSize 16 -survivingSeeds 8 -genQThreshold 0.999 -subMaxTime 10
   -subMaxLoops 32000 -subTraceInterval 1000 -subMaxIdleLoops 700 -finalMaxLoops 20000 -finalRunTime
   20 -finalMaxIdle 2000  -optiOp Gs2p2  -genQThresholdDecayRate 0.05  -genMaxEqualLoops 15
   -genMaxIdleLoops 15  -genCrossOverThreshold 0.9
```

M.6 makefile für die graphische Aufbereitung der Ergebnisse eines Optimiererlaufes

```
#
# gemeinsames Makefile fuer alle Unterverzeichnisse
# interpretiert die statistischen Daten (log) der Optimierung und
# erzeugt alle Graphen.

# diese Datei wird in jedem Unterverzeichnisse angepasst.
include ./makefile.conf

########################
all: prep createGPL eps

prep: .is_stripped  subLoopSumme extractData

.is_stripped: trace
```

```
   make stripTrace
   touch .is_stripped

stripTrace:
   stripToGEN.pl trace


${DIR}.globalMeanSubLoopStat:
   calcLoopSum.pl -in trace -def global -def group -def groupMean -outBase ${DIR}

${DIR}.Stat: stat
   perl -n -e 'if(/^\d+\s+[SF]\s+\d+\s+\d+/){ print ; };' < stat > ${DIR}.Stat

subLoopSumme: .is_stripped ${DIR}.globalMeanSubLoopStat ${DIR}.Stat

extractData: .is_stripped
   extractData.pl -in trace -outBase ${DIR} -arenaSize ${ARENA_SIZE} -survivingSeeds
   ${SURVIVING_SEEDS}

# erzeugt die Graphen, die den Zeitlichen Verlauf der Optimierung darstellen.
createGPL: .is_stripped
   filterTraceEvol.pl -in trace -outBase ${DIR} -form all -plot qal2_all -arenaSize ${ARENA_SIZE}
   -survivingSeeds ${SURVIVING_SEEDS}  -def fixY="300:700"
   filterTraceEvol.pl -in trace -outBase ${DIR} -form all -plot qWolke -arenaSize ${ARENA_SIZE}
   -survivingSeeds ${SURVIVING_SEEDS}  -def fixY="300:700"
   filterTraceEvol.pl -in trace -outBase ${DIR} -form qWolkeMulti -plot qWolkeMulti -arenaSize
   ${ARENA_SIZE} -survivingSeeds ${SURVIVING_SEEDS} -def noTitle -def noShow  -def noKeys -def
   fixY="300:700"

eps:
   find . -maxdepth 1 -name "${DIR}*.gpl" -exec gnuplot {} \;

####################
groups:
   groupStat.pl -set evol_q -in ${DIR}.g3.evol -in ${DIR}.g2.evol -in ${DIR}.g1.evol -out
   test.group.evol
   meanStat.pl -set evol_q -in ${DIR}.g3.evol -in ${DIR}.g2.evol -in ${DIR}.g1.evol -out
   test.grouping.evol  -groups "all"

groups_arena: groups
   meanStat.pl -set evol_q -in ${DIR}.g3.evol -in ${DIR}.g2.evol -in ${DIR}.g1.evol -out
   test.grouping16.evol  -groups "arena=${ARENA_SIZE},${ELTERN},${SURVIVING_SEEDS}"
   matUnion.pl  -in test.grouping16.evol  -in  ${DIR}.groupMeanSubLoopStat  -out
   test.groupingUnion.evol


####################
clean_gpl:
   rm -f ${DIR}*.gpl

clean:
   rm -f *.eps *.gpl
   rm -f ${DIR}.g*
```

M.7 makefile.conf

```
#ein Bespiel für die Konfiguration des vorigen Makefiles
#
DIR=R-15
ARENA_SIZE=64
ELTERN=16
SURVIVING_SEEDS=16
```

M.8 prepareProject.sh

```
#!/bin/sh

#
#  Vorbereitung eines Projekts.
#
#  vorher:
#     as1char.txt erzeugen.
#     lib/Protein.pl mit dem Protein und der Sequenz ergänzen.
#     erzeuge ein   .profile.local   (kopieren & Pfade anpassen)
#     erzeuge ein    setEnv.neu      (kopieren & Pfade anpassen)
#
#  ASS zeigt zum Basisverzeichnis des Assign Projekts
#
export PROJ_BASE_DIR=/simul/va
export PROJNAME=Trigger

export PROJDIR=${PROJ_BASE_DIR}/${PROJNAME}
cd     ${PROJ_BASE_DIR}

#
#
if [ ! -d ${PROJDIR} ] ; then
#
# pre-A) anlegen der Verzeichnisse und Datenstrukturen für Trigger
#
#
mkdir  ${PROJNAME}
cd     ${PROJDIR}
mkdir  orginale
ln -s $ASS/profile
ln -s $ASS/pattern
ln -s $ASS/bin
touch 00README
touch  NetLog

if [ ! -e ${PROJDIR}/Protein ] ; then
    # vorher:
    #    as1char.txt erzeugen
    #    lib/Protein.pl mit dem Protein und der Sequenz ergänzen

    # Protein Datei anlegen
    genProteinFrom1Char.pl Trigger > ${PROJDIR}/Protein
fi

#
test -d cache     ||  mkdir cache
test -d sequences ||  mkdir sequences

cd sequence
ln -s $ASS/OptimizerScripts  macros
# gen und rand  nehmen die Optimierungsergebnisse auf.
mkdir gen
mkdir rand

## end if dir PROJNAME not found
fi
```

Anhang N Quelltexte

Die folgenden Seiten enthalten einen kleinen Ausschnitt der C++ Quellen, die zum ASSIGN Projekt gehören. Es wurden nur die Quellen ausgewählt, die direkt zum RANDOM und GENETISCHEN−Optimierer beitragen. Basisklassen wie PeakNet, Context, Resource, Peak usw. wurden nicht aufgenommen. Außerdem mußten alle Teile des Mustersuchalgorithmus, der knotenlokalen Bedingungen, die graphischen Oberflächen für die Konfiguration der knotenlokalen Bedingungen und die Hilfsprogramme aus Platzgründen weggelassen werden.

N.1 Quelltext: randomOptimize.cc

```
//
//
// $Id: randomOptimize.cc,v 1.11.2.1 2000/06/22 12:25:42 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "StreamRedirect.h"
#include "gr_prob.h"
#include "MyACG.h"
#include "Split.h"
#include "MyApplication.h"
#include "MySmplHist.h"
#include "Cache.h"
#include "Protein.h"
#include "PeakPool.h"
#include "ParsePoolFile.h"
#include "PeakNet.h"
#include "ParsePeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "AsciiContextReader.h"
#include "Expression.h"
#include "Condition.h"
#include "AllConditions.h"
#include "ExpressionBuilder.h"
#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "ParseCondition.h"
#include "LinkBuilder.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizer.h"
#include "RandomOpti_Calc.h"
#include "ContextRegister.h"
#include "StateMetrics.h"
#include "stdexcept"

////////////////////////////////////////////////////////////
template <class TVec >
void showStat(ostream& os, const char* name, const TVec& x, int top)
{
    ASSERT(top <= x.count());

    SampleStatistic statistic;

    if ( top > 0 ){
        for(int i = 0; i< top;i++){
            statistic += ( (double)x[i] );
        }
        os << indent << setw(16)<< name
           << "\t  " << statistic.mean()
           << "  "   << statistic.stdDev()
           << endl;
    } else {
        statistic += (double)0;
        os << indent << setw(16)<< name
           << "\t  " << 0
           << "  "   << 0
           << endl;
    }
}

////////////////////////////////////////////////////////////
CmdArgUsage         arg_usage('?', "help", "print usage and exit");

static CmdArgStr    arg_debug('d',"debug","debug", "list of enabled debugging options" ,
   CmdArg::isOPTVALREQ);
static CmdArgStr    arg_action('r',"action","initstate|optimize", "what should it do?" ,
   CmdArg::isOPTVALREQ);

static CmdArgStr    arg_poolFile('p',"poolfile","poolfile", "pool file name" , CmdArg::isOPTVALREQ);
static CmdArgStr    arg_netFile('n',"netfile","netfile", "net file name" , CmdArg::isOPTVALREQ);
static CmdArgStr    arg_optiFileName('n',"optiFileName","filename", "input opti state file name" ,
   CmdArg::isOPTVALREQ);

static CmdArgStr    arg_traceFileName('r',"traceFile","filename", "file for trace messages" ,
   CmdArg::isOPTVALREQ);
static CmdArgStr    arg_stateFileName('n',"indepFile","filename", "state independent file name" ,
   CmdArg::isOPTVALREQ);

static CmdArgStr    arg_referenzState('r',"referenceState","filename", "file of  reference state" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt    arg_multiRun('m',"multiRun","count", "run optimize multiple times (requires
   random=time)" , CmdArg::isOPTVALREQ);
static CmdArgFloat    arg_statTime('m',"statTime","min", "time for all stat loops" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt    arg_statMin('m',"statMin","loop", "lower border for statistic (min loop count)"
   , CmdArg::isOPTVALREQ);
static CmdArgInt    arg_statMax('m',"statMax","loop", "upper border for statistic (max loop count)"
   , CmdArg::isOPTVALREQ);
static CmdArgStr    arg_statOutFile('r',"statOutFile","filename", "output file for statics" ,
   CmdArg::isOPTVALREQ);

static CmdArgStr    arg_calculator('r',"calc","name", "name of calculator" , CmdArg::isOPTVALREQ);
////////////////////////////////////////////////////////////////////
int main(int argc, char ** argv){
    MyApplication app;

    int ret = -1;
    try {
        ret = app.main(argc,argv);
    }
    catch (std::exception& e){
        ErrMessage err;
        err << error
            << "an exception escaped from the main program\n"
            << "exception type is "<<e.what()
            << eom;
    }
    catch (...){
        ErrMessage err;
        err << error
            << "an unknown exception type escaped from the main program."
            << eom;
    }
    return ret;
};
int MyApplication::main(int argc, char** argv){

    debug_set(10);
    force_all_Conditions();
    disableAllAS();

    CmdLine  cmd(*argv,  &arg_usage,
        // PARAMETER
        // OPTIONS
                // main options
        &arg_debug,&arg_action,
                &arg_statTime,
                &arg_multiRun,&arg_statMin,&arg_statMax,&arg_statOutFile,
```

```
            &arg_poolFile,&arg_netFile,
            // files
            &arg_optiFileName,&arg_referenzState,
            &arg_stateFileName,&arg_traceFileName,
            &arg_calculator,
    NULL);
cmd.clear();
cmd.set(CmdLine::KWDS_ONLY | CmdLine::OPTS_FIRST |
   CmdLine::NO_GUESSING  );
// fuer jeden verwendeten Optimizer MUESSEN die Optionen
// registriert werden.
RandomOptimizerBASE::registerOptions(cmd);
RandomOptimizer::registerOptions(cmd);
CmdArgvIter  argv_iter(argc-1, argv+1);
if (cmd.parse(argv_iter)) {
   cerr << "ERROR: need  args: " << endl;
   cmd.error() << "parsing errors occurred!" << endl ;
   return (-1);
}

MyString sep = "/";   // dir separator
double   d;           // temp variable
long int  statTime         = -2;     // [min]
long int  nrOfRuns         = 1;      // nr of runs for statistics
long int  statMin          = 10;
long int  statMax          = 50;

////////////////////////////////
// wenn statTime auf eine Wert größer 0 gesetzt ist
// wird der erste Lauf mit statTime als runTime gestartet.
// neue Läufe starten mit (statTime - schon verbrauchte Zeit)
// bis die maximale Anzahl der Läufe erreicht ist
//    oder
//     die Restzeit == 0 ist .
MyString optiFileName;
MyString stateFileName;
MyString tempFileName;   // set to a temp-file if nrOfRuns > 0
MyString traceFileName;
MyString statOutFileName;

MyString poolName;    //
MyString netName;     //
MyString debugOptions; //

enum action_e { optimize, initState };
action_e action = optimize;
MyString actionOption;

evalOption(debugOptions,"OFF","VA_DEBUG",arg_debug);
debugOptions = upcase(debugOptions) ;

evalOption(optiFileName,"","",arg_optiFileName);
evalOption(stateFileName,"","",arg_stateFileName); // independent data
evalOption(traceFileName,"","",arg_traceFileName);
evalOption(d,-1.0L,"",arg_statTime);
statTime = long(double(d) * 60.0);    // min -> sec
evalOption(nrOfRuns,1,"",arg_multiRun); nrOfRuns=max(long(1), nrOfRuns);
evalOption(statMin,10,"",arg_statMin);  statMin=max(long(1), statMin);
evalOption(statMax,50,"",arg_statMax);  statMax=max(long(1), statMax);

// name of the calculator&cache
MyString calcName = "none";
evalOption(calcName,"none","",arg_calculator);

bool statisticRuns = givenOption(arg_multiRun);

evalOption(actionOption,"OPTIMIZE","",arg_action);
actionOption = upcase(actionOption);
if( actionOption == "OPTIMIZE"){
    action = optimize;
} else if( actionOption == "INITSTATE"){
    action = initState;
} else {
    ErrMessage err;
    err << error
        << "unknown action:"<< actionOption << endl
        << "allowed are: optimize OR init"<< eom;
    return -1;
}

if(statisticRuns){
    const char* name = tempnam ("/temp",NULL);
    tempFileName = name;
    free((void*)name); // allocated in C library with malloc
}

OutputRedirect oRedir(arg_statOutFile);
ostream &out = oRedir.stream();

evalOption(netName, "./netfile","NETFILE",arg_netFile);
evalOption(poolName,"./Poolfile","POOLFILE",arg_poolFile);

Context::Handle globalCx = Context::globalHandle();
////////////////////////////
// get the PeakPool
if(!statisticRuns)
    cout << "reading pool " <<poolName << endl;
PeakPool::Handle pool = PeakPool::make();
if(!loadPeakPool(pool, poolName,*globalCx)){
    return -1;
}

////////////////////////////
// read the Net
if(!statisticRuns)
    cout << "reading net "<< netName << endl;
PeakNet::Handle net;
if(!loadPeakNet(net, netName, pool)){
    return -1;
}

//////////////////////////////////////////
//////////////////////////////////////////
if(!statisticRuns)
    cout << "building Opti independent data" << endl;
if(!statisticRuns)
    cout << "load Opti independent data "<< stateFileName << endl;
StateIndependent::Handle  independent = StateIndependent::loadState(
        stateFileName, pool, net, globalCx);
if(isNull(independent)){
    return -1;
}
independent->calculatorName(calcName);

// configure optimization data
Array<State::Handle> referenceStates;
if(statisticRuns && givenOption(arg_referenzState)){
    cout << "registering CX" << flush;
    ContextRegister reg;
    ContextList::Handle list = independent->allCXList();
    reg.registerCXList( list );
    cout << ".. done" << endl;
    MyString name;
    evalOption(name, "","",arg_referenzState);
    MyStringVec names;
    MyRegex rxSemikolon(";");
    names = split(rxSemikolon,name);
    referenceStates.resize(names.count());
    for(int i=0;i<names.count(); ++i)
    {
        cout << "reading reference state from "<< names[i] << flush;
        RandomOptimizerBASE::Handle opti1(new RandomOptimizerBASE,IINIT);
        opti1->peakNet(net).peakPool(pool).useIndependent(independent);
        if(opti1->loadState(names[i], pool, net)){
            return -1;
        };
        opti1->rebuildQAt(-1);
        referenceStates[i] = opti1->stateAt(-1);
        cout << " Q:="<<referenceStates[i] -> quality();

        cout << " .. done" << endl;
    }

}
```

```
cout << "building Optis " << endl;
RandomOptimizer*   optiP  = new RandomOptimizer;
RandomOptimizerBASE::Handle   opti(optiP, IINIT);

// setze net und pool
opti->peakNet(net).peakPool(pool);

fstream traceFile;
if(traceFileName!=""){
    traceFile.open(traceFileName,ios::out|ios::app);
    if(!traceFile){
        ErrMessage err;
        err << error << "could not open trace file \""
            << traceFileName <<"\" " << eom;
        return -1;
    }
    traceFile << "START_TRACE;"<<now<<endl;
    opti->debugStream(traceFile);
} else {
    opti->debugStream(cerr);
    optiP->debugStream(cerr);
}
opti->debug(debugOptions);
opti->useIndependent(independent);

// statistic arrays
int statTop     =0;
int fail_statTop=0;
longVec    loops(nrOfRuns);
doubleVec  singleTime(nrOfRuns);
longVec    diffActions(nrOfRuns); diffActions.fill(0);

longVec    fail_loops(nrOfRuns);
doubleVec  fail_singleTime(nrOfRuns);
longVec    fail_diffActions(nrOfRuns); fail_diffActions.fill(0);
longVec    fail_asCount(nrOfRuns);
longVec    fail_linkCount(nrOfRuns);
doubleVec  fail_quality(nrOfRuns);

time_t restZeit    = statTime;
time_t usedTime    = -1;   // zeit fuer den vorhergehenden Durchgang
while(nrOfRuns>0 && ((statTime>0 && restZeit>0) || statTime <0)){
    opti->initState();
    // und lade sie neu.
    if(optiFileName != ""){
        cout << "reading opti from "<<optiFileName<<endl;
        if( opti->loadState(optiFileName, pool, net))
            return -1;
    }
    if(!optiP->configure(argv_iter))
        exit(-1);

    // lösche die State Objecte
    if(statisticRuns){
        opti->dumpFileName(tempFileName);
        opti->runTime(restZeit);
        opti->maxLoop(10000000); // sehr gross, nie erreicht
    }
    // starte den Optimizer neu.
    opti->restartRun();

    if(action == initState){
        cout << "init.... " << endl;
        opti->initState();
    } else {
        cout << "optimize.... " << endl;
        time_t startT = time(NULL);
        opti->optimize();
        time_t endT = time(NULL);
        usedTime = endT - startT;
    }
    //
    nrOfRuns --;
    restZeit -= usedTime;
    if(restZeit < 10 ) restZeit = 0;
    //  ignoriere den letzten Durchlauf, wenn er
    //  durch TIMEOUT beeendet wurde.
    //  Der letzte Lauf verfälscht sonst die Statistik.
    //Ausnahme: Der erste Lauf hat die gesammte Zeit verbraucht!
    //  Dann ist dies das Ergebnis und darf nicht ignoriert werden.
    //pseudo code:
    //  wenn (statistik  UND entweder (
    //          erster Lauf
    //        ODER
    //          nicht durch TIMEOUT beendet ))
    if(statisticRuns
       &&  (
              // der erste Durchlauf
              ( fail_statTop + statTop == 0 )  ||
              // nicht durch TIMEOUT beendet.
              (opti->exitReason() != "max time reached" && restZeit > 0)
              )
       )
    {
        const double epsilon = 1e-4;
        bool failed = false;
        int  actions = 0;

        if ( referenceStates.count() > 0 ){
            failed = ((referenceStates[0]->quality()
                       - opti->stateAt(-1)->quality()) > epsilon);
            actions = MAXINT;
            for(int i=0;i<referenceStates.count(); ++i)
            {
                StateDiff diff( opti->stateAt( -1 ),  referenceStates[i]);
                // diff.reduceSwaps();
                diff.finalizePath();
                actions = min(diff.atomDistance(),actions);
            }
        } else {
            failed = ((opti->endQ()
                       - opti->stateAt(-1)->quality()) > epsilon);
        }
        if(!failed){
            loops[statTop]       =  opti->loopCount();
            singleTime[statTop]  =  usedTime;
            diffActions[statTop] =  actions;

            statTop ++;
        } else {
            fail_loops[fail_statTop]     =  opti->lastQChanged();
            fail_quality[fail_statTop]   =  opti->stateAt(-1)->quality();
            fail_asCount[fail_statTop]   =  opti->stateAt(-1)->usedCount();
            fail_linkCount[fail_statTop] =  opti->stateAt(-1)->usedLinks();
            fail_singleTime[fail_statTop]=  usedTime;
            fail_diffActions[fail_statTop]= actions;

            fail_statTop ++;
        }
    } else {
        opti->dumpState();
    }
}
if(statisticRuns){
    WriteBlock block(out,"STATISTIC_DATA");
    for(int i = 0; i< statTop;i++){
        out<< i+1
           << "   S"
           << "  " << loops[i]
           << "  " << singleTime[i]
           << "  " << diffActions[i]
           << endl ;
    }
    for(int i = 0; i< fail_statTop;i++){
        out<< statTop+i+1
           << "   F"
           << "  " << fail_loops[i]
           << "  " << fail_singleTime[i]
           << "  " << fail_quality[i]
           << "  " << fail_asCount[i]
           << "  " << fail_linkCount[i]
           << "  " << fail_diffActions[i]
           << endl;
```

```
        }

        WriteBlock block2(out,"STATISTIC");
        showStat(out, "LOOPS", loops,statTop);
        showStat(out, "TIME",singleTime ,statTop);
        showStat(out, "ACTIONS",diffActions ,statTop);

        showStat(out, "FAIL_AS", fail_asCount,fail_statTop);
        showStat(out, "FAIL_LINK",fail_linkCount ,fail_statTop);
        showStat(out, "FAIL_Q",fail_quality,fail_statTop);
        showStat(out, "FAIL_ACTIONS",fail_diffActions ,fail_statTop);
        showStat(out, "FAIL_LOOPS", fail_loops,fail_statTop);
        showStat(out, "FAIL_TIME",fail_singleTime ,fail_statTop);
        unlink(tempFileName); // unlink tempfile
    }
    traceFile << "END_TRACE;" << now << endl;
    return 0;
};
```

N.2 Quelltext: geneticOptimize.cc

```
//
//
// $Id: geneticOptimize.cc,v 1.14 2000/06/12 15:10:52 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "StreamRedirect.h"
#include "gr_prob.h"
#include "MyACG.h"

#include "Split.h"
#include "MyApplication.h"
#include "MySmplHist.h"
#include "Cache.h"
#include "Protein.h"
#include "PeakPool.h"
#include "ParsePoolFile.h"
#include "PeakNet.h"
#include "ParsePeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "AsciiContextReader.h"
#include "Expression.h"
#include "Condition.h"
#include "AllConditions.h"
#include "ExpressionBuilder.h"

#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "ParseCondition.h"

#include "LinkBuilder.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizer.h"
#include "GeneticRandomOptimizer.h"
#include "GeneticOptimizerFactory.h"

#include "ContextRegister.h"
#include "StateMetrics.h"
#include "stdexcept"
////////////////////////////////////////////////////////////
template <class TVec >
void showStat(ostream& os, const char* name, const TVec& x, int top)
{
    ASSERT(top <= x.count());

    SampleStatistic statistic;

    if ( top > 0 ){
        for(int i = 0; i< top;i++){
            statistic += ( (double)x[i] );
        }
        os << indent << setw(16)<< name
           << "\t  " << statistic.mean()
           << "  "   << statistic.stdDev()
           << endl;
    } else {
        statistic += (double)0;
        os << indent << setw(16)<< name
           << "\t  " << 0
           << "  "   << 0
           << endl;
    }
}
template void showStat<longVec>(
        ostream& os, const char* name,
        const longVec& x, int top);
template void showStat<doubleVec>(
        ostream& os, const char* name,
        const doubleVec& x, int top);

////////////////////////////////////////////////////////////
CmdArgUsage             arg_usage('?', "help", "print usage and exit");
static CmdArgStr        arg_debug('d',"debug","debug", "list of enabled debugging options" ,
   CmdArg::isOPTVALREQ);
static CmdArgStr        arg_action('r',"action","initstate|optimize", "what should it do?" ,
   CmdArg::isOPTVALREQ);
static CmdArgStr        arg_poolFile('p',"poolfile","poolfile", "pool file name" ,
   CmdArg::isOPTVALREQ);
static CmdArgStr        arg_netFile('n',"netfile","netfile", "net file name" , CmdArg::isOPTVALREQ);
static CmdArgStr    arg_optiFileName('n',"optiFileName","filename", "input opti state file name" ,
   CmdArg::isOPTVALREQ);
static CmdArgStr        arg_traceFileName('r',"traceFile","filename", "file for trace messages" ,
   CmdArg::isOPTVALREQ);
static CmdArgStr        arg_stateFileName('n',"indepFile","filename", "state independent file name" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt        arg_finalMaxIdleLoops('t',"finalMaxIdle","loops","like -maxIdleLoops, used
   for the finalizer", CmdArg::isOPTVALREQ);
static CmdArgInt        arg_finalMaxLoops('t',"finalMaxLoops","loops","like -maxLoops, used for the
   finalizer", CmdArg::isOPTVALREQ);
static CmdArgInt        arg_finalRunTime('t',"finalRunTime","min","like -runTime, used for the
   finalizer", CmdArg::isOPTVALREQ);
static CmdArgStr    arg_referenzState('r',"referenceState","filename", "file of  reference state" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt        arg_multiRun('m',"multiRun","count", "run optimize multiple times (requires
   random=time)" , CmdArg::isOPTVALREQ);
static CmdArgFloat       arg_statTime('m',"statTime","min", "time for all stat loops" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt        arg_statMin('m',"statMin","int", "lower border for statistic (max loop
   count)" , CmdArg::isOPTVALREQ);
static CmdArgInt        arg_statMax('m',"statMax","int", "upper border for statistic (min loop
   count)" , CmdArg::isOPTVALREQ);
static CmdArgStr        arg_statOutFile('r',"statOutFile","filename", "output file for statics" ,
   CmdArg::isOPTVALREQ);
static CmdArgStr    arg_calculator('r',"calc","name", "name of calculator" , CmdArg::isOPTVALREQ);
static CmdArgStr    arg_optiOpName('o',"optiOp","key", "name of Optimizer operator (default=Gs2p2)"
    , CmdArg::isOPTVALREQ);
////////////////////////////////////////////////////////////////////
int main(int argc, char ** argv){
```

```
    MyApplication app;
    int ret = -1;
    try {
        ret = app.main(argc,argv);
    }
    catch (std::exception& e){
        ErrMessage err;
        err << error
            << "an exception escaped from the main program\n"
            << "exception type is "<<e.what()
            << eom;
        ret = -1;
    }
    catch (...){
        ErrMessage err;
        err << error
            << "an unknown exception type escaped from the main program."
            << eom;
        ret = -1;
    }
    return ret;
};
int MyApplication::main(int argc, char** argv){

    debug_set(10);
    force_all_Conditions();
    disableAllAS();

    CmdLine  cmd(*argv,  &arg_usage,
        // PARAMETER
        // OPTIONS
                // main options
        &arg_debug,&arg_action,
                &arg_statTime,
                &arg_multiRun,&arg_statMin,&arg_statMax,&arg_statOutFile,
                 // files
                &arg_optiFileName,&arg_referenzState,
                &arg_poolFile,&arg_netFile,&arg_stateFileName,
                &arg_traceFileName,
                // finalizer
                &arg_finalMaxIdleLoops,
                &arg_finalMaxLoops,&arg_finalRunTime,
                &arg_calculator,&arg_optiOpName,
                // files
        NULL);
    cmd.clear();
    cmd.set(CmdLine::KWDS_ONLY | CmdLine::OPTS_FIRST |
       CmdLine::NO_GUESSING  );
    // fuer jeden verwendeten Optimizer MUESSEN die Optionen
    // registriert werden.
    RandomOptimizerBASE::registerOptions(cmd);
    RandomOptimizer::registerOptions(cmd);
    SubRandomOptimizer::registerOptions(cmd);
    GeneticOptimizer::registerOptions(cmd);

    CmdArgvIter  argv_iter(argc-1, argv+1);
    if (cmd.parse(argv_iter)) {
       cerr << "ERROR: need  args: " << endl;
       cmd.error() << "parsing errors occurred!" << endl ;
       return (-1);
    }

    MyString sep = "/";   // dir separator
    double   d;           // temp var
    long int  statTime         = -1;     // [min]
    long int  nrOfRuns         = 1;           // nr of runs for statistics
    long int  statMin          = 10;
    long int  statMax          = 50;

    MyString optiFileName;
    MyString stateFileName;
    MyString tempFileName;   // set to a temp-file if nrOfRuns > 0
    MyString traceFileName;
    MyString statOutFileName;
    MyString poolName;
    MyString netName;
    MyString debugOptions;

    enum action_e { optimize, initState };
    action_e action = optimize;
    MyString actionOption;

    evalOption(debugOptions,"OFF","VA_DEBUG",arg_debug);
    debugOptions = upcase(debugOptions) ;

    evalOption(optiFileName,"","",arg_optiFileName);
    evalOption(stateFileName,"","",arg_stateFileName); // independent data
    evalOption(traceFileName,"","",arg_traceFileName);
    evalOption(d,-1.0,"",arg_statTime);
    statTime = long(d * 60);     // min -> sec
    evalOption(nrOfRuns,1,"",arg_multiRun);
    evalOption(statMin,10,"",arg_statMin);
    evalOption(statMax,50,"",arg_statMax);

    MyString optiOpName;
    evalOption(optiOpName,"default","",arg_optiOpName);

    // name of the calculator&cache
    MyString calcName = "none";
    evalOption(calcName,"none","",arg_calculator);

    bool statisticRuns = givenOption(arg_multiRun);

    evalOption(actionOption,"OPTIMIZE","",arg_action);
    actionOption = upcase(actionOption);
    if( actionOption == "OPTIMIZE"){
        action = optimize;
    } else if( actionOption == "INITSTATE"){
        action = initState;
    } else {
        ErrMessage err;
        err << error
            << "unknown action:"<< actionOption << endl
            << "allowed are: optimize OR initstate"<< eom;
        return -1;
    }

    if(statisticRuns){
        const char* name = tempnam ("/temp",NULL);
        tempFileName = name;
        free((void*)name); // allocated in C library with malloc
    }

    OutputRedirect oRedir(arg_statOutFile);
    ostream &out = oRedir.stream();

    evalOption(netName, "./netfile","NETFILE",arg_netFile);
    evalOption(poolName,"./Poolfile","POOLFILE",arg_poolFile);

    Context::Handle globalCx = Context::globalHandle();
    ////////////////////////////
    // get the PeakPool
    cout << "reading pool " <<poolName << endl;
    PeakPool::Handle pool = PeakPool::make();
    if(!loadPeakPool(pool, poolName,*globalCx)){
        return -1;
    }

    ////////////////////////////
    // read the Net
    cout << "reading net "<< netName << endl;
    PeakNet::Handle net;
    if(!loadPeakNet(net, netName, pool)){
        return -1;
    }

//////////////////////////////////////////
 //////////////////////////////////////////
    cout << "load Opti independent data "<< stateFileName << endl;
    StateIndependent::Handle  independent = StateIndependent::loadState(
            stateFileName, pool, net, globalCx);
    if(isNull(independent)){
        return -1;
    }
```

```
independent->calculatorName(calcName);
Array<State::Handle> referenceStates;
if(statisticRuns && givenOption(arg_referenzState)){
    cout << "registering CX" << flush;
    ContextRegister reg;
    ContextList::Handle list = independent->allCXList();
    reg.registerCXList( list );
    cout << ".. done" << endl;
    MyString name;
    evalOption(name, "","",arg_referenzState);
    MyStringVec names;
    MyRegex rxSemikolon(";");
    names = split(rxSemikolon,name);
    referenceStates.resize(names.count());
    for(int i=0;i<names.count(); ++i)
    {
        cout << "reading reference state from "<< names[i] << flush;
        RandomOptimizerBASE::Handle opti1(new RandomOptimizerBASE,IINIT);
        opti1->peakNet(net).peakPool(pool).useIndependent(independent);
        if(opti1->loadState(names[i], pool, net)){
            return -1;
        };
        opti1->rebuildQAt(-1);
        referenceStates[i] = opti1->stateAt(-1);
        cout << " Q:="<<referenceStates[i] -> quality();

        cout << " .. done" << endl;
    }
}

// configure optimization data
cout << "building Optis " << endl;
SubRandomOptimizer                  *randomOpti1  = NULL;
SubRandomOptimizer                  *randomOpti2  = NULL;
GeneticOptimizer                    *geneticOpti = NULL;
randomOpti1     = new SubRandomOptimizer; // typespecific
randomOpti2     = new SubRandomOptimizer; // typespecific
geneticOpti     = GeneticOptimizerFactory::make(optiOpName);

RandomOptimizerBASE::Handle   opti(geneticOpti, IINIT);
RandomOptimizerBASE::Handle   subOpti(randomOpti1, IINIT);
RandomOptimizerBASE::Handle   finalOpti(randomOpti2, IINIT);
geneticOpti->subRO(subOpti); // typespecific
geneticOpti->finalRO(finalOpti); // typespecific

////////////////
long finalMaxIdleLoops     = 1000;
long finalMaxLoops         = 20000;
long finalRunTime          = 10;     // [min]
evalOption(finalMaxIdleLoops,1000,  "",arg_finalMaxIdleLoops);
evalOption(finalMaxLoops,    20000, "",arg_finalMaxLoops);
evalOption(finalRunTime,     10,    "",arg_finalRunTime);

finalRunTime *= 60; // [min] -> [sec]

// setze net und pool
opti->peakNet(net).peakPool(pool);
subOpti->peakNet(net).peakPool(pool);
finalOpti->peakNet(net).peakPool(pool);

fstream traceFile;
if(traceFileName!=""){
    traceFile.open(traceFileName,ios::out|ios::app);
    if(!traceFile){
        ErrMessage err;
        err << error << "could not open trace file \""
            << traceFileName <<"\" " << eom;
        return -1;
    }
    traceFile << "START_TRACE;"<<now<<endl;
    traceFile << "CROSSOVER_OPERATOR;"<<geneticOpti->crossoverOpName()<<endl;
    opti->debugStream(traceFile);
    subOpti->debugStream(traceFile);
    finalOpti->debugStream(traceFile);
} else {
    opti->debugStream(cerr);
    subOpti->debugStream(cerr);
    finalOpti->debugStream(cerr);
}
opti   ->debug(debugOptions).useIndependent(independent);
subOpti->debug(debugOptions).useIndependent(independent);
finalOpti->debug(debugOptions).useIndependent(independent);

// statistic arrays
int statTop=0;
int fail_statTop=0;
longVec    loops(nrOfRuns);
longVec    seedLoops(nrOfRuns);
longVec    longestLoops(nrOfRuns);
doubleVec  singleTime(nrOfRuns);
longVec    diffActions(nrOfRuns); diffActions.fill(0);

longVec    fail_loops(nrOfRuns);
doubleVec  fail_singleTime(nrOfRuns);
longVec    fail_diffActions(nrOfRuns); fail_diffActions.fill(0);
longVec    fail_asCount(nrOfRuns);
longVec    fail_linkCount(nrOfRuns);
doubleVec  fail_quality(nrOfRuns);
longVec    fail_seedLoops(nrOfRuns);
longVec    fail_longestLoops(nrOfRuns);

time_t restZeit    = statTime;
time_t usedTime    = -1;   // zeit fuer den vorhergehenden Durchgang
while(nrOfRuns>0 && ((statTime>0 && restZeit>0) || statTime <0)){
    // configure optimization behaviour
    opti->initState();
    if(optiFileName != ""){
        cout << "reading opti from "<<optiFileName<<endl;
        if( opti->loadState(optiFileName, pool, net))
            return -1;
    }
    if(!opti->configure(argv_iter))
        exit(-1);
    if(!subOpti->configure(argv_iter))
        exit(-1);
    if(!finalOpti->configure(argv_iter))
        exit(-1);

    // ade, schoene typen unabhaengigkeit..
    randomOpti2->maxIdleLoops(finalMaxIdleLoops);
    randomOpti2->maxLoop(finalMaxLoops);
    randomOpti2->runTime(finalRunTime);


    if(statisticRuns){
        opti->dumpFileName(tempFileName);
        opti->runTime(restZeit);
        opti->maxLoop(10000000); // sehr gross, nie erreicht
    }
    // starte den Optimizer neu.
    opti->restartRun();
    if(action == initState){
        cout << "init.... " << endl;
        opti->initState();
    } else {
        cout << "optimize.... " << endl;
        time_t startT = time(NULL);
        opti->optimize();
        time_t endT   = time(NULL);
        usedTime = endT - startT;
    }
    //
    nrOfRuns --;
    restZeit -= usedTime;
    if(restZeit < 10 ) restZeit = 0;
    //  ignoriere den letzten Durchlauf, wenn er
    //  durch TIMEOUT beeendet wurde.
    //  Der letzte Lauf verfälscht sonst die Statistik.
    //Ausnahme: Der erste Lauf hat die gesammte Zeit verbraucht!
    //  Dann ist dies das Ergebnis und darf nicht ignoriert werden.
    //pseudo code:
    //  wenn (statistik  UND entweder (
    //          erster Lauf
    //        ODER
    //          nicht durch TIMEOUT beendet ))
```

```
        if(statisticRuns
           &&  (
                   // der erste Durchlauf
                   ( fail_statTop + statTop == 0 )  ||
                   // nicht durch TIMEOUT beendet.
                   (opti->exitReason() != "max time reached" && restZeit > 0)
                   )
           )
        {
            const double epsilon = 1e-4;
            bool failed = false;
            int actions = 0;

            if ( referenceStates.count() > 0 ){
                failed = ((referenceStates[0]->quality()
                           - opti->stateAt(-1)->quality()) > epsilon);
                actions = MAXINT;
                for(int i=0;i<referenceStates.count(); ++i)
                {
                    StateDiff diff( opti->stateAt( -1 ),  referenceStates[i]);
                    diff.finalizePath();
                    actions = min(diff.atomDistance(),actions);
                }
            } else {
                failed = ((opti->endQ()
                           - opti->stateAt(-1)->quality()) > epsilon);
            }
            if(!failed){
                loops[statTop]       = opti->loopCount();
                singleTime[statTop]  = usedTime;
                diffActions[statTop] = actions;
                longestLoops[statTop]= geneticOpti->longestCountThread();
                seedLoops[statTop]   = geneticOpti->seedLoopThread();
                statTop ++;
            } else {
                fail_loops[fail_statTop]     =  opti->lastQChanged();
                fail_quality[fail_statTop]   =  opti->stateAt(-1)->quality();
                fail_asCount[fail_statTop]   =  opti->stateAt(-1)->usedCount();
                fail_linkCount[fail_statTop] =  opti->stateAt(-1)->usedLinks();
                fail_singleTime[fail_statTop]=  usedTime;
                fail_diffActions[fail_statTop] = actions;
                fail_longestLoops[fail_statTop]=  geneticOpti->longestCountThread();
                fail_seedLoops[fail_statTop] =  geneticOpti->seedLoopThread();
                fail_statTop ++;
            }
        } else {
            opti->dumpState();
        }
    }
    if(statisticRuns){
        WriteBlock block(out,"STATISTIC_DATA");
        for(int i = 0; i< statTop;i++){
            out<< i+1
               << "   S"
               << "  " << loops[i]
               << "  " << singleTime[i]
               << "  " << diffActions[i]
               << "  " << longestLoops[i]
               << "  " << seedLoops[i]
               << endl ;
        }
        for(int i = 0; i< fail_statTop;i++){
            out<< statTop+i+1
               << "   F"
               << "  " << fail_loops[i]
               << "  " << fail_singleTime[i]
               << "  " << fail_quality[i]
               << "  " << fail_asCount[i]
               << "  " << fail_linkCount[i]
               << "  " << fail_diffActions[i]
               << "  " << fail_longestLoops[i]
               << "  " << fail_seedLoops[i]
               << endl;
        }

        WriteBlock block2(out,"STATISTIC");
        showStat(out, "LOOPS", loops,statTop);
        showStat(out, "TIME",singleTime ,statTop);
        showStat(out, "ACTIONS",diffActions ,statTop);
        showStat(out, "SEED_LOOPS",    seedLoops,statTop);
        showStat(out, "LONGEST_LOOPS", longestLoops,statTop);

        showStat(out, "FAIL_AS", fail_asCount,fail_statTop);
        showStat(out, "FAIL_LINK",fail_linkCount ,fail_statTop);
        showStat(out, "FAIL_Q",fail_quality,fail_statTop);
        showStat(out, "FAIL_ACTIONS",fail_diffActions ,fail_statTop);
        showStat(out, "FAIL_LOOPS", fail_loops,fail_statTop);
        showStat(out, "FAIL_TIME",fail_singleTime ,fail_statTop);
        showStat(out, "FAIL_SEED_LOOPS", fail_seedLoops,fail_statTop);
        showStat(out, "FAIL_LONGEST_LOOPS", fail_longestLoops,fail_statTop);
        unlink(tempFileName); // unlink tempfile
    }
    traceFile << "END_TRACE;" << now << endl;
    return 0;
};
```

N.3 Quelltext: RandomOptimizer.h

```
#ifndef __RandomOptimizer_H__
#define __RandomOptimizer_H__  1
//
//
// $Id: RandomOptimizer.h,v 1.4 2000/05/29 16:27:14 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "Protein.h"
#include "PeakPool.h"
#include "PeakNet.h"
#include "Context.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizerBASE.h"

////////////////////////////////////////////////////////////////
class StateIndependent;
class State;
class LinkBuilder;
class  RandomOptimizerBASE;
////////////////////////////////////////////////////////////////
class RandomOptimizer;
class RandomOptimizer : public RandomOptimizerBASE
{
public:
    typedef RandomOptimizer Self;
    typedef IMngObject< RandomOptimizerBASE > Handle;

    virtual ~RandomOptimizer(void);
    RandomOptimizer(void);

//////////////////////////////////////
    ////////////////////////// ACTIONS
    //: fill all empty Positions with random entries
```

```
    //  added entries must be aquirable!
    void randomFillAt(int seqIndex);
    //: exchange several CX ( n = itsMaxSwapItemRounds)
    void swapItemsAt(int seqIndex);
    //:
    void doNothingAt(int UNUSED(seqIndex));
    //:
    void mergeFromBestAt(int seqIndex);
    //: erzeuge eine neue Sequenz mit zufallswerten.
    void randomGenerateAt(int seqIndex);
    //: tausche eine einzelne AS aus (eine Pos)
    void singleExchangeAt(int seqIndex);
    //: bewegt den CX von POS1 nach POS2 und setzt einen neuen CX auf POS1
    void movePosAt(int seqIndex);
    //:
    void singleUpgradeAt(int seqIndex);
    //: move linked ranges in sequence
    void moveLinkedRangeAt(int seqIndex);

////////////////////////////////////////
    //: führe die eigentliche Optimierung durch.
    virtual void optimize(void);
    //: soll optimize beendet werden?
    // INHERIT virtual bool abbruchbedingung(void);
    //: bewerte die Arena.
    void evaluateArena(void);

    static void  registerOptions(CmdLine& cmd);
    virtual bool configure(CmdLineArgIter& cmdIter);
    virtual void restartRun(void);

////////////////////////////////////////
    inline RandomOptimizer&   setTTLBorder(void);
    inline bool               inProtectedTTLRange(int t) const;

    int   startTTL(void)const;
    RandomOptimizer&   maxTTL(int i);
    int   maxTTL(void)const;
    RandomOptimizer&   minTTL(int i);
    int   minTTL(void)const;
////////////////////////////////////////
    //:
    bool loadProfile(const MyString& name);
    bool loadProfile(istream& is );

    RandomOptimizer&   vetoPeriod(long i);
    long   vetoPeriod(void)const;

    RandomOptimizer& thresholdDecayRate(double d);
    double thresholdDecayRate(void)const;

    //: start value for qThreshold
    RandomOptimizer& initialQThreshold(double d);
    double initialQThreshold(void)const;

    //: current value for qThreshold
    RandomOptimizer& qThreshold(double d);
    double qThreshold(void)const;

    ////////////////////
    //: laed den config teil des Dumps
    virtual bool loadConfig(istream& is );
    virtual void dumpConfig(ostream& os);
protected:
    virtual bool abbruchbedingung_qualityChange(void);

    virtual RandomOptimizer& debugParams(void);
    RandomOptimizer& debugTraceActions(void);
    RandomOptimizer& debugTraceActionTableChange(void);
    RandomOptimizer& debugTraceQ(void);
    RandomOptimizer& debugTraceEachQ(void);

    static const int     itsProfileCountDefault;
    static double       *itsCurrProfile;
    static int           itsProfileIndex;

    static double        itsProfileChangeDefault[];
    static const double  itsPctProfilesDefault[];
    static double        itsMaxProfileDefault[];

    static int           itsProfileCount;
    static double       *itsProfileChange;
    static double       *itsPctProfiles;
    static double       *itsMaxProfile;

    //: possible actions
    enum sequenceChangeActions {
        swapItems,       singleExchange,
        crossOver,       randomGenerate,
        mergeFromBest,   doNothing ,
        moveLinkedRange, removeRange,
        singleUpgrade,   movePos,

        countChangeActions  // counter for actions
    };

    //: name of actions
    static const char *  theActionName [];

    //: performance statistics table
    static long          itsActionStat[];
    static int           theGroupCount;
    enum statGroups {
        statEnter, statSuccess, statFailed, statFailed2 ,statImproved,
        statLast
    };

    sequenceChangeActions chooseAction(void);
    static const char*    action2Name(sequenceChangeActions  a);

    void         nextActionProfile(void);
    static void  calcProfiles(void);

    static void  dumpStatTable(ostream& os);
    static long  statCounter(sequenceChangeActions a, statGroups group);
    static void  incStatCounter(sequenceChangeActions a, statGroups group);

    // debug helper
    virtual void invariante_checkAllCacheMembers(
            int  seqIndex,
            bool checkUnlockedToo = true
            )const;

    // find a CX which uses the same Basepeak as the curent one in position pos.
    Context::Handle   singleUpgradeAtPos(int seqIndex,int pos);

    ////////////////////////// CONFIG DATA
    //:  max loop time to live for a State object.
    int    itsCurrMaxTTL;
    //:  min loop time to live for a State object.
    int    itsCurrMinTTL;
    //:  last ttl value which protects a State object
    //   objects in range [maxTTL .. border] may have any quality
    //   they are not replaced!
    //   border = maxTTL - minTTL + 1
    int    itsCurrTTLBorder;   // internal data!

    //: threshold factor for quality
    double itsQThreshold;
    //: initial threshold factor for quality
    double itsInitialQThreshold;
    //: decay factor for itsQThreshold
    double itsThresholdDecayRate;
    ////////////////////////// INTERNAL DATA
    Array< sequenceChangeActions > itsLastAction;

};
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//
inline RandomOptimizer&   RandomOptimizer::setTTLBorder(void)
{
    itsCurrTTLBorder = itsCurrMaxTTL- itsCurrMinTTL+1;
    return *this;
}
inline bool RandomOptimizer::inProtectedTTLRange(int t) const
```

```
{
    return (t >= itsCurrTTLBorder);
}

////////////////////////////////////////////////////
//
inline RandomOptimizer&   RandomOptimizer::minTTL(int ttl){
    itsCurrMinTTL = ttl;  setTTLBorder();
    return *this;
};
inline int   RandomOptimizer::minTTL(void)const{
    return itsCurrMinTTL;
};
////////////////////////////////////////////////////
//
inline RandomOptimizer&   RandomOptimizer::maxTTL(int ttl){
    itsCurrMaxTTL = ttl;  setTTLBorder();
    return *this;
};
inline int   RandomOptimizer::maxTTL(void)const{
    return itsCurrMaxTTL;
};
////////////////////////////////////////////////////
//
inline int   RandomOptimizer::startTTL(void)const{
    return itsCurrMaxTTL;
};
////////////////////////////////////////////////////
//
inline RandomOptimizer& RandomOptimizer::qThreshold(double d){
    itsQThreshold     = d;
    return *this;
};
inline double RandomOptimizer::qThreshold(void)const{
    return itsQThreshold;
};
////////////////////////////////////////////////////
//
inline RandomOptimizer& RandomOptimizer::initialQThreshold(double d){
    itsInitialQThreshold     = d;
    return *this;
};
inline double RandomOptimizer::initialQThreshold(void)const{
    return itsInitialQThreshold;
};
////////////////////////////////////////////////////
//
inline double RandomOptimizer::thresholdDecayRate(void)const{
    return itsThresholdDecayRate;
};

inline RandomOptimizer& RandomOptimizer::thresholdDecayRate(double d){
    itsThresholdDecayRate = d;
    return *this;
};
////////////////////////////////////////////////////
//
inline long RandomOptimizer::vetoPeriod(void)const{
    return State::vetoPeriod();
};

//////////////////////////////////////////////////////////////
//
inline long  RandomOptimizer::statCounter(
        RandomOptimizer::sequenceChangeActions a,
        RandomOptimizer::statGroups group)
{
    return RandomOptimizer::itsActionStat[group*countChangeActions + a];
}
inline void  RandomOptimizer::incStatCounter(
        RandomOptimizer::sequenceChangeActions a,
        RandomOptimizer::statGroups group)
{
    RandomOptimizer::itsActionStat[group*countChangeActions + a]++;
}
//////////////////////////////////////////////////////////////
// avoids  warnings
inline ErrMessage& operator<<(
        ErrMessage& os, RandomOptimizer::sequenceChangeActions e)
{
    os << int( e );    return os;
}

#endif // __RandomOptimizer_H__
```

N.4 Quelltext: RandomOptimizer.cc

```
//
//
// $Id: RandomOptimizer.cc,v 1.12 2000/05/29 16:26:03 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include "NoDump.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"

#include "Cache.h"
#include "Protein.h"
#include "PeakPool.h"

#include "PeakNet.h"

#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"

#include "myiomanip.h"
#include "ParseBlock.h"

#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizerBASE.h"
#include "RandomOptimizer.h"
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
#include "RandomOptimizer.debug.h"

//////////////////////////////////////////////////////////////////
class StateIndependent;
class State;
class LinkBuilder;
class RandomOptimizerBASE;
class RandomOptimizer;
class GeneticRandomOptimizer;
class SubRandomOptimizer;

//////////////////////////////////////////////////////////////////
//
void   RandomOptimizer::restartRun(void)
{
    RandomOptimizerBASE::restartRun();
    qThreshold(initialQThreshold());
}

////////////////////////////////////////////////////
```

```
// (DEBUGGING helper)
//: teste ob alle erworbenen AS auch Mitglieder des Cache sind
//  wenn nicht, ist eine der INVARIANTE verletzt worden.
//
//////
// checkUnlockedToo   : true, die Invariante soll auch fuer
//                      die noch nicht erworbenene AS getestet werden.
//virtual
#ifdef PRODUCTION
inline
#endif // PRODUCTION
void RandomOptimizer::invariante_checkAllCacheMembers(
        int seqIndex,
        bool checkUnlockedToo  /* = true */
        )const
{
#ifdef PRODUCTION
    return;
#endif // PRODUCTION
    DEBUGTracedFrame;
    State &S = *(const_cast<Self*>(this)->stateAt(seqIndex));
    ActiveAccount active(S.account());

    ASSERT(1000 > S.asCount());// 1000 >> size of any reasonable assignable protein
    int stack[1000];
    int top = 0;

    for(int pos = 0; pos < S.asCount(); ++pos){
        if(!isNull(S.asAt(pos))){
            // wenn der CX weder erworben noch (test fuer nicht erworbene erforderlich)
            //   dann ignoriere diesen CX
            if(!S.asLockAt(pos).isAquired() && ! checkUnlockedToo)
                continue;
            if(!S.asListAt(pos)->contains(S.asAt(pos))){
                stack[top++] = pos+1;
            }
        }
    }
    if( top>0 ){
        ErrMessage err;
        err << error << "INVARIANT VIOLATION: assigned AS not in assigned ASList!"
            << " action was "<< action2Name(itsLastAction[ seqIndex ] )
            <<"\n";
        err << " ASNR[";
        for(int i=0; i<top; i++){
            err << stack[i];
            if(i+1<top) err << ", ";
        }
        err << "]\n";
    }
}
////////////////////////////////////////////////////////////////////
//
bool RandomOptimizer::abbruchbedingung_qualityChange(void)
{
    if(itsLastQChanged < itsLoopCount - itsMaxIdleLoops * 2
       && itsLastQ >= itsBest->quality()
       && itsQThreshold > 0.9999
       ){
        exitReason("no quality improvement for (2*maxIdleLoop) loops");
        return true;
    }
    return false;
}


////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
// RandomOptimizer
static CmdArgStr       arg_profileFileName('p',"profileFileName","filename", "profile file name" ,
   CmdArg::isOPTVALREQ);
static CmdArgFloat     arg_qThreshold('l',"qThreshold","float", "Quality Threshold (% 0.0-1.0 )" ,
   CmdArg::isOPTVALREQ);
static CmdArgFloat     arg_thresholdDecayRate('s',"thresholdDecayRate","float", "thresholdDecayRate"
   , CmdArg::isOPTVALREQ);
static CmdArgInt       arg_vetoPeriod('s',"vetoPeriod","count", "vetoPeriod (small positive
   integer)" , CmdArg::isOPTVALREQ);
static CmdArgInt       arg_minTTL('s',"minTTL","count", "min loops to live" , CmdArg::isOPTVALREQ);
static CmdArgInt       arg_maxTTL('s',"maxTTL","count", "max loops to live" , CmdArg::isOPTVALREQ);

////////////////////////////////////////////////////////////
//: no of profiles
const int     RandomOptimizer::itsProfileCountDefault = 4;

//: threshold for profile change
//  change to second  profile if(itsQThreshold>itsProfileChange[0])
double     RandomOptimizer::itsProfileChangeDefault[/* profileCount */] = {
    0.6  , // change from profile 0 to 1
    0.8  , // change from profile 1 to 2
    0.95 , // change from profile 2 to 3
    1.0001 // 1.001 == means never
};
const double  RandomOptimizer::itsPctProfilesDefault[/* profileCount */]  =
{
////////////////////////////////////////////////////////////
//              swapItems,      crossOver,  mergeFromBest, moveLinkedRange, singleUpgrade
//                 singleExchange, randomGenerate,    doNothing,   removeRange   movePos

/* 0.0  .. 0.6  */    5.0,    92.5,    0.5,    0.0,    0.0,  0.0,   2.0, 0.01, 0.01, 0.01,
/* 0.6  .. 0.8  */   70.0,    26.5,    0.5,    0.0,    0.0,  0.0,   3.0, 0.01, 0.01, 0.01,
/* 0.8  .. 0.95 */   86.0,     8.5,    0.5,    0.0,    0.0,  0.0,   5.0, 0.01, 0.01, 0.01,
/* 0.95 .. 1    */   86.0,     4.5,    0.5,    0.0,    0.0,  0.0,   9.0, 0.01, 0.01, 0.01
};
double  RandomOptimizer::itsMaxProfileDefault[/* profileCount */] = {
/* range 1      */  101.0,    0.0,    0.0,    0.0,    0.0,  0.0,   0.0,  0.0,  0.0, 0.0,
/* range 2      */    0.0,    0.0,    0.0,    0.0,    0.0,  0.0,   0.0,  0.0,  0.0, 0.0,
/* range 3      */    0.0,    0.0,    0.0,    0.0,    0.0,  0.0,   0.0,  0.0,  0.0, 0.0,
/* range 4      */    0.0,    0.0,    0.0,    0.0,    0.0,  0.0,   0.0,  0.0,  0.0, 0.0
};
//////////////////////////
// initialize all to default
int      RandomOptimizer::itsProfileCount  = RandomOptimizer::itsProfileCountDefault;
double  *RandomOptimizer::itsProfileChange = RandomOptimizer::itsProfileChangeDefault;
double  *RandomOptimizer::itsPctProfiles
        = const_cast<double *>(RandomOptimizer::itsPctProfilesDefault);
double  *RandomOptimizer::itsMaxProfile    = RandomOptimizer::itsMaxProfileDefault;

const char * RandomOptimizer::theActionName [] = {
    "swapItems",
    "singleExchange",
    "crossOver",
    "randomGenerate",
    "mergeFromBest" ,
    "doNothing",
    "moveLinkedRange",
    "removeRange",
    "singleUpgrade",
    "movePos",
    NULL
};

int      RandomOptimizer::theGroupCount = RandomOptimizer::statLast;
long     RandomOptimizer::itsActionStat[
       RandomOptimizer::statLast *  RandomOptimizer::countChangeActions ] = {
/* enter   */   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,// enter
/* success */   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,// success
/* failed  */   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,// failed
/* fail2   */   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,// fail2
/* improve */   0,  0,  0,  0,  0,  0,  0,  0,  0,  0 // improve
};

double *RandomOptimizer::itsCurrProfile   = RandomOptimizer::itsMaxProfile;
int     RandomOptimizer::itsProfileIndex  = 0;


////////////////////////////////////////////////////////////////////
RandomOptimizer::~RandomOptimizer(void){
    dumpStatTable(cout);
}
////////////////////////////////////////////////////////////
RandomOptimizer::RandomOptimizer(void)
: RandomOptimizerBASE(),
  itsCurrMaxTTL(10),itsCurrMinTTL(2), itsCurrTTLBorder(0),
  itsLastAction(4)
```

```
{
    setTTLBorder();
    if(RandomOptimizer::itsMaxProfile[0] == 101.0){
        RandomOptimizer::calcProfiles();
    }

    itsQThreshold = 0.8;
    itsInitialQThreshold = 0.8;
    itsThresholdDecayRate = 0.0005;
};
////////////////////////////////////////////////////////////
void RandomOptimizer::calcProfiles(void)
{
    for(int n=0; n<itsProfileCount ; n++){
        //*itsCurrProfile     =  (*itsPctProfiles)   + countChangeActions * n;
        double *dest       =  &itsMaxProfile[countChangeActions  * n];
        const double *src  =  &itsPctProfiles[countChangeActions * n];
        *dest = *src;
        for(int i=1; i<countChangeActions; i++){
            dest[i] = dest[i-1] + src[i];
        }
    }
}
////////////////////////////////////////////////////////////
//
void RandomOptimizer::nextActionProfile(void){
    const double* end = &itsMaxProfile[0] + itsProfileCount * countChangeActions;
    itsCurrProfile  = itsMaxProfile + itsProfileIndex * countChangeActions;

    if(itsCurrProfile + countChangeActions < end){
        itsProfileIndex ++;
        itsCurrProfile  = itsMaxProfile + itsProfileIndex * countChangeActions;
        debugTraceActionTableChange();
    }
};
////////////////////////////////////////////////////////////
void RandomOptimizer::registerOptions(CmdLine& cmd){
    static bool done = false;
    RandomOptimizerBASE::registerOptions(cmd);
    if(!done){
        done = true;
        cmd << arg_profileFileName
            << arg_qThreshold
            << arg_thresholdDecayRate
            << arg_vetoPeriod
            << arg_minTTL
            << arg_maxTTL
            ;
    }
};
////////////////////////////////////////////////////////////////////
//
bool  RandomOptimizer::configure(CmdLineArgIter& cmdIter){
    long     count;
    double   d;
    MyString s;

    if(!RandomOptimizerBASE::configure(cmdIter))
        return false;
    if(givenOption(arg_profileFileName)){
        evalOption(s,"","",arg_profileFileName);
        if(s != ""){
            if(loadProfile( s ))
                return false;
        }
    }
    if(givenOption(arg_maxTTL)){
        evalOption(count,2,"",arg_maxTTL);
        maxTTL( count );
    }
    if(givenOption(arg_minTTL)){
        evalOption(count,2,"",arg_minTTL);
        minTTL( count );
    }
    if(givenOption(arg_qThreshold)){
        evalOption(d,0.3,"",arg_qThreshold);
        initialQThreshold( d );
        qThreshold( d );
    }
    if(givenOption(arg_thresholdDecayRate)){
        evalOption(d,0.3,"",arg_thresholdDecayRate);
        thresholdDecayRate( d );
    }
    if(givenOption(arg_vetoPeriod)){
        evalOption(count,2,"",arg_vetoPeriod);
        vetoPeriod( count );
    }
    return true;
}

////////////////////////////////////////////////////
//
RandomOptimizer&  RandomOptimizer::vetoPeriod(long i){
    if(protein()!=NULL){
        if(i > protein()->length() / 2){
            ErrMessage err;
            err << warning
                << "State::vetoPeriod set very high (more than (nr of as / 2))\n"
                <<"\t new vetoPeriod="<< i
                <<",  current asCount="<< protein()->length()
                << eom;
        }
    }
    State::vetoPeriod(i);
    return *this;
};
////////////////////////////////////////////////////
//
void RandomOptimizer::optimize(void)
{
    Array<bool> usedPos(sequenceCount());
    debugParams();
    debugTraceQ();
    itsEndTime = time(NULL) + runTime();
    itsLastAction.fill(doNothing);
    for(int seqIndex = -1; seqIndex < sequenceCount(); ++seqIndex){
        State &S = *stateAt(seqIndex);
        ActiveAccount active(S.account());
        if( S.usedCount () == 0){
            randomInitAt(seqIndex);
            //werfe nicht erwerbbare raus == leere die Position
            if( RandomOptimizer::itsCanRemoveNotLocked )
                removeNotLockedAt(seqIndex);
            // fuelle die Sequenz mit erwerbbaren CX auf
            if(  RandomOptimizer::itsCanFillEmpty )
                randomFillAt(seqIndex);

            evaluateAt(seqIndex);
            // werfe nicht verknuepfte AS raus == leere die Position
            // muss nach dem ersten ::evaluateAt geschehen,
            // da sonst die Links noch nicht gebildet wurden!
            if( RandomOptimizer::itsCanRemoveNotLinked ){
                removeNotLinkedAt(seqIndex);
                evaluateAt(seqIndex);
            }
        }
        rebuildQAt(seqIndex);
        invariante_checkAllCacheMembers(seqIndex);
    }
    if( itsBest->usedCount () == 0){
        itsBest->copyFrom(*itsArena[0]);
    }
    rebuildBestQ();
    while(itsQThreshold > itsProfileChange[itsProfileIndex])
        nextActionProfile();
    // force dump in first loop
    itsLastTraceCount = itsLoopCount-itsMaxTraceInterval-1;
    itsIsImproved     = false;
    debugTraceQ();
    debugTraceActionTableChange();
#ifdef DEBUG_TRACE_CALCULATION_Q
    unsigned long long  actionCount = 0;
#endif // DEBUG_TRACE_CALCULATION_Q
    while( ! abbruchbedingung() ){
```

```
        // fuer jede Sequenz.
        usedPos.fill(false);
        for(int seqIndex = 0; seqIndex < sequenceCount(); ++seqIndex){
            double oldQ = itsArena[seqIndex]->quality();
            // ignore already used sequence positions (from cross over),
            // don't change it again
            if(! usedPos[seqIndex] ){
                invariante_checkAllCacheMembers(seqIndex);
                //   waehle aktion aus
                sequenceChangeActions action = chooseAction();
                itsLastAction[ seqIndex ] =action;
#ifdef DEBUG_TRACE_CALCULATION_Q
                rebuildQAt(seqIndex);
                if(abs(stateAt(seqIndex)->quality()-oldQ)>1e-3){
                    (debugStream())
                        << "WRONG_Q; initial q wrong;"
                        <<" old=" << oldQ
                        <<" new="<<stateAt(seqIndex)->quality()
                        <<" (o-n)="<<(oldQ-stateAt(seqIndex)->quality())
                        << endl;
                }
                actionCount ++;
                {
                    const char sep = ';';
                    (debugStream())
                        << "NEXT_ACTION"
                        <<sep <<"#"<< actionCount
                        <<sep << action2Name(itsLastAction[seqIndex])
                        <<sep << "index="<<seqIndex+1
                        <<sep << "Q="<< oldQ
                        <<sep << "ttl="<< stateAt(seqIndex)->getTTL()
                        <<sep << "nrOfUsed="<< stateAt(seqIndex)->usedCount()
                        << endl;
                }
#endif // DEBUG_TRACE_CALCULATION_Q
                switch (action){
                case doNothing: { //   wenn AKTION == tue Nichts
                    // just in case there is some bookkeeping or debugging.
                    doNothingAt(seqIndex);
                    break;
                }
                case singleExchange :{ // high probability
                    singleExchangeAt(seqIndex);
                    break;
                }
                case movePos :{ // high probability
                    movePosAt(seqIndex);
                    break;
                }
                case singleUpgrade :{ // high probability
                    singleUpgradeAt(seqIndex);
                    break;
                }
                case swapItems :{ // highest probability
                    swapItemsAt(seqIndex);
                    break;
                }
                case crossOver :{   // low probability
                    // cout << "crossOver["<<seqIndex<<"]"<<endl;
                    if(seqIndex>sequenceCount()-4) {seqIndex--; continue; }

                    // suche best, second best, worst, second worst
                    double bestQ[2];
                    double worstQ[2];
                    int    bestI[2];
                    int    worstI[2];
                    int count=0;

                    bestQ[0]  = itsArena[seqIndex]->quality();
                    bestQ[1]  = 0.0;
                    worstQ[0] = MAXDOUBLE;
                    worstQ[1] = MAXDOUBLE;

                    bestI[0]  = seqIndex;  bestI[1]  = -1;
                    worstI[0] = -1;        worstI[1] = -1;
                    intVec used(sequenceCount());
                    used.fill(0);
```



```
                    for(int i=seqIndex; i< sequenceCount();i++){
                        if(!usedPos[i]){
                            count++; // first loop, count nr of unused
                            if(itsArena[i]->quality() > bestQ[0]){
                                // and store the new one
                                bestQ[0] = itsArena[i]->quality();
                                bestI[0] = i;
                            }
                        }
                    }
                    if(bestI[0]!=-1) used[bestI[0]] = 1;
                    for(int i=seqIndex; i< sequenceCount();i++){
                        if(!usedPos[i]){
                            if( used[i]==0 &&
                                itsArena[i]->quality() <= worstQ[0])
                            {
                                // and store the new one
                                worstQ[0] = itsArena[i]->quality();
                                worstI[0] = i;
                            }
                        }
                    }
                    if(worstI[0]!=-1) used[worstI[0]] = 1;
                    for(int i=seqIndex; i< sequenceCount();i++){
                        if(!usedPos[i]){
                            if( used[i] == 0  &&
                                itsArena[i]->quality() >= bestQ[1] )
                            {
                                // and store the new one
                                bestQ[1] = itsArena[i]->quality();
                                bestI[1] = i;
                            }

                        }
                    }
                    if(bestI[1]!=-1) used[bestI[1] ] = 1;
                    for(int i=seqIndex; i< sequenceCount();i++){
                        if(!usedPos[i]){
                            if( used[i] == 0 &&
                                itsArena[i]->quality() <= worstQ[1])
                            {
                                // and store the new one
                                worstQ[1] = itsArena[i]->quality();
                                worstI[1] = i;
                            }
                        }
                    }
                    if(worstI[1]!=-1) used[worstI[1]] = 1;
                    // cout << "opti::crossOverAt "<< worstI[0]<<", "<<worstI[1]<<", "
                    //      << bestI[0]<<", "<< bestI[1]<< endl;
                    // not enough Idle Pos
                    if(count < 4  ||
                       worstI[0] == worstI[1] ||  bestI[0]== bestI[1] ||
                       worstI[1] == -1
                       )  break;
                    crossOverAt(worstI[0],worstI[1], bestI[0], bestI[1]);

                    // beide sperren fuer Weiterbearbeitung
                    usedPos[worstI[0]] = true;
                    usedPos[worstI[1]] = true;
                    itsLastAction[worstI[0]] = crossOver;
                    itsLastAction[worstI[1]] = crossOver;

                    // fix seqIndex, if seqAt(seqIndex) was not a destination
                    if(!usedPos[seqIndex]){
                        seqIndex --;
                        continue; // foreach seqIndex
                    }

                    break;
                }
                case mergeFromBest  :{ // low probability
                    // ersetze diese Sequenz durch eine der gespeicherten
                    // besten Sequenzen und fuehre mit ihr einen
                    // 'singleExchange' aus.
                    if(!isNull(itsBest)){
```

```
                    mergeFromBestAt(seqIndex);
                }
                break;
            }
            case randomGenerate: { // low probability
                randomGenerateAt(seqIndex);
                break;
            }
            case moveLinkedRange: { // low probability
                moveLinkedRangeAt(seqIndex);
                break;
            }
            case removeRange: {  // low probability
                removeRangeAt(seqIndex);
                break;
            }
            case countChangeActions:; /* FALLTHROUGH */
            default :{
                ERRMESSAGE (err);
                err << error << "generated illegal action value="
                    << action << eom;
            }
            }; // switch action
        }// if not used Pos

#ifdef DEBUG_TRACE_CALCULATION_Q
        double newQ_pre = itsArena[seqIndex]->quality();
#endif // DEBUG_TRACE_CALCULATION_Q
        // bewerte Verteilung, muss vor UND nach fill ... geschehen
        // damit fill ... den Contexte erwerben kann.
        // andernfalls existieren gueltige, gerade erst eingefuegte, CX,
        // die aber noch nicht erworben wurden.
        // fill... fuegt dann weiter CX ein die manchmal mit den vorherigen
        // CX kollidieren wuerden, wenn diese schon erworben waeren.
        // ==> FEHLER , seq enthaelt CX die man nicht erwerben kann.
        evaluateAt(seqIndex);
#ifdef DEBUG_TRACE_CALCULATION_Q
        // wenn hier ein Fehler eintritt muss die einzelne Aktion
        // falsch sein.
        if( true  ||
            itsLastAction[seqIndex] == moveLinkedRange ||
            itsLastAction[seqIndex] == crossOver )
        {
            double newQ2_pre = itsArena[seqIndex]->quality();

            rebuildQAt(seqIndex);
            if(abs(itsArena[seqIndex]->quality() - newQ2_pre) > 1E-3){
                const char sep = ';';
                (debugStream())
                    << "WRONG_Q1"
                    <<sep<< action2Name(itsLastAction[seqIndex])
                    <<sep<< seqIndex+1
                    <<sep<<"old="<< oldQ
                    <<sep<<"new1(before eval)="<< newQ_pre
                    <<sep<<"new2(wrong)*="<< newQ2_pre
                    <<sep<<"new3(rebuild)*="<< itsArena[seqIndex]->quality()
                    <<sep<<"delta(#3-#2)="
                    << (itsArena[seqIndex]->quality() - newQ2_pre)<< endl;
            }
        }
        {
            const char sep = ';';
            (debugStream())
                << "PREV_ACTION1"
                <<sep <<"#"<< actionCount
                <<sep << action2Name(itsLastAction[seqIndex])
                <<sep << "index="<<seqIndex+1
                <<sep << "oldQ="<< oldQ
                <<sep << "ttl="<< stateAt(seqIndex)->getTTL()
                <<sep << "nrOfUsed="<< stateAt(seqIndex)->usedCount()
                <<sep << "newQ="<<itsArena[seqIndex]->quality()
                << endl;
        }
#endif  // DEBUG_TRACE_CALCULATION_Q
        //werfe nicht erwerbbare raus == leere die Position
        if( RandomOptimizer::itsCanRemoveNotLocked )
            removeNotLockedAt(seqIndex);
        // fuelle die Sequenz mit erwerbbaren CX auf
        if(  RandomOptimizer::itsCanFillEmpty )
            randomFillAt(seqIndex);

#ifdef DEBUG_TRACE_CALCULATION_Q
        double newQ = itsArena[seqIndex]->quality();
#endif // DEBUG_TRACE_CALCULATION_Q
        // bewerte Verteilung
        evaluateAt(seqIndex);
        // werfe nicht verknuepfte AS raus == leere die Position
        // muss nach dem ersten ::evaluateAt geschehen,
        // da sonst die Links noch nicht gebildet wurden!
        if( RandomOptimizer::itsCanRemoveNotLinked ){
            removeNotLinkedAt(seqIndex);
            evaluateAt(seqIndex);
        }

#ifdef DEBUG_TRACE_CALCULATION_Q
        if( true  || // <-- disable action filter
            itsLastAction[seqIndex] == moveLinkedRange ||
            itsLastAction[seqIndex] == crossOver )
        {
            double newQ2 = itsArena[seqIndex]->quality();

            rebuildQAt(seqIndex);
            if(abs(itsArena[seqIndex]->quality() - newQ2) > 1E-3){
                const char sep = ';';
                (debugStream())
                    << "WRONG_Q2"
                    <<sep<< action2Name(itsLastAction[seqIndex])
                    <<sep<< seqIndex+1
                    <<sep<<"old="<< oldQ
                    <<sep<<"new1(before eval)="<< newQ
                    <<sep<<"new2(wrong)*="<< newQ2
                    <<sep<<"new3(rebuild)*="<< itsArena[seqIndex]->quality()
                    <<sep<<"delta(#3-#2)="
                    << (itsArena[seqIndex]->quality() - newQ2)<< endl;
            }
        }
        {
            const char sep = ';';
            (debugStream())
                << "PREV_ACTION2"
                <<sep <<"#"<< actionCount
                <<sep << action2Name(itsLastAction[seqIndex])
                <<sep << "index="<<seqIndex+1
                <<sep << "oldQ="<< oldQ
                <<sep << "ttl="<< stateAt(seqIndex)->getTTL()
                <<sep << "nrOfUsed="<< stateAt(seqIndex)->usedCount()
                <<sep << "newQ="<<itsArena[seqIndex]->quality()
                << endl;
        }
#endif  // DEBUG_TRACE_CALCULATION_Q
        if( oldQ<itsArena[seqIndex]->quality() ){
            incStatCounter(itsLastAction[seqIndex],statImproved );
        };
    }
    // auswerten.
    // behalte beste (n) Sequenzen
    // kopiere
    debugTraceQ();
    evaluateArena();
    debugTraceActions();
  }
  debugTraceQ();
  debugTraceFinalPopulationStat();
}
//////////////////////////////////////////////////////////////////
//: decide which objects should stay
void  RandomOptimizer::evaluateArena(void){
  // noch wissen wir nicht ob eine Verbesserung eingtreten ist.
  itsIsImproved = false;
  // please remember:  better quality means higher Q values!
  double midQ = 0.0;
  double sumQ = 0.0;
  int posBest = 0, posWorst = 0;
  for(int i=0; i< sequenceCount(); i++){
```

```
        itsArena[i]->nextEpoch();
        sumQ += itsArena[i]->quality();
        if(itsArena[i]->quality()<itsArena[posWorst]->quality()){
            posWorst = i;
        } else if(itsArena[i]->quality()>itsArena[posBest]->quality()){
            posBest = i;
        }
    }
    midQ = sumQ / (double)sequenceCount();
    // akzeptiere alle die besser als threshold sind
    // die anderen (Q < threshold) werden ersetzt
    if(itsLastQ>itsBest -> quality()){
        debugStream()
            << "LAST_Q;reset;old="<<itsLastQ
            <<";bestQ="<<itsBest -> quality()
            <<endl;
        itsLastQ = 0;
    }
    // keep this in sync with debugTraceEachQ()!
#ifdef  FOLLOW_MID_Q
    double  Q = midQ;
#elif defined(FOLLOW_BEST_Q)
    double  Q = itsBest->quality();
#endif   // FOLLOW_XXX
    // muss jedes mal neu berechnet werden damit FOLLOW_MID_Q funktioniert.
    double threshold = max(itsLastQ ,Q) * qThreshold();
    if(Q > itsLastQ){ // neuer wert ist besser
        itsLastQ = Q;
        itsLastQChanged = itsLoopCount;
    }
#ifdef DEBUG_CALCULATION_Q
    cout << "evaluateArena; lastQ="<< itsLastQ
         <<"; curr bestQ("<<posBest<<")="<< itsArena[posBest]->quality()
         <<"; curr worstQ("<<posWorst<<")="<< itsArena[posWorst]->quality()
         <<"; bestQ="<< itsBest->quality()
         <<"; threshold="<< threshold
         << endl;
#endif // DEBUG_CALCULATION_Q
    if(itsArena[posBest]->quality() > itsBest -> quality()){
        double bestQ = itsArena[posBest]->quality();
        // berechne den besten wert neu
        rebuildQAt(posBest);
        // gibt es einen Fehler in den Berechnungsroutinen ?
        if(itsArena[posBest]->quality() > itsBest -> quality()){
            itsIsImproved = true;
            itsBestASCount   = 0;
            itsBestLinkCount = 0;
#ifdef DEBUG_CALCULATION_Q
            cout << "BACKUP NEW BEST("<<posBest<<");oldBest="<<itsBest -> quality();
#endif // DEBUG_CALCULATION_Q
            debugTracePopulationStat();
            itsArena[posBest]->setTTL(startTTL());
            itsBest->copyFrom(*itsArena[posBest]);
#ifdef DEBUG_CALCULATION_Q
            cout << ";newQ="<<itsBest -> quality()<< endl;
#endif // DEBUG_CALCULATION_Q
        } else if(abs(itsArena[posBest]->quality() - itsBest->quality())>1e-4){
            // calculation error
            ERRMESSAGE(err);
            err << warning
                <<"loop=" << itsLoopCount
                <<" action=" << action2Name(itsLastAction[posBest])
                <<" wrong quality for SEQ#"<<posBest+1
                <<" before="<<bestQ
                <<" after="<< itsArena[posBest]->quality()
                <<" (b-a)="<<(bestQ - itsArena[posBest]->quality())
                <<eom;
        }
    }
    itsBest->setEpoch(itsLoopCount);
    if(itsBestASCount == 0)
        for(int as =0; as < itsBest->asCount(); as++){
            if(!isNull(itsBest->asAt(as))) itsBestASCount++;
        }
    if(itsBestLinkCount == 0)
        for(int link =0; link < itsBest->linkCount(); link++){
            if(!isNull(itsBest->linkAt(link))) itsBestLinkCount++;
        }
    if(! inProtectedTTLRange(itsArena[posWorst]->getTTL()) ){
#ifdef DEBUG_CALCULATION_Q
        double oldQ = itsArena[posWorst] -> quality();
#endif // DEBUG_CALCULATION_Q
        itsArena[posWorst]->copyFrom(*itsBest);
        itsArena[posWorst]->setTTL(startTTL()); // setze den Anfangswert.
        itsLastAction[posWorst] = mergeFromBest;
        // rebuildQAt(posWorst);
#ifdef DEBUG_CALCULATION_Q
        cout << "REPLACE("<<posWorst<<") FROM BEST ;alt Q="<<oldQ
             << ";new Q="<<itsBest -> quality()<<endl;
#endif // DEBUG_CALCULATION_Q
    }
    for(int i=0; i<sequenceCount(); i++) {
        // ersetze alle Sequenzen,
        //  A) mit einer TTL unterhalb 0
        //  B) die die Mindest-TTL hinter sich haben und unterhalb der
        //     Schwelle liegen.
        if( (itsArena[i]->quality() < threshold
             && ! inProtectedTTLRange(itsArena[i]->getTTL()) )
            || ( itsArena[i]->getTTL() <= 0 )
           )
        {
            if(1 || odd(i)){
#ifdef DEBUG_CALCULATION_Q
                double oldQ = itsArena[i]-> quality();
#endif // DEBUG_CALCULATION_Q
                itsArena[i]->copyFrom(*itsBest);
                //rebuildQAt(i);
#ifdef DEBUG_CALCULATION_Q
                cout << "REPLACE("<<i<<") FROM BEST ;alt Q="<< oldQ
                     << ";new Q="<< itsArena[i] -> quality()<<endl;
#endif // DEBUG_CALCULATION_Q
                itsLastAction[i] = mergeFromBest;
            } else {
                singleCrossOverAt(i, i, -1,true);
                evaluateAt(i);
                itsLastAction[i] = crossOver;
                //rebuildQAt(i);
            }
            itsArena[i]->setTTL(startTTL());
        }
    }
    if(itsLastQChanged < itsLoopCount - itsMaxIdleLoops){
        itsQThreshold = itsQThreshold + (itsThresholdDecayRate * (1 -itsQThreshold));
        if(itsQThreshold > itsProfileChange[itsProfileIndex]){
            nextActionProfile();
        }
    }
};

//////////////////////////////////////////////////////////////
//: fill all empty Positions with random entries
//  added entries must be aquirable!
void     RandomOptimizer::randomFillAt(int seqIndex)
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    CHECK_LOCK;
    // bilde den Array der leeren Positionen, zufaellig mischen.
    // find all empty Positions
    intVec  posVec( S.asCount());
    int     topPosVec = 0;
    // foreach in range [first assignable pos .. last]
    for(int pos = 1; pos < S.asCount(); ++pos){
        if(isNull(S.asAt(pos)) && S.canChangeAt(pos) ){
            posVec[topPosVec++] = pos;
        }
    }
    if(topPosVec == 0) return;

    Context::Handle cxNULL;
    // random merge == count*0.75 mal random swap
    random_shuffle(posVec, topPosVec);

    for(int pos = 0; pos < topPosVec; ++pos){
```

```
        int asIndex = posVec[pos];
        if( S.asListAt(asIndex)->length() > 0 ){
            ContextLock     currASLock;
            //suche einen erwerbbaren CX aus dem Pool fuer diese Position aus (RANDOM)
            const ContextList &currList = *S.asListAt(asIndex);
            int count = itsMaxSeekAquirablePos; // -3
            ContextList::Iterator iter = currList.last();
            // die Liste sollte mindestens eine Moeglichkeit enthalten.
            // suche einen iterator im bereich [0 .. list.length-1]
            do {
                count--;
                int r = longInRange0(currList.length()-1);
                // jump to the  r-th  iterator, or wrap around
                while(r-- > 0){  --iter; if(!iter) iter = currList.last(); }
            } while( count >= 0 &&
                     (!iter || !currASLock.aquire(const_cast<Context::Handle&>(*iter))));
            // wenn ein akzeptabler Cx gefunden wurde
            if(iter && currASLock.isAquired()){
                // setze *iter an diese Position
                //S.asAt(asIndex) = *iter;
                currASLock.release(true);
                S.putASAt(asIndex,*iter);
                //S.buildASTypeQAt(pos);
                S.pushChange(asIndex);
                evaluateLocalAt(seqIndex,asIndex);
            } else {
                // ueberfluessig, ist NULL, s.o.;   S.putASAt(asIndex,cxNULL);
            }
        }
    }
}
////////////////////////////////////////////////////////////////////////////
//: exchange two CX ( n = itsMaxSwapItemRounds)
void RandomOptimizer::swapItemsAt(int seqIndex){
    DEBUGTracedFrame;
    incStatCounter(swapItems,statEnter);

    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    ASSERT(itsMaxSwapItemRounds<S.asCount());

#ifdef DEBUG_CALCULATION_Q
    cout << "swapItemsAt("<<(seqIndex+1)<<")";

#endif // DEBUG_CALCULATION_Q
    CHECK_LOCK;
    if( S.usedCount() < 5) {
#ifdef DEBUG_CALCULATION_Q
        cout<< " to few cx ! RETURN"<<endl;
#endif // DEBUG_CALCULATION_Q
        incStatCounter(swapItems,statFailed2);return;
    }
#ifdef DEBUG_CALCULATION_Q
    cout<< " select item positions."<<endl;
#endif // DEBUG_CALCULATION_Q
    bool done = false;
    for(int n=0; !done && n<itsMaxSwapItemRounds; n++){
        int trieCount = 10;
        int pos1=-1, pos2=-1;
        Context::Handle cx1;
        Context::Handle cx2;
        //: index einer AS [0 .. as count-1]
        int count = -1;
        intVec otherAS;
        do {
            // suche zwei nicht veraenderte positionen
            // die zweite darf leer sein
            // bedingung:  pos1 != pos2
            // sets cx1  == NULL if first not found
            // sets pos2 == -1   if second not found

            // int pos = S.findPos_inUse_anyWhere(); (???POS)
            // suche eine position mit cx im bereich [1..len-1]
            count = 10;
            do {
                pos1 = longInRange0(S.asCount()-2)+1;
            } while( ( !S.canChangeAt(pos1)
                    || S.changedAS().get(pos1)
                    || isNull(S.asAt(pos1)) )
                  && count-- > 0 );
            cx1 = S.asAt(pos1);
#ifdef DEBUG_CALCULATION_Q
            cout << "SEQUENCE "<< seqIndex+1 << " start: "<< pos1 << ",  ";
            cout << "CX1 is "<< ( !isNull(S.asAt(pos1))?"not null":"NULL" ) << endl;
#endif
            if(count != 0 && !isNull(S.asAt(pos1))) { // gueltiges cx1 gefunden.
                // findPos_withCX_notEqPos(cx,pos1)  // (???POS)
                otherAS = S.usedIndependent()->asForPeak(S.basePeakIDAt(pos1)-1);
#ifdef DEBUG_CALCULATION_Q
                cout << "BasePeak index1 at AS1("<<pos1<<")= "<< S.basePeakIDAt(pos1)-1 << endl;
                cout << "##Q before="<<S.quality()<<endl;
                cout << "AS("<<pos1<<") is "<< ( !isNull(S.asAt(pos1))?" not null":" NULL    " ) ;
                cout << " asQ="  << S.asQAt(pos1);
                cout << " lock=" << S.asLockAt(pos1).isAquired();
                cout << " linkQ-1=" <<((pos1-1>=0)?S.linkQAt(pos1-1):-1.0);
                cout << " linkQ="   <<((pos1<S.linkCount())?S.linkQAt(pos1):-1.0);
                cout << endl;
                cout << "OTHER AS  (1)          "<< otherAS<< endl;
#endif
                random_shuffle(otherAS);

#ifdef DEBUG_CALCULATION_Q
                cout << "OTHER AS (shuffle) "<<  otherAS<< endl;
#endif
                int n=0;
                for(n = 0; n < otherAS.count() && otherAS[n]!=pos1; n++){
                    pos2 = otherAS[n];
                    cx2  = S.asAt(pos2);
#ifdef DEBUG_CALCULATION_Q
                    cout << "SEQUENCE "<< seqIndex+1 << " end: "<< pos2 << ",  ";
                    cout << "CX2 is "<< (! isNull(S.asAt(pos2))?"not null":"NULL" ) << endl;
#endif
                    if( isNull(S.asAt(pos2))){
                        if(S.asListAt(pos2)->contains(cx1) )
                        {
                            done = true;
                            break; // breaks for(n<otherAS.count()),  akzeptiert.
                        }
#ifdef DEBUG_CALCULATION_Q
                        cout << "   ... cx1 not in list of pos2" << endl;
#endif
                        pos2 = -1;
                        continue;  // try next pos
                    }

                    const intVec& asV = S.usedIndependent()->asForPeak(
                            S.basePeakIDAt(pos2)-1);
#ifdef DEBUG_CALCULATION_Q
                    cout << "OTHER AS (2) :=    "<<  asV << endl;
#endif
                    if( -1 != binsearch(asV,pos2)
                        && S.asListAt(pos2)->contains(cx1)
                        && S.asListAt(pos1)->contains(cx2) )
                    {
                        done = true;
                        break; // breaks for (n<asCount),  gefunden!
                    }
#ifdef DEBUG_CALCULATION_Q
                    cout << "   ... cx1 not in list of pos2" << endl;
#endif
                    pos2 = -1; // mache pos2 ungueltig
                }
            }
            // solange eine position ungueltig ist und
            // die Maximalzahl der Fehlversuche nicht erreicht ist.
        }while( !done && (isNull(cx1) || pos2 == -1 ) && trieCount-- > 0 );
        // abbruch wenn kein swap Paar gefunden wurde.
        if( trieCount <= 0 || isNull(cx1)) {
            incStatCounter(swapItems,statFailed);return;
        }
//////////////
        // valid swap found.
        // now perform that swap.
#ifdef DEBUG_CALCULATION_Q
```

```
        cout << "SEQUENCE "  <<seqIndex+1
             << " swapItems("<< pos1<< " <-> " << pos2 << ')'<<endl ;
        cout << "##Q before="<<S.quality()<<endl;
        cout << "AS("<<pos1<<") is "<< ( !isNull(S.asAt(pos1))?" not null":" NULL    " ) ;
        cout << " asQ="  << S.asQAt(pos1);
        cout << " lock=" << S.asLockAt(pos1).isAquired();
        cout << " linkQ-1=" <<((pos1-1>=0)?S.linkQAt(pos1-1):-1.0);
        cout << " linkQ="   <<((pos1<S.linkCount())?S.linkQAt(pos1):-1.0);
        cout << endl;
        cout << "AS("<<pos2<<") is "<< ( !isNull(S.asAt(pos2))?" not null":" NULL    " ) ;
        cout << " asQ="  << S.asQAt(pos2);
        cout << " lock=" << S.asLockAt(pos2).isAquired();

        cout << " linkQ-1=" <<((pos2-1>=0)?S.linkQAt(pos2-1):-1.0);
        cout << " linkQ="   <<((pos2<S.linkCount())?S.linkQAt(pos2):-1.0);
        cout << endl;
#endif
        // erase the old value, make cx valid or NULL
        S.clearAS(pos1,true);
        S.clearAS(pos2,true);
        if(!isNull(cx1)){
#ifdef DEBUG_CALCULATION_Q
            cout << "   .. put cx1 at " << pos2 << "   cx1 is "
                 <<(isNull(cx1)?" NULL":" not null")<< endl;
#endif
            S.putASAt(pos2, cx1);        S.pushChange(pos2);
            evaluateLocalAt(seqIndex,pos2);
        }

        // lokal Linkq
        if(!isNull(cx2)){
#ifdef DEBUG_CALCULATION_Q
            cout << "   .. put cx2 at " << pos1 << "   cx2 is "
                 <<(isNull(cx1)?" NULL":" not null")<< endl;
#endif
            S.putASAt(pos1, cx2);        S.pushChange(pos1);
            evaluateLocalAt(seqIndex,pos1);
        }

//////////////
#ifdef DEBUG_CALCULATION_Q
        cout << "##Q after="<<S.quality()<<endl;
        cout << "AS("<<pos1<<") is "<< ( !isNull(S.asAt(pos1))?" not null":" NULL    " ) ;
        cout << " asQ="  << S.asQAt(pos1);
        cout << " lock=" << S.asLockAt(pos1).isAquired();
        cout << " linkQ-1=" <<((pos1-1>=0)?S.linkQAt(pos1-1):-1.0);
        cout << " linkQ="   <<((pos1<S.linkCount())?S.linkQAt(pos1):-1.0);
        cout << endl;
        cout << "AS("<<pos2<<") is "<< ( !isNull(S.asAt(pos2))?" not null":" NULL    " ) ;
        cout << " asQ="  << S.asQAt(pos2);
        cout << " lock=" << S.asLockAt(pos2).isAquired();

        cout << " linkQ-1=" <<((pos2-1>=0)?S.linkQAt(pos2-1):-1.0);
        cout << " linkQ="   <<((pos2<S.linkCount())?S.linkQAt(pos2):-1.0);
        cout << endl;
#endif
    }
    incStatCounter(swapItems,statSuccess);
}
////////////////////////////////////////////////////////////////////////
// (ACTION), keep this! maybe we need some debugging code later!
void     RandomOptimizer::doNothingAt(int UNUSED(seqIndex)){
   /* EMPTY */
}
//////////////////////////////////////////////////////////////////////
//
void     RandomOptimizer::mergeFromBestAt(int UNUSED(seqIndex)){
   /* EMPTY */
};
//////////////////////////////////////////////////////////////////////
//
void     RandomOptimizer::randomGenerateAt(int UNUSED(seqIndex)){
   /* EMPTY */
};
//////////////////////////////////////////////////////////////////////
//: move linked ranges in sequence
void     RandomOptimizer::moveLinkedRangeAt(int seqIndex)
{
    DEBUGTracedFrame;
    incStatCounter(moveLinkedRange,statEnter);
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
#ifdef moveLinkedRange_DEBUG_CALCULATION_Q
    double Q = S.quality();
    cout << "SEQUENCE["<< seqIndex+1 << "] moveLinked  Q="<<Q << endl;
#endif
    CHECK_LOCK;

    const int maxLinks = 4;
    intVec baseIDs1(maxLinks+1); // id of peaks for AS
    int    linkLen1;
    // TRIES:
    bool done = false;
    for(int r=0; !done && r<theMoveLinkedRangeMaxRounds; r++){
        // suche eine pos bei der ein gelinkter CX steht. (linkAt(pos)!=NULL)
        // suche eine zweite entsprechende Stelle im State, die
        //   alle cx ab pos1 koennen auch ab pos2 stehen.
        //   (pos2 in asForPeak(basePeakAt(pos1)))
        int pos1=-1;
        Context::Handle cx1;
        Context::Handle cx2;
        //: index einer AS [0 .. as count-1]
        int count = -1;
        intVec otherPos;

        ////////////////////
        // suche eine Position mit cx im bereich [1 .. linkCount()-1 ]
        // die 1 bis maxLinks nachfolgende Links hat.
        count = 10;
        do {// -1  +1 for [ 1 ...
            // -1     for  as -> link count
            // -1     for  count -> index
            // -1     for  at least one one neigbour
            //                V  V
            pos1 = longInRange0(S.asCount()-4)+1;
        } while( ( !S.canChangeAt(pos1)
                   || S.changedAS().get(pos1)
                   || isNull(S.asAt(pos1))
                   || isNull(S.linkAt(pos1)))
                 && count-- > 0 );
        cx1 = S.asAt(pos1);// erase the old value, make cx1 valid or NULL

        ////////////////////
#ifdef moveLinkedRange_DEBUG_CALCULATION_Q
        cout << "SEQUENCE "<< seqIndex+1 << " start: "<< pos1 << endl;
        cout << "CX1 is "<< ( !isNull(S.asAt(pos1))?" not null":" NULL" ) << endl;
#endif
        if(count > 0 && !isNull(S.asAt(pos1))) { // gueltiges cx1 gefunden
            // nr of links in from start point,  maximum==maxLinks .
            linkLen1 = S.countLinksInRange(pos1,maxLinks);
            S.baseIDsInRange(baseIDs1, pos1, linkLen1+1);
            otherPos = S.usedIndependent()->asForPeak(baseIDs1[0]);

            // schraenke den Bereich der moeglichen Positionen ein.
            int otherPosLen = otherPos.count(); // nr of alternative positions
            removeElement(otherPos,otherPosLen,pos1);
            int k = 1;
            bool f=true;
            for(k=1;
                ( f && (k < linkLen1+1)  &&
                  (f = subset(otherPos, otherPosLen, S.usedIndependent()->asForPeak(baseIDs1[k]),
   -k)));
                k++);
            if(!f) k--;   // fix k, it is off by one, if subset was not successfull
            linkLen1 = k - 1; // // asLen -> linkLen

            if(linkLen1 == 0 || otherPosLen == 0){
                // schon der erste Nachbar (in Position pos1+1)
                // hat keine gemeinsame zweite Position mit asAt(pos1)
                // naechster Versuch
                incStatCounter(moveLinkedRange,statFailed2);
                continue; //  for( t ) TRIES:
            }
            // es gibt [1..maxLinks+1] Nachbarn die  otherPosLen
            // gemeinsame andere Positionen haben. (otherPosLen >= 1)
```

```
            ASSERT(otherPosLen >= 1);

#ifdef moveLinkedRange_DEBUG_CALCULATION_Q
            cout << "BasePeak index1 at AS1("<<pos1<<")= "<< baseIDs1[0] << endl;
            cout << "length of link from AS1("<<pos1<<")="<< linkLen1 << endl;
            cout << "used length of OTHER AS             "<< otherPosLen << endl;
            if(otherPosLen < otherPos.count()) otherPos[otherPosLen] = -1;
            cout << "OTHER AS  (1)                       "<< otherPos<< endl;
#endif
            // gib allen Positionen die gleiche Chance
            random_shuffle(otherPos,otherPosLen);

#ifdef moveLinkedRange_DEBUG_CALCULATION_Q
            cout << "OTHER AS (shuffle)                  "<<  otherPos<< endl;
#endif

            int as = findFirstMatch(seqIndex,pos1, linkLen1+1, otherPos, otherPosLen);
            if( as == -1 ) continue; // for( t ) TRIES:

            copyRange(seqIndex,otherPos[as], pos1, linkLen1+1);

#ifdef moveLinkedRange_DEBUG_CALCULATION_Q
            cout << "moveLinkedRange("<<seqIndex<<") range["
                 << pos1 << " .. " << pos1+linkLen1<< "] --> ["
                 << otherPos[as] <<" .. " << otherPos[as] +linkLen1<< "]  peaks(";
            for(int i=0; i<linkLen1+1; i++){
                cout << baseIDs1[i] ;
                if(i+1<linkLen1+1) cout << ", " ;
            }
            cout    << ")" << endl;
#endif
            incStatCounter(moveLinkedRange,statSuccess);
            return ; // FINISH
        }
    }
    incStatCounter(moveLinkedRange,statFailed);
#ifdef moveLinkedRange_DEBUG_CALCULATION_Q
    cout << "SEQUENCE["<< seqIndex+1 << "] moveLinked  FAILED"<< endl;
#endif
};
////////////////////////////////////////////////////////////////////////////
void     RandomOptimizer::singleExchangeAt(int seqIndex)
{
    DEBUGTracedFrame;
    incStatCounter(singleExchange,statEnter);
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    // fuer N Zufallsereignisse {
    //   suche eine position aus (RANDOM)
    //      entsperre den CX der an dieser Pos vorhandenen ist. ==> prevCX
    //   suche einen CX aus dem Pool fuer diese Position aus (RANDOM)
    //       versuche N mal (=3) einen erwerbbaren CX zu finden ( != prevCx )
    //       danach akzeptiere irgendeinen.
    //       setze prevCX ein wenn kein anderer moeglich ist.
    //   wenn weiterreichen,  suche eine neue position aus (RANDOM)
    //   sperren den State fuer weitere Bearbeitung
    //}
    ASSERT(itsMaxSingleExchangeRandomPos<S.asCount());
#ifdef singleExchange_DEBUG_CALCULATION_Q
    cout << "SEQUENCE["<< seqIndex+1 << "] singleExchange" << endl;
    double Q = S.quality();
#endif  // singleExchange_DEBUG_CALCULATION_Q
    CHECK_LOCK;

    for(int n=0; n<itsMaxSingleExchangeRandomPos; n++){
        // suche eine nicht veraenderte Position, fuer die es CX gibt.
        //: index einer AS [1 .. as count-1]
        int pos = chooseAS(S);
#ifdef  SINGLE_EXCHANGE_FILL_EMPTY_POS
        // if no pos found and at least one empty position.
        if(pos == -1 && S.usedCount()< S.asCount()-1){
            // choose an empty as position
            int i=chooseEmptyAS(S);
            if(i!=-1 && i<S.asCount() && !isNull(S.asAt(i))) pos = i;
        }
#endif  // SINGLE_EXCHANGE_FILL_EMPTY_POS
        if(pos == -1){
            incStatCounter(singleExchange,statFailed); return;
        }

        Context::Handle prevCX = S.asAt(pos);
        S.clearAS(pos,true); // S.pushChange(pos);
#ifdef singleExchange_DEBUG_CALCULATION_Q
        cout << "SINGLE_EXCHANGE  CLEAR AS("<<pos<<")=" << (Q - S.quality()) << endl;
#endif // singleExchange_DEBUG_CALCULATION_Q
        ContextLock     currASLock;
        //suche einen **erwerbbaren** CX aus dem Pool fuer diese Position aus (RANDOM)
        const ContextList &currList = *S.asListAt(pos);
        //int count = min(itsMaxSeekAquirablePos,currList.length());
        // die Liste sollte mindestens eine Moeglichkeit enthalten.
        // versuche N mal (=3) einen neuen erwerbbaren CX zu finden ( cx != prevCx )
        // danach akzeptiere irgendeinen.
        const int smallStepSize = 5;
        int count = min((currList.length()-1)/smallStepSize,itsMaxSeekAquirablePos);
        if( count == 0 )
            count = currList.length()-1;
        ContextList::Iterator iter = currList.last();
        ASSERT(iter);
        int r = longInRange0(currList.length()-1); // range [0 .. arg]
        while(r-- > 0){  --iter; if(!iter) iter = currList.last(); }
        while( --count >= 0 &&
               ! currASLock.aquire(const_cast<Context::Handle&>(*iter)) )
        {
            int r = longInRange0(smallStepSize); // range [0 .. arg]
            while(r-- > 0){  --iter; if(!iter) iter = currList.last(); }
            // jump to the  r-th  iterator, or wrap around
        }
        if(currASLock.isAquired()){
            // fuege *iter an position pos ein, loesche die berechneten Daten
            currASLock.release(true); // evaluateLocalAt will aquire it

            S.putASAt(pos,*iter);S.pushChange(pos);
            evaluateLocalAt(seqIndex,pos);
#ifdef singleExchange_DEBUG_CALCULATION_Q
            cout << "SINGLE_EXCHANGE  PUT AT "<<pos << " =" << (Q - S.quality()) << endl;
#endif // singleExchange_DEBUG_CALCULATION_Q
            incStatCounter(singleExchange,statSuccess);
            if(SHOW_Q_TRACE) cout <<"END---"<< __FUNCTION__ << endl;
#ifdef singleExchange_DEBUG_CALCULATION_Q
            CHECK_QUALITY_STABLE(seqIndex)
#endif // singleExchange_DEBUG_CALCULATION_Q
            return ; // FINISH
        } else {
#ifdef SINGLE_EXCHANGE_RESTORE_OLD_IF_FAILED
            // restore the old value
            S.putASAt(pos,prevCX);S.pushChange(pos);
            evaluateLocalAt(seqIndex,pos);
#else // SINGLE_EXCHANGE_RESTORE_OLD_IF_FAILED
#ifdef singleExchange_DEBUG_CALCULATION_Q
            cout << "SINGLE_EXCHANGE  FAILED AT "<<pos << " =" << (Q - S.quality()) << endl;
#endif // singleExchange_DEBUG_CALCULATION_Q
            incStatCounter(singleExchange,statFailed2);
#endif // SINGLE_EXCHANGE_RESTORE_OLD_IF_FAILED
        }
    }
    incStatCounter(singleExchange,statSuccess);
    if(SHOW_Q_TRACE) cout <<"END---"<<__FUNCTION__ << endl;
}
////////////////////////////////////////////////////////////////////////////
//: bewegt einen CX von POS1 nach POS2 und setzt einen neuen CX auf POS1
void RandomOptimizer::movePosAt(int seqIndex)
{
    DEBUGTracedFrame;
    incStatCounter(movePos,statEnter);

    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    ASSERT(itsMaxSwapItemRounds<S.asCount());

#ifdef DEBUG_CALCULATION_Q
    cout << "movePosAt("<<(seqIndex+1)<<")";

#endif // DEBUG_CALCULATION_Q
    CHECK_LOCK;
```

```
    if( S.usedCount() < 5) {
#ifdef DEBUG_CALCULATION_Q
        cout<< " to few cx ! RETURN"<<endl;
#endif // DEBUG_CALCULATION_Q
        incStatCounter(movePos,statFailed2);return;
    }
#ifdef DEBUG_CALCULATION_Q
    cout<< " select item positions."<<endl;
#endif // DEBUG_CALCULATION_Q
    //: true, wenn die Kanidaten fuer die Aktion gefunden wurde.
    //      dh.wenn pos2 die Position fuer cx1 enthaelt
    //        und cx2 nach pos1 gesetzt wird wenn pos1 != -1 ist.
    bool done = false;
    for(int n=0; !done && n<itsMaxMoveItemRounds; n++){
        int trieCount = 10;
        int pos1=-1, pos2=-1;
        Context::Handle cx1;
        Context::Handle cx2;
        // Fuer Pos1 ist ein ContextLock ueberfluessig, cx1 bleibt
        // im State erworben, bis der Tausch wirklich ausgefuehrt wird.
        // Dagegen braucht cx2 einen Lock und muss geloescht und notfalls
        // restauriert werden.

        //: index einer AS [0 .. as count-1]
        int count = -1;
        intVec otherAS;
        do {
            // suche zwei nicht veraenderte positionen (pos1, pos2)
            // die pos2 darf leer sein (!isNull(asAt(pos1))
            // pos2 muss den CX cx1 aufnehmen koennen.
            // bedingung:  pos1 != pos2
            // sets cx1  == NULL if first not found
            // sets pos2 == -1   if second not found

            // suche eine belegte position im bereich [1..len-1]
            count = 10;
            do {
                pos1 = longInRange0(S.asCount()-2)+1;
            } while( ( !S.canChangeAt(pos1)
                       || S.changedAS().get(pos1)
                       || isNull(S.asAt(pos1)) )
                     && count-- > 0 );
            cx1 = S.asAt(pos1);// backup der alten Position
#ifdef DEBUG_CALCULATION_Q
            cout << "SEQUENCE "<< seqIndex+1 << " start: "<< pos1 << ",  ";
            cout << "CX1 is "<< ( !isNull(S.asAt(pos1))?"not null":"NULL" ) << endl;
#endif

            if(count != 0 && !isNull(S.asAt(pos1))) {
                // Eine gueltige position mit cx wurde gefunden. (pos1 und cx1)
                // gibt es eine zweite Position die cx1 aufnehmen kann?
                // suche erst den Vektor der AS-POS, die den gleichen Peak
                // aufnehmen koennen.
                otherAS = S.usedIndependent()->asForPeak(S.basePeakIDAt(pos1)-1);
                if(otherAS.count()<2){
                    // diese List enhaelt keine moegliche Alternative fuer cx1.
                    continue; // neue startposition (pos1) suchen
                }
#ifdef DEBUG_CALCULATION_Q
                cout << "BasePeak indexl at AS1("<<pos1<<")= "<< S.basePeakIDAt(pos1)-1 << endl;
                cout << "##Q before="<<S.quality()<<endl;
                cout << "AS("<<pos1<<") is "<< ( !isNull(S.asAt(pos1))?" not null":" NULL    " ) ;
                cout << " asQ="  << S.asQAt(pos1);
                cout << " lock=" << S.asLockAt(pos1).isAquired();
                cout << " linkQ-1=" <<((pos1-1>=0)?S.linkQAt(pos1-1):-1.0);
                cout << " linkQ="   <<((pos1<S.linkCount())?S.linkQAt(pos1):-1.0);
                cout << endl;
                cout << "OTHER AS  (1)          "<< otherAS<< endl;
#endif
#ifdef MOVE_POS_FAST_ALGO
                // suche eine andere Position (pos2) die den Context cx1
                // enthaelt. jede Position die den Context nicht enthaelt
                // wird aus der Liste entfernt.
                random_shuffle( otherAS );
                int top = otherAS.count()-1;
                do {
                    pos2 = otherAS[top];
                    if(pos2 != pos1 && S.asListAt(pos2)->contains(cx1)) break;
                } while(--top > -1);
                if( top == -1 ){
                    // diese Liste enhaelt keine moegliche alternative Position fuer cx1.
                    continue; // neue startposition (pos1) suchen
                }
                ASSERT(S.asListAt(pos2)->contains(cx1));
                S.clearAS(pos2,true);
               ////////////////////////////////////////////////////////
                // cx1 wird auf pos2 gesetzt.
                // jetzt wird nur noch ein neuer CX fuer pos1 gesucht.
                ContextLock     pos2ASLock;
                const ContextList &currList = *S.asListAt(pos1);
                const int smallStepSize = 5;
                int count = min((currList.length()-1)/smallStepSize,20);
                if( count == 0 )
                    count = min(3,currList.length()-1);
                // jump to some position in list
                int r = longInRange0(currList.length()-1);
                ContextList::Iterator iter = currList.last();
                while(r-- > 0){--iter; if(!iter) iter = currList.last(); }
                //   then move in smaller steps.
                while( --count >= 0 &&
                       (!iter
                        || S.asListAt(pos1)->contains(*iter)
                        || !pos2ASLock.aquire(const_cast<Context::Handle&>(*iter))))
                {
                    // jump to the  r-th  iterator, or wrap around
                    r = longInRange0(smallStepSize);
                    while(r-- > 0){--iter; if(!iter) iter = currList.last();}
                }
                if(iter && pos2ASLock.isAquired()){
                    // loese Lock, ::evaluateLocalAt will den cx erwerben.
                    pos2ASLock.release(true);
                    // *iter ist ein CX der auf Position pos1 gesetzt werden kann.
                    cx2 = *iter;
                    // gefunden
                    //   in cx1 ist der CX fuer pos2
                    //   in cx2 ist der CX fuer pos1
                    // beide koennen gleichzeitig erworben werden.
                    // (cx1 ist noch erworben, cx2 ist es nicht mehr )
#ifdef DEBUG_CALCULATION_Q
                    cout << "   ... cx2 for pos1 found" << endl;
#endif
                    done = true;
                    break; // for ( find po2 )
                } else {
                    // der alte Zustand in pos2 wird nicht wieder hergestellt.
                    // nach dem Verschieben befindet sich cx1 in pos2
                    // und pos1 bleibt leer.
#ifdef DEBUG_CALCULATION_Q
                    cout << "   ... cx2 for pos1 stays empty" << endl;
#endif
                    cx2 = Context::Handle();
                    done = true;
                    break; // for ( find po2 )
                }
////////////////////////////////////////////////////////////////////////
#endif //  MOVE_POS_FAST
////////////////////////////////////////////////////////////////////////
            }
            // schleife
            // solange eine position ungueltig ist und
            // die Maximalzahl der Fehlversuche nicht erreicht ist.
        }while( !done && (isNull(cx1) || pos2 == -1 ) && trieCount-- > 0 );
        // abbruch wenn kein swap Paar gefunden wurde.
        if( trieCount <= 0 || isNull(cx1) || !done) {
            incStatCounter(movePos,statFailed);
            return;
        }
//////////////
        // valid move found. now perform it.
        //   in cx1 ist der CX fuer pos2
        //   in cx2 ist der CX fuer pos1
        // beide koennen gleichzeitig erworben werden.
        // (cx1 ist noch erworben, cx2 ist es nicht mehr )
        // position pos2 wurde schon geloescht.
        // erase the old positions, make cx valid or NULL
#ifdef DEBUG_CALCULATION_Q
```

```
        cout << "SEQUENCE "  <<seqIndex+1
             << " movePos("<< pos1<< " <-> " << pos2 << ')'<<endl ;
        cout << "##Q before="<<S.quality()<<endl;
        cout << "AS("<<pos1<<") is "<< ( !isNull(S.asAt(pos1))?" not null":" NULL    " ) ;
        cout << " asQ="  << S.asQAt(pos1);
        cout << " lock=" << S.asLockAt(pos1).isAquired();
        cout << " linkQ-1=" <<((pos1-1>=0)?S.linkQAt(pos1-1):-1.0);
        cout << " linkQ="   <<((pos1<S.linkCount())?S.linkQAt(pos1):-1.0);
        cout << endl;
        cout << "AS("<<pos2<<") is "<< ( !isNull(S.asAt(pos2))?" not null":" NULL    " ) ;
        cout << " asQ="  << S.asQAt(pos2);
        cout << " lock=" << S.asLockAt(pos2).isAquired();

        cout << " linkQ-1=" <<((pos2-1>=0)?S.linkQAt(pos2-1):-1.0);
        cout << " linkQ="   <<((pos2<S.linkCount())?S.linkQAt(pos2):-1.0);
        cout << endl;
#endif
        ASSERT(pos1 != pos2);
        ASSERT(pos1 >= -1);
        ASSERT(pos1 < S.asCount());
        ASSERT(pos2 > -1);
        ASSERT(pos2 < S.asCount());
        ASSERT(!isNull(cx1));

        S.clearAS(pos1,true);
        if(!isNull(cx1)){
#ifdef DEBUG_CALCULATION_Q
            cout << "   .. put cx1 at " << pos2 << "   cx1 is "
                 <<(isNull(cx1)?" NULL":" not null")<< endl;
#endif
            ASSERT(S.asListAt(pos2)->contains(cx1));
            S.putASAt(pos2, cx1);        S.pushChange(pos2);
            evaluateLocalAt(seqIndex,pos2);
        } else
            ASSERT(isNull(S.asAt(pos2)));
        if(!isNull(cx2) ){
#ifdef DEBUG_CALCULATION_Q
            cout << "   .. put cx2 at " << pos1 << "   cx2 is "
                 <<(isNull(cx1)?" NULL":" not null")<< endl;
#endif
            ASSERT(S.asListAt(pos1)->contains(cx2));

            S.putASAt(pos1, cx2);        S.pushChange(pos1);
            evaluateLocalAt(seqIndex,pos1);
        } else
            // this can fail if pos1 == pos2
            ASSERT(isNull(S.asAt(pos1)));

//////////////
#ifdef DEBUG_CALCULATION_Q
        cout << "##Q after="<<S.quality()<<endl;
        cout << "AS("<<pos1<<") is "<< ( !isNull(S.asAt(pos1))?" not null":" NULL    " ) ;
        cout << " asQ="  << S.asQAt(pos1);
        cout << " lock=" << S.asLockAt(pos1).isAquired();
        cout << " linkQ-1=" <<((pos1-1>=0)?S.linkQAt(pos1-1):-1.0);
        cout << " linkQ="   <<((pos1<S.linkCount())?S.linkQAt(pos1):-1.0);
        cout << endl;
        cout << "AS("<<pos2<<") is "<< ( !isNull(S.asAt(pos2))?" not null":" NULL    " ) ;
        cout << " asQ="  << S.asQAt(pos2);
        cout << " lock=" << S.asLockAt(pos2).isAquired();

        cout << " linkQ-1=" <<((pos2-1>=0)?S.linkQAt(pos2-1):-1.0);
        cout << " linkQ="   <<((pos2<S.linkCount())?S.linkQAt(pos2):-1.0);
        cout << endl;
#endif
    }
    incStatCounter(movePos,statSuccess);
}

////////////////////////////////////////////////////////////////////////////
// ersetze eine belegte Position durch einen CX mit dem gleichen Basisknoten
void     RandomOptimizer::singleUpgradeAt(int seqIndex)
{
    DEBUGTracedFrame;
    incStatCounter(singleUpgrade,statEnter);
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    // fuer N Zufallsereignisse {
    //   suche eine position aus (RANDOM)
    //      entsperre den CX der an dieser Pos vorhandenen ist. ==> prevCX
    //   suche einen CX aus dem Pool fuer diese Position aus (RANDOM)
    //       versuche N mal (=3) einen erwerbbaren CX zu finden ( != prevCx )
    //       danach akzeptiere irgendeinen.
    //       setze prevCX ein wenn kein anderer moeglich ist.
    //   wenn weiterreichen,  suche eine neue position aus (RANDOM)
    //   sperren den State fuer weitere Bearbeitung
    //}
    ASSERT(itsMaxSingleUpgradeRandomPos<S.asCount());
#ifdef singleUpgrade_DEBUG_CALCULATION_Q
    cout << "SEQUENCE["<< seqIndex+1 << "] singleUpgrade" << endl;
    double Q = S.quality();
#endif
    CHECK_LOCK;

    for(int n=0; n<itsMaxSingleUpgradeRandomPos; n++){
        // suche eine nicht veraenderte Position, fuer die es CX gibt.
        //: index einer AS [1 .. as count-1]
        int pos = chooseAS(S);

#ifdef  SINGLE_UPGRADE_FILL_EMPTY_POS
        // if no pos found and at least one empty position exists.
        if(pos == -1){
            singleExchangeAt(index);
            incStatCounter(singleUpgrade,statFailed1);
            if(SHOW_Q_TRACE) cout <<"END---"<<__FUNCTION__ << endl;
            return ;
        }
#endif  // SINGLE_UPGRADE_FILL_EMPTY_POS
        if(pos == -1){
            incStatCounter(singleUpgrade,statFailed);
            if(SHOW_Q_TRACE) cout <<"END---"<<__FUNCTION__ << endl;
            return;
        }
        Context::Handle newH = singleUpgradeAtPos(seqIndex, pos);

        if(!isNull(newH)){
            // fuege newH an position pos ein, loesche die berechneten Daten

            S.putASAt(pos,newH);S.pushChange(pos);
            evaluateLocalAt(seqIndex,pos);
#ifdef singleUpgrade_DEBUG_CALCULATION_Q
            cout << "SINGLE_UPGRADE  PUT AT "<<pos << " =" <<  (Q - S.quality()) << endl;
            CHECK_QUALITY_STABLE(seqIndex)
#endif

                incStatCounter(singleUpgrade,statSuccess);
            return ;
        } else {
            incStatCounter(singleUpgrade,statFailed2);
        }

    }
    incStatCounter(singleUpgrade,statSuccess);
    if(SHOW_Q_TRACE) cout <<"END---"<<__FUNCTION__ << endl;
}
////////////////////////////////////////////////////////////////////////////
// finde einen CX, der an Stelle des Cx in AS[pos] eingesetzt werden kann
// und der den gleichen Basisknoten verwendet.
Context::Handle   RandomOptimizer::singleUpgradeAtPos(int seqIndex,int pos)
{
    DEBUGTracedFrame;
    incStatCounter(singleUpgrade,statEnter);
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());

    if(pos == -1){
        return Context::Handle();
    }

    Context::Handle prevCX = S.asAt(pos);
    const Resource* prevR = prevCX->Lookup(itsRootName);
    if(prevR==NULL){
        ErrMessage err;
```

```
        err << fatal << "context # without root resource in as #"
            <<pos<<eom;
    }
    S.clearAS(pos,true); // S.pushChange(pos);
#ifdef singleUpgrade_DEBUG_CALCULATION_Q
    cout << "SINGLE_UPGRADE  CLEAR AS("<<pos<<")=" << (Q - S.quality()) << endl;
#endif
    ContextLock     currASLock;
    // suche einen **erwerbbaren** CX aus dem Pool fuer diese Position aus (RANDOM)
    // der dem gleichen Basisknoten angehoert wie prevCX
    const ContextList &currList = *S.asListAt(pos);
    int count = min(itsMaxSeekSameNode,currList.length());
    ContextList::Iterator iter = currList.last();
    ASSERT(iter);
    // die Liste sollte mindestens eine Moeglichkeit enthalten.
    // versuche N mal (~10) einen neuen erwerbbaren CX zu finden ( cx != prevCx )
    // danach akzeptiere irgendeinen.
    const Resource *res = NULL;
    do {
        ContextList::Iterator startIter = iter;
        int r = longInRange0((currList.length()-1)/10)+1; // range [1 .. arg]
        // jump to the  r-th  iterator with same base node or wrap around
        while(r>0){
            --iter; if(!iter){ iter = currList.last(); }
            if(startIter==iter){
                count=0; // wrap around and nothing found.
                break;   // while (r>0)
            }
            res = (*iter)->Lookup(itsRootName);
            if(res==NULL){
                ErrMessage err;
                err << fatal
                    << "context without root resource in as #"<<pos<<eom;
            }
            if (prevR->compWith(*res)==0){
                r--;
            }
        }
    } while( --count >= 0 &&
             ! (prevR->compWith(*res)==0) &&
             ! currASLock.aquire(const_cast<Context::Handle&>(*iter)) );

    if(currASLock.isAquired()){
        currASLock.release(true);
        return Context::Handle();
    }

    // restore the old value
    S.putASAt(pos,prevCX);S.pushChange(pos);
    evaluateLocalAt(seqIndex,pos);

    return *iter;
}
//////////////////////////////////////////////////////////////////////
RandomOptimizer::sequenceChangeActions RandomOptimizer::chooseAction(void)
{
    double r = getRandomDouble()*100.0; // non const operation!!

    if(r < (itsCurrProfile)[swapItems])      return swapItems;
    if(r < (itsCurrProfile)[singleExchange]) return singleExchange;
    if(r < (itsCurrProfile)[crossOver])      return crossOver;
    if(r < (itsCurrProfile)[randomGenerate]) return randomGenerate;
    if(r < (itsCurrProfile)[mergeFromBest])  return mergeFromBest;
    if(r < (itsCurrProfile)[moveLinkedRange])return moveLinkedRange;
    if(r < (itsCurrProfile)[removeRange])    return removeRange;
    if(r < (itsCurrProfile)[singleUpgrade])  return singleUpgrade;
    if(r < (itsCurrProfile)[movePos])        return movePos;
    return doNothing;
}
//////////////////////////////////////////////////////////////////////
const char* RandomOptimizer::action2Name(RandomOptimizer::sequenceChangeActions  a)
{
    return theActionName[a];
}
//////////////////////////////////////////////////////////////////
//
static  const char* eventNames[] = {
    "enter",     "success",    "fail",    "fail2",    "better"
};
void RandomOptimizer::dumpStatTable(ostream& os)
{
    const int nameSize  = 20;
    const int fieldSize =  8;
    os << setw(nameSize) << "EVENT "<<" :";
    for(int n= 0; n<RandomOptimizer::theGroupCount; n++){
        os << setw(fieldSize)<< eventNames[n];
        if(n+1<RandomOptimizer::theGroupCount)
            os << ',';
    }
    os << endl;
    for(int i=0; i<countChangeActions; i++){
        os << setw(nameSize)<< action2Name(sequenceChangeActions(i)) <<" [";
        for(int n= 0; n<RandomOptimizer::theGroupCount; n++){
            os << setw(fieldSize)
               << RandomOptimizer::statCounter(sequenceChangeActions(i),statGroups(n));
            if(n+1<RandomOptimizer::theGroupCount)
                os << ',';
        }
        os << "]\n";
    }
}

//////////////////////////////////////////////////////////////////////
//
bool RandomOptimizer::loadProfile(
        const MyString& name)
{
    enum{ ok = 0 , fail = 1 };
    fstream is(name, ios::in|ios::nocreate);
    if(!is){
        ErrMessage err;
   err << error << "loadState:"<< "can't open profile file \"" <<name<<"\"";
        return fail;
    }

    return loadProfile(is);
}

//////////////////////////////////////////////////////////////////////
//
bool RandomOptimizer::loadProfile(
        istream& is )
{
    enum{ ok = 0 , fail = 1 };
    int errPos=0;
    AssertHook noStackDump(NULL);

    MyString version;
    streampos fPos;

    double summe;
    MyString temp;
    int realProfileCount = countChangeActions;

    // reset
    itsBestASCount   = 0;
    itsBestLinkCount = 0;

    ReadBlock  block1(is,"PROFILES");

    if(!block1.open())  ERROR_RECOVER;
    int profileCount;
    fPos = is.tellg();

    // lookahead for VERSION field
    if(!( is >> ws && getString(is,version) )){
       ErrMessage err;
       err << error  <<"missing field name expected VERSION or LENGTH, found "
            << version << endl;
       return fail;
    }
    is.seekg(fPos);
    if(UPCASE(version)=="VERSION"){
        if( parseDQField(is, "VERSION",version) ) ERROR_RECOVER;
```

```
    } else {
        version = "";
    }
    ////////////////////////
    // changes of versions
    // versions before v 1.01;  swapItems .. moveLinkedRange ( 7 )
    // versions  v 1.01;        added removeRange   ( 8 )
    // versions  v 1.02;        added singleUpgrade ( 9 )
    // versions  v 1.03;        added movePos       ( 10 )

    // nr of actions per line in stream
    if(  version >= "1.03")       realProfileCount = 10;
    else if(  version == "1.02")  realProfileCount = 9;
    else if(  version == "1.01")  realProfileCount = 8;
    else if(  version == "")      realProfileCount = 7;

    if(parseField(is, "LENGTH", profileCount) ) ERROR_RECOVER;

    if(parseField(is, "NAMES", temp) ) ERROR_RECOVER; // dummy read
    if(itsProfileChange == itsProfileChangeDefault){
        itsProfileChange = new double [ profileCount * countChangeActions ];
        itsPctProfiles   = new double [ profileCount * countChangeActions ];
        itsMaxProfile    = new double [ profileCount * countChangeActions ];
    } else {
        delete [] itsProfileChange;
        delete [] itsPctProfiles;
        delete [] itsMaxProfile;
        itsProfileChange = new double [ profileCount * countChangeActions ];
        itsPctProfiles   = new double [ profileCount * countChangeActions ];
        itsMaxProfile    = new double [ profileCount * countChangeActions ];
    }
    for(int i=0; i<profileCount; i++){
        //format:  PROFILE  upperThreshold  profilePct ...
        if(parseField(is, "PROFILE", itsProfileChange[i] , false) ) ERROR_RECOVER;
        for(int n=0; n<realProfileCount; n++){
            double a;
            if(!(is && is >> ws >> a )){
                ErrMessage err;
                err << warn
                    << "value="<<a<<"  current profile nr n="<<n
                    <<"  current action index i="<<i
                    << "  countChangeActions="<<countChangeActions
                    << "  realProfileCount="  <<realProfileCount<< eom;
                ERROR_RECOVER;
            }
            if(n<countChangeActions)
                const_cast<double&>(itsPctProfiles[n+ i*countChangeActions]) = a;
        }
        is >> skipline;
        if(realProfileCount<countChangeActions)
            itsPctProfiles[(countChangeActions-1)+ i*countChangeActions] = 0.05;

        // test  summe (itsProfiles[i]) == 100 ( oder normieren )
        summe = 0.0;
        for(int n=0;n<countChangeActions; n++){
                summe += itsPctProfiles[n + i*countChangeActions];
        }
        for(int n=0;n<realProfileCount; n++){
            itsPctProfiles[n + i*countChangeActions] /= summe;
            itsPctProfiles[n + i*countChangeActions] *= 100.0; // range 0 .. 100
        }
    }
    if(!block1.close()) ERROR_RECOVER;

    itsProfileCount = profileCount;
    itsProfileChange[profileCount-1] = 1.01; // terminate profile change
    itsCurrProfile  = itsMaxProfile;
    itsProfileIndex = 0;

    // generate threshold values.
    RandomOptimizer::calcProfiles();
    return ok;
error_recover:
    ErrMessage err;
    err << error
        << " parsing profile failed. (code =  " << errPos
        << " at line " << streampos2lineNr(is,is.tellg())
        << ")"  << eom;
    return fail;
}
////////////////////////////////////////////////////////////////////
//
void RandomOptimizer::dumpConfig(ostream& os)
{
    enum{ ok = 0 , fail = 1 };
    {WriteBlock  block2(os,"CONFIG");
    if(block2){
        os << indent << "VERSION           2" <<endl;
        os << indent << "DumpTimeInterval  " << itsDumpTimeInterval << endl;
        os << indent << "LoopCount         " << itsLoopCount << endl;
        os << indent << "MaxLoopCount      " << itsMaxLoopCount << endl;
        os << indent << "LastQ             " << itsLastQ << endl;
        os << indent << "LastQChanged      " << itsLastQChanged  << endl;
        os << indent << "QTolerance        " << itsQThreshold  << endl;
        os << indent << "toleranceDecayRate  " << itsThresholdDecayRate << endl;
        os << indent << "maxIdleLoops      "     << itsMaxIdleLoops << endl;
        }
    }

    {WriteBlock  block3(os,"ACTIONCONFIG");
    if(block3){
        {WriteBlock  block6(os,"PROFILES");
        if(block6){
            os << indent << "VERSION " << "\"1.02\"" << endl;
            os << indent << "LENGTH " << itsProfileCount << endl;
            os << indent << "NAMES  " << setw(10) << "upperLimit";
            for(int i= 0; i<countChangeActions; i++){
                os  << " " << action2Name(sequenceChangeActions(i));
            }
            os << endl;
            for(int i= 0; i<itsProfileCount; i++){
                os << indent << "PROFILE "<< setw(6) << itsProfileChange[i];
                // print thresholds as percents
                os <<setw(10)<<"  "<<itsPctProfiles[i*countChangeActions];
                for(int n=1; n<countChangeActions; n++)
                        os <<setw(10)<<"  "<<itsPctProfiles[i*countChangeActions+n] ;
                os << endl;
            }
        }
        }
    }
    }
}

////////////////////////////////////////////////////////////////////

bool RandomOptimizer::loadConfig(istream & is)
{
    enum{ ok = 0 , fail = 1 };
    int errPos=0;
    AssertHook noStackDump(NULL);

    MyString s;
    MyString version = "1"; // Config Version

    ReadBlock  block2(is,"CONFIG");
    ReadBlock  block3(is,"ACTIONCONFIG");
    if(!block2.open())  ERROR_RECOVER;
    if(!block2.skipToEnd())  ERROR_RECOVER;
    if(!block3.open())  ERROR_RECOVER;
    if(!block3.skipToEnd())  ERROR_RECOVER;

    return ok;
  error_recover:
    ErrMessage err;
    err << error
        << " parsing optimizer seed failed. (code =  " << errPos
        << " at line " << streampos2lineNr(is,is.tellg())
        << ")"  << eom;
    return fail;
}
////////////////////////////////////////////////////////////////////
//
```

```
RandomOptimizer& RandomOptimizer::debugTraceQ(void){
    if((itsIsImproved
        || (itsMaxTraceInterval > 0
            && itsMaxTraceInterval < (itsLoopCount - itsLastTraceCount))))
    {
        itsLastTraceCount = itsLoopCount;
        if(itsDebugFlags[traceQ]){
            double best=0.0, worst=itsArena[0]->quality(), mid=0.0;
            for(int i=0; i<sequenceCount(); i++){
                double currQ =itsArena[i]->quality();
                if(best<currQ) best  = currQ;
                if(worst>currQ)worst = currQ;
                mid += currQ;
            }
            mid /=sequenceCount();
            const char sep = ';';
            (debugStream()) << "TRACE_Q"<<sep<< itsLoopCount << sep
                            << itsLastQ << sep
                            << qThreshold() << sep
                            << thresholdDecayRate() << sep
                            << max(itsLastQ ,mid) * qThreshold() << sep
                            << itsBest->quality() << sep
                            << itsBestASCount << sep
                            << itsBestLinkCount << sep;

            (debugStream()) << best << sep << (mid) << sep << worst << sep;
            (debugStream()) << endl;
        } else {
            debugTraceEachQ();
        }
    }
    return *this;
};
//////////////////////////////////////////////////////////////////////
//
RandomOptimizer& RandomOptimizer::debugTraceEachQ(void){
    if(itsDebugFlags[traceEachQ]){
        const char sep = ';';
        (debugStream()) << "TRACE_EACHQ"<<sep<< itsLoopCount << sep;
        double bestQ=0.0, worstQ=itsArena[0]->quality(), midQ=0.0;
        for(int i=0; i<sequenceCount(); i++){
            double currQ =itsArena[i]->quality();
            if(bestQ<currQ) bestQ  = currQ;
            if(worstQ>currQ)worstQ = currQ;
            midQ += currQ;
        }
        midQ /=sequenceCount();
#ifdef  FOLLOW_MID_Q
        double  Q = midQ;
#elif defined(FOLLOW_BEST_Q)
        double  Q = itsBest->quality();
#endif   // FOLLOW_XXX

        double threshold = max(itsLastQ ,Q) * qThreshold();
        (debugStream())
            << itsLastQ << sep
            << qThreshold() << sep
            << thresholdDecayRate() << sep
            << threshold << sep
            << itsBest->quality() << sep
            << itsBestASCount << sep
            << itsBestLinkCount << sep;
        (debugStream())
            << bestQ << sep
            << midQ << sep
            << worstQ << sep
            << '*' << sep;
        for(int i=0; i<sequenceCount(); i++){
            (debugStream())  << itsArena[i]->quality()<< sep;
        }
        (debugStream()) << "\n";
    }
    return *this;
};

//////////////////////////////////////////////////////////////////////
//
```



```
RandomOptimizer& RandomOptimizer::debugTraceActions(void){
    if(itsDebugFlags[traceActions]){
        const char sep = ';';
        (debugStream())
            << "TRACE_ACTIONS";
        for(int i=0; i< sequenceCount(); i++){
            (debugStream()) << sep << action2Name(itsLastAction[i]);
        }
        (debugStream())
            << endl;
    }
    return *this;
};
//////////////////////////////////////////////////////////////////////
//
RandomOptimizer& RandomOptimizer::debugTraceActionTableChange(void){
    if(itsDebugFlags[traceActionTableChange]){
        (debugStream())
            << "TRACE_ACTIONTABLECHANGE";
        for(int i=0; i<countChangeActions; i++){
            (debugStream())
                << ';'<<  RandomOptimizer::itsCurrProfile[i];
        }
        (debugStream())
            << endl;
    }
    return *this;
};
//////////////////////////////////////////////////////////////////////
//
RandomOptimizer& RandomOptimizer::debugParams(void){
    if(itsDebugFlags[traceParams]){
        (debugStream())
            << "OPTI_PARAMS;"
            << "  dumpfile="<< itsDumpFileName
            << "  nextDumpTime="<<itsNextDumpTime
            << "  dumpTimeInterval="<<itsDumpTimeInterval
            << "  endTime="<<itsEndTime
            << "  maxLoopCount="<<itsMaxLoopCount
#ifdef WITH_VETOPERIOD
            << "  vetoPeriod="<<State::vetoPeriod()
#else
            << "  vetoPeriod=(disabled)"
#endif
            << "  initialQThreshold="<<initialQThreshold()
            << "  qThreshold="<<itsQThreshold
            << "  thresholdDecayRate="<<itsThresholdDecayRate
            << "  maxIdleLoops="<<itsMaxIdleLoops
            << "  arenaSize="<<sequenceCount()
            << "  minTTL="<<minTTL()
            << "  maxTTL="<<maxTTL()
            << "  randomSeed="<<itsRndSeed
            << "  debugFlags=\""<<itsDebugOptions<<"\""
            << "  maxTraceInterval="<< itsMaxTraceInterval
            << endl;
        (debugStream())
            << "OPTI_DEFINES; "
            << "WITH_VETOPERIOD="
#ifdef     WITH_VETOPERIOD
            << "(defined="<<#WITH_VETOPERIOD<<")"
#else  //  WITH_VETOPERIOD
            << "(undef)"
#endif //  WITH_VETOPERIOD
            << " removeNotLocked="<<(itsCanRemoveNotLocked?"true":"false")
            << " removeNotLinked="<<(itsCanRemoveNotLinked?"true":"false")
            << " canFillEmpty="<<(itsCanFillEmpty?"true":"false")
            << " initialMinAssignedCxCountPct="<<initialMinAssignedCxCountPct
            << endl;
    }
    return *this;
}

//////////////////////////////////////////////////////////////////////
#include "Array.cc"
template class Array<RandomOptimizer::sequenceChangeActions>;
```

N.5 Quelltext: RandomOptimizerBASE.h

```
#ifndef __RandomOptimizerBASE_H__
#define __RandomOptimizerBASE_H__  1
//
//
// $Id: RandomOptimizerBASE.h,v 1.10 2000/05/29 16:21:49 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "MyACG.h"

#include "Cache.h"
#include "Protein.h"
#include "PeakPool.h"
#include "PeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"

#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"

#include "RandomOpti_StateIndependent.h"
#include "RandomOpti_Calc.h"

//////////////////////////////////////////////////////////////
//: build subset of two intVecs,
//////
//  return: true       res and other   have common members
//          false      no overlapp,  res and resLen are unchanged
//

bool  subset(
        intVec& res,          // in|out: result vector
        int & resLen,         // in|out: count of used elements
        const intVec& other,  // in:     other set
        int offSet=0);        // in:     shift for elements

//: remove the element key from res
//  move all elements behind key one position forward.
//
void removeElement(intVec& res, int& resLen, int key);

//////////////////////////////////////////////////////////////////

class StateIndependent;
class State;
class LinkBuilder;
class StateDiff;

//////////////////////////////////////////////////////////////////
class  RandomOptimizerBASE;
```

```
class  RandomOptimizerBASE  : public RCObject
{
public:
    typedef IMngObject< RandomOptimizerBASE > Handle;
    typedef RandomOptimizerBASE Self;

    virtual ~RandomOptimizerBASE(void);
    RandomOptimizerBASE(void);

    ////////////////////////// ACTION Helper
    //:  erzeuge eine Zufallsbelegung
    void randomInitAt(int seqIndex);
    //: suche AS die nicht erworben sind und
    //  ersetze sie durch eine leere Stelle
    void removeNotLockedAt(int seqIndex);
    //: suche AS die nicht auf beiden Seiten verknuepft sind
    //  und ersetze sie durch eine leere Stelle
    void removeNotLinkedAt(int seqIndex);
    //: loesche den CX an position pos == index der as
    void removeRangeAt(int seqIndex);

    //: bewerte eine einzelne AS
    void evaluateLocalAt(int seqIndex, int asPos);
    //: schließe die Bewertung eines States ab.
    void evaluateAt(int seqIndex);
    //: calculate AS local Quality for range [firstPos .. len -1]
    void localEvaluateInRange(int seqIndex,int firstPos, int len);
    //: berechne Q vollstaendig neu.
    void rebuildQAt(int seqIndex);
    //: berechne Q vollstaendig neu.
    void rebuildBestQ(void);

    //: bewerte die Arena
    virtual void evaluateArena(void);

    //: generates two new States by merging two other States
    virtual void     crossOverAt(
            int destIndex1,int destIndex2, // dest
            int srcIndex1 ,int srcIndex2   // src
            );
    //:  generates *one* of two new States by merging two other States
    //   kopiere Stuecke aus srcS1 und srcS2 nach destS1
    //   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
    virtual void     singleCrossOverAt(
            int destIndex1,
            int srcIndex1 ,int srcIndex2,
            bool generateFirst   // true generate dest1 , false generates dest2
            );

    //:
    virtual void optimize(void);
    //:
    virtual bool abbruchbedingung(void);

    ////////////////////// HELPER Functions
    //: search for matching dest position for AS's starting at position firstPos
    int  findFirstMatch(
            int seqIndex,
            int firstPos,   //  anfang des bereichs
            int rangeLen,   //  laenge des bereichs (== linkLen+1)
            const intVec& destPos, // alternative positionen
            int posLen      // genutzte laenge in destPos
            );

    //: copy one range of AS from one State to another.
    void   crossCopyRange(
            int destIndex, // dest
            int srcIndex , // src
            int firstAS, int len
            );

    //: wendet einen Auschnitt eines StateDiffs auf einen State an.
    //  dannach werden die neuen CX erworben und dann all alten Positionen,
    //  die im Konflikt mit den neuen stehen, verworfen.
    //  Qualität und Locks im Endergebnis sind gültig.
    //  (im Gegensatz zum applyDiff)
    void     applyDiffRangeAt(
```

```
        int destIndex1,

        const StateDiff & diff,
        // erste Aktion im Diff,
        int start = 0,
        // Anzahl der Folgeaktionen. ( <0 : len of diff )
        int len   = -1 ,
        // SWAP's are different from ADD and SUB operations,
        // not all cx combinations are swappable.
        // So, SWAP's are not always applicable.
        // iff they are applied as two sub and two add CX Operations!
        //    NOT as an exchange CX !
        bool apply_swaps = true
        );

//: kopiere den Bereich [srcPos .. srcPos+len-1]
//  nach [destPos .. destPos+len-1]
//  innerhalb einer Sequenz, achtet auf ueberlappte Bereiche
void copyRange(int seqIndex, int destPos, int srcPos, int len);

////////////////////// Data Access Funtions
RandomOptimizerBASE&     useIndependent(StateIndependent::Handle p);
StateIndependent::Handle usedIndependent(void)const;

//: get/copy State Objects
RandomOptimizerBASE& stateAt(int i, State::Handle& st);
State::Handle&       stateAt(int seqIndex);
//: Ausgabe der State Objekte auf dem Dumpfile.
virtual void dumpState(void);
//: Lesen der State Objekte aus dem File "name".
virtual bool loadState(
        const MyString& name, PeakPool::Handle pool, PeakNet::Handle net);

//////////////////////
//: laed/speichert den config teil des Dumps
virtual bool loadConfig(istream& is);
virtual void dumpConfig(ostream& os);

//: loesche den Zustand aller State-Objekte.
void initState(void);
//: setze alle Optimizer spezifischen Variable (loop, time, ..)
//  auf den Startzustand zurueck.  Die STATE bleiben unveraendert.
virtual void restartRun(void);

//: the peak net
RandomOptimizerBASE&     peakNet( const PeakNet::Handle& p);
const PeakPool::Handle&  peakPool( void)const;

//: the peak pool
RandomOptimizerBASE&     peakPool( const PeakPool::Handle& p);
const PeakNet::Handle&   peakNet( void)const;

//: the protein
RandomOptimizerBASE& protein( Protein::Handle& p);
Protein::Handle      protein( void)const;

//:  time interval from start to end
RandomOptimizerBASE& runTime( time_t p);
time_t               runTime(void)const;
//: end time for optimize in abbruch ( == starttime + runTime() )
time_t               endTime(void)const;

//: time interval for dumps
RandomOptimizerBASE& dumpTimeInterval(time_t p);
time_t               dumpTimeInterval(void)const;

//: end of optimize in abbruch
RandomOptimizerBASE& maxLoop( long p);
long                 maxLoop(void)const;

//: current loop count (increasing in optimize)
RandomOptimizerBASE&  loopCount(long l);
long   loopCount(void)const;

//: loop count of last q change
long   lastQChanged(void)const;

//: resize Arena and initialize form best value
    RandomOptimizerBASE& sequenceCount(int i);
    int sequenceCount(void)const;

    //: quality at which optimize will stop in abbruch
    RandomOptimizerBASE& endQ(double d);
    double endQ(void)const;

    //: no of loops without an quality improvement after
    //  which optimize will stop in abbruch
    RandomOptimizerBASE& maxIdleLoops(long l);
    long maxIdleLoops(void)const;

    //: no of loops between dumps ( UNUSED ??? )
    RandomOptimizerBASE& dumpLoopInterval(long l);
    long dumpLoopInterval(void)const;

    //: no of loops between trace messages if no quality improvement( UNUSED ??? )
    RandomOptimizerBASE& maxTraceInterval(long l);
    long maxTraceInterval(void)const;

    //: minimum no of loops between trace messages even if quality was improved
    RandomOptimizerBASE& populationTraceInterval(long l);
    long populationTraceInterval(void)const;

    //: amount of applied changes  in guidedCrossOverAt
    RandomOptimizerBASE& guidedCrossOverSelectionPct(double & d);
    double guidedCrossOverSelectionPct(void)const;


    //: name of root resource for patterns
    RandomOptimizerBASE& rootName(const ResourceName& name);
    const ResourceName& rootName(void)const;

    //: init seed for random number source
    RandomOptimizerBASE& randomSeed(const MyString& randomText);

    RandomOptimizerBASE& exitReason(const MyString& r);
    const MyString&   exitReason(void)const;

    //:
    void dumpFileName(const MyString& name);
    const MyString& dumpFileName(void)const;

    ////////////////////// TRACE Functions
    RandomOptimizerBASE& debug(const MyString& options);
    const MyString&  debug(void)const;
    RandomOptimizerBASE& debugStream(ostream& os);
    ostream&         debugStream(void)const ;

    //
    static void registerOptions(CmdLine& cmd);
    virtual bool configure(CmdLineArgIter& cmdIter);

protected:
    //: anzahl der versuche einen noch nicht erworbenen CX zu finden
    static const int itsMaxSeekAquirablePos        =  4;
    //:anzahl der versuche einen CX mit dem gleichen Node zu finden(inner loop)
    static const int itsMaxSeekSameNode            = 20;
    //:anzahl der runden (replace) (outer loop)
    static const int itsMaxSingleUpgradeRandomPos  = 3;
    //:anzahl der  runden (swap)
    static const int itsMaxSwapItemRounds          = 1;
    //:anzahl der  runden (movePos)
    static const int itsMaxMoveItemRounds          = 2;
    //:anzahl der runden (replace)
    static const int itsMaxSingleExchangeRandomPos = 3;
    //:anzahl der runden (moveRange)
    static const int theMoveLinkedRangeMaxRounds   = 4;

protected:
    virtual bool abbruchbedingung_qualityChange(void);

    bool      itsIsDebugStreamOwner;
    ostream * itsDebugStream;
    MyString  itsDebugOptions;
    enum debugOptions_e {
        traceQ,traceEachQ,traceActions,traceParams,traceActionTableChange,
        traceDumps,tracePopulationStat,tracePopulationDiff,
```

```
        countDebugOptions
    };
    bool itsDebugFlags[countDebugOptions];
    //: drop a trace of  [action]
    virtual RandomOptimizerBASE& debugParams(void);
    virtual RandomOptimizerBASE& debugTraceQ(void);
    virtual RandomOptimizerBASE& debugTraceEachQ(void);
    virtual RandomOptimizerBASE& debugTracePopulationUsedCX(void);
    virtual RandomOptimizerBASE& debugTracePopulationDiff(void);
    virtual RandomOptimizerBASE& debugTracePopulationStat(void);
    virtual RandomOptimizerBASE& debugTraceFinalPopulationStat(void);
#define WITH_TIME_STAMPS    1
#undef  WITH_TIME_STAMPS
#ifdef    WITH_TIME_STAMPS
    RandomOptimizerBASE& debugTraceLoop(void);
#endif // WITH_TIME_STAMPS
    RandomOptimizerBASE& debugTraceDumps(void);

    //: helper functions
    int                   chooseAS(const State &S,      int  maxTry=5);
    int                   chooseEmptyAS(const State &S, int  maxTry=5);
    ContextList::Iterator chooseIterAt(const State &S, int pos, int  maxTry=5);

    //: returns a random float number in range 0<=x<1
    double getRandomDouble(void);
    //: returns a random integer number in range MIN_LONG<=x<=MAX_LONG
    long getRandomLong(void);

    //: returns a random integer number in range low<=x<=high
    long longInRange(long low, long high);
    //: returns a random integer number in range 0<=x<=high
    long longInRange0(long high);

    void random_shuffle(intVec &x);
    void random_shuffle(intVec &x, int len);

    ////////////////////////////////////////////////////////
    static const bool    itsCanRemoveNotLocked;
    static const bool    itsCanRemoveNotLinked;
    static const bool    itsCanFillEmpty;
    static const double  initialMinAssignedCxCountPct;

    ////////////////////////////////////////
    virtual void invariante_checkAllCacheMembers(
            int seqIndex,
            bool checkUnlockedToo = true)const;

    ////////////////////////////////////////
    //: start value for no of loops between time checks
    RandomOptimizerBASE& initialTC(long l);
    long initialTC(void)const;


    ////////////////////////////////////////
    //: name of the dumpfile, defaults to ".dumpfile"
    MyString                          itsDumpFileName;

    //: why was this optimizer dumped
    MyString                          itsDumpReason;

    //: Startwert des randomnumber-generators
    MyACG::seed_t                     itsRndSeed;

    //: the random number generator
    MyACG                             itsGen;

    //: peakpool
    PeakPool::Handle                  itsPool;

    //: peaknet
    PeakNet::Handle                   itsNet;

    //: independent data
    StateIndependent::Handle          itsIndependent;

    //: calculation & caching
    MyString                           itsCalculatorName;
    RandomOpti_Calc_noCache::Handle      itsCalc;

    //: the working set of Assignments.
    Array< State::Handle > itsArena;
    //: set of best sequences (not shared with itsArena!)
    State::Handle  itsBest;
    int itsBestASCount;
    int itsBestLinkCount;

    //: how log should it run [sec] (time interval)
    time_t itsRunTime;
    //: when should it end [sec]
    time_t itsEndTime; // = time_at_start_of_optimize + itsRunTime
    //: time of last dump
    time_t itsNextDumpTime;
    //: maximum time interval of dump [sec]
    time_t itsDumpTimeInterval;
    //: interval for dumps (UNUSED !!)
    long itsDumpLoopInterval;
    //: start value for itsTC, default 60, 0 if one cycle is slow (geneticOpti)
    //  don't make it larger than itsMaxLoopCount !
    int itsInitialTC;
    //: counter, check sytem time, whenever itsTC is counted down to 0 from 60
    int itsTC;

    //: current loop count
    long itsLoopCount;
    //: maximum loop count, stop if reached
    long itsMaxLoopCount;
    //: interval for traces, ( if not improved ) ignored i == 0
    long itsMaxTraceInterval;
    //: loop count of last trace message
    long itsLastTraceCount;
    //: did this loop improve itsBest? (valid 'inside' ::optimize)
    bool itsIsImproved;

    //: current quality
    double itsLastQ;
    //: loop count of last quality improvement
    long   itsLastQChanged;

    //: maximum quality, optimize stops if bestQ >= itsEndQ
    //  ignored if endQ < 0
    double itsEndQ;

    //: amount of applied changes  in guidedCrossOverAt
    double  itsGuidedCrossOverSelectionPct;

    //: after how many loops without quality improvement
    //  should we decrease the quality threshold
    long   itsMaxIdleLoops;
    ////////////////////////////////
    //: loop count beim letzten STAT / DIFF Dump.
    long                          itsPopulationLoop;
    long                          itsPopulationDumpInterval;

    //: Abstand zum Besten in ADD/SUB Bewegungen.
    Matrix<long>                   itsAtomDist;
    //:  Qualitätsabstand zum Besten.
    Matrix<double>                 itsQDist;

    //:anzahl der veränderungen je Aminosäureposition
    longVec                       itsPerAS;
    //: anzahl der veränderungen je Basispeak
    longVec                       itsPerBase;
    //: anzahl der veränderungen je CX
    longVec                       itsPerCX;

    ////////////////////////////////
    //: der root name der suchmuster
    ResourceName    itsRootName;

    //: default ascii context writer
    IMngPointer<ContextWriter>  itsDebugContextWriter;
};

//////////////////////////////////////////////////////////////////
//
```

```
inline ostream& RandomOptimizerBASE::debugStream(void)const{
    return *itsDebugStream;
};
////////////////////////////////////////////////////////////
//
/////////////////////////////////////////////////////
//: returns a random float number in range 0<=x<1
inline double RandomOptimizerBASE::getRandomDouble(void){
    return itsGen.asDouble();
};
/////////////////////////////////////////////////////
//: returns a random integer number in range MIN_LONG<=x<=MAX_LONG
inline long RandomOptimizerBASE::getRandomLong(void){
    return itsGen.asLong();
};
/////////////////////////////////////////////////////
//
inline long RandomOptimizerBASE::longInRange(long low, long high)
{
    return (itsGen.asLong() % (high-low+1)) + low;
}

inline long RandomOptimizerBASE::longInRange0(long high)
{
    return (itsGen.asLong() % (high+1));
}
////////////////////////////////////////////////////////////
//: assign an independent
//
inline StateIndependent::Handle RandomOptimizerBASE::usedIndependent( void)const{
    return itsIndependent;
}
/////////////////////////////////////////////////////
//
inline Protein::Handle RandomOptimizerBASE::protein(void)const{
    return itsIndependent->protein();
};
/////////////////////////////////////////////////////
//
inline long   RandomOptimizerBASE::loopCount(void)const
{
    return itsLoopCount;
}
inline RandomOptimizerBASE& RandomOptimizerBASE::loopCount(long l)
{
    itsLoopCount = l; return *this;
}
/////////////////////////////////////////////////////
//
inline long   RandomOptimizerBASE::lastQChanged(void)const
{
    return itsLastQChanged;
}
/////////////////////////////////////////
//: start value for no of loops between time checks
//  this is an IMPLEMENTATION method!
inline RandomOptimizerBASE&  RandomOptimizerBASE::initialTC(long l){
    itsInitialTC = l;
    return *this;
};
inline long  RandomOptimizerBASE::initialTC(void)const
{
    return itsInitialTC;
}
/////////////////////////////////////////////////////
//
inline RandomOptimizerBASE&  RandomOptimizerBASE::exitReason(const MyString& r){
    itsDumpReason = r;
    return *this;
};
inline const MyString&   RandomOptimizerBASE::exitReason(void)const{
    return itsDumpReason;
};
/////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::endQ(double d){
    itsEndQ     = d;
    return *this;
};
inline double RandomOptimizerBASE::endQ(void)const{
    return itsEndQ;
};

/////////////////////////////////////////////////////
//
inline long RandomOptimizerBASE::maxIdleLoops(void)const{
    return itsMaxIdleLoops;
};

inline RandomOptimizerBASE& RandomOptimizerBASE::maxIdleLoops(long i){
    itsMaxIdleLoops   = i;
    return *this;
};
/////////////////////////////////////////////////////
//
inline int RandomOptimizerBASE::sequenceCount(void)const{
    return itsArena.count();
};
/////////////////////////////////////////////////////
//
inline const MyString& RandomOptimizerBASE::dumpFileName(void)const
{
    return itsDumpFileName;
};
inline void RandomOptimizerBASE::dumpFileName(const MyString& name)
{
    itsDumpFileName = name;
};
////////////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::runTime( time_t p){
    itsRunTime=p; return *this;
};
inline time_t RandomOptimizerBASE::runTime(void)const{
    return itsRunTime;
};
////////////////////////////////////////////////////////////
//
inline time_t RandomOptimizerBASE::endTime(void)const{
    return itsEndTime;
};
////////////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::dumpTimeInterval(time_t p){
    itsDumpTimeInterval=p; return *this;
};
inline time_t RandomOptimizerBASE::dumpTimeInterval(void)const{
    return itsDumpTimeInterval;
};
////////////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::dumpLoopInterval(long p){
    itsDumpLoopInterval=p; return *this;
};
inline long RandomOptimizerBASE::dumpLoopInterval(void)const{
    return itsDumpLoopInterval;
};
////////////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::maxLoop( long p){
    itsMaxLoopCount=p; return *this;
};
inline long   RandomOptimizerBASE::maxLoop(void)const{
    return itsMaxLoopCount;
};
////////////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::maxTraceInterval( long p){
    itsMaxTraceInterval=p; return *this;
};
inline long   RandomOptimizerBASE::maxTraceInterval(void)const{
    return itsMaxTraceInterval;
};
////////////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::populationTraceInterval(
```

```
        long p)
{
    itsPopulationDumpInterval=p; return *this;
};
inline long   RandomOptimizerBASE::populationTraceInterval(void)const{
    return itsPopulationDumpInterval;
};
////////////////////////////////////////////////////
//: amount of applied changes  in guidedCrossOverAt
inline RandomOptimizerBASE& RandomOptimizerBASE::guidedCrossOverSelectionPct(
        double & d)
{
    itsGuidedCrossOverSelectionPct = d;
    return *this;
};
inline double RandomOptimizerBASE::guidedCrossOverSelectionPct(void)const
{
    return itsGuidedCrossOverSelectionPct;
};

////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::rootName(const ResourceName& root){
    itsRootName     = root;
    return *this;
};
inline const ResourceName& RandomOptimizerBASE::rootName(void)const{
    return  itsRootName;
};
////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::peakPool(const PeakPool::Handle& pool){
    itsPool     = pool;
    return *this;
};
inline const PeakPool::Handle& RandomOptimizerBASE::peakPool(void)const{
    return  itsPool;
};
////////////////////////////////////////////////////
//
inline RandomOptimizerBASE& RandomOptimizerBASE::peakNet(const PeakNet::Handle& net){
    itsNet      = net;
    return *this;
};
inline const PeakNet::Handle& RandomOptimizerBASE::peakNet(void)const{
    return  itsNet;
};

////////////////////////////////////////////////////////////
//
inline State::Handle& RandomOptimizerBASE::stateAt(int seqIndex){
    if(seqIndex == -1){
        return itsBest;
    } else {
        return itsArena[seqIndex];
    }
};
inline RandomOptimizerBASE& RandomOptimizerBASE::stateAt(
        int i, State::Handle& st)
{
    State &destS = *stateAt(i);
    ActiveAccount active(destS.account());
    destS.copyFrom(*st);
    return *this;
}
inline const MyString&  RandomOptimizerBASE::debug(void)const{
    return itsDebugOptions;
};


#endif // __RandomOptimizerBASE_H__
```

N.6 Quelltext: RandomOptimizerBASE.cc

```
//
//
// $Id: RandomOptimizerBASE.cc,v 1.20 2000/05/29 16:24:34 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include "NoDump.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "MyACG.h"
#include "Cache.h"
#include "Protein.h"
#include "PeakPool.h"
#include "PeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "AsciiContextReader.h"
#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizerBASE.h"

#include "StateMetrics.h"

//////////////////////////////////////////////////////////////
#include "RandomOptimizer.debug.h"
//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
//: build subset of two intVecs,
//////
//  return: true       res   and   have common members
//          false      no overlapp,  res and resLen are unchanged
//

bool  subset(intVec& res,          // in|out: result vector
             int & resLen,         // in|out: count of used elements
             const intVec& other,  // in:     other set
             int offSet=0);         // in:     shift for elements

//: remove the element key from res
//  move all elements behind key one position forward.
//
void removeElement(intVec& res, int& resLen, int key);

//////////////////////////////////////////////////////////////
//: build subset of two intVecs,
//////
//  return: true       res   and   have common members
//          false      no overlapp,  res and resLen are unchanged
//

bool  subset(intVec& res,          // in|out: result vector
             int & resLen,         // in|out: count of used elements
             const intVec& other,  // in:     other set
```

```
            int offSet)           // in:     shift for elements
{
    int top = 0;
    for(int n=0;n<resLen; n++){
        int k = 0;
        for(k=0;k<other.count() && res[n] != other[k]+offSet; k++);
        if(k<other.count()){
            res[top++] = other[k]+offSet;
        }
    }
    if(top == 0){
        return false;
    }
    resLen = top;
    return true;
}
//: remove the element key from res
//  move all elements behind key one position forward.
//
void removeElement(intVec& res, int& resLen, int key)
{
    int k = 0;
    for(k=0;k<resLen && res[k] != key; k++);
    if(k<resLen){
        for(;k+1<resLen; k++) res[k]=res[k+1];
        resLen--;
    }
}

////////////////////////////////////////////////////////////////
class StateIndependent;
class State;
class RandomOptimizerBASE;
class RandomOptimizer;
class GeneticRandomOptimizer;
class SubRandomOptimizer;

////////////////////////////////////////////////////////////////
const bool   RandomOptimizerBASE::itsCanRemoveNotLinked        = false;
const bool   RandomOptimizerBASE::itsCanRemoveNotLocked        = false;
const bool   RandomOptimizerBASE::itsCanFillEmpty              = true;
const double RandomOptimizerBASE::initialMinAssignedCxCountPct = 0.70L;

////////////////////////////////////////////////////////////////////////
//
static CmdArgStr       arg_dumpFileName('n',"dumpFileName","filename", "dump file name" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt       arg_maxTime('s',"maxTime","minutes", "maxTime [min]" , CmdArg::isOPTVALREQ);
static CmdArgInt       arg_dumpInterval('s',"dumpInterval","minutes", "dumpInterval [min]" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt       arg_maxTraceInterval('s',"traceInterval","loops", "traceInterval [loops]" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt       arg_maxLoops('s',"maxLoops","count", "maximum nr of loops" ,
   CmdArg::isOPTVALREQ);
static CmdArgInt       arg_maxIdleLoops('s',"maxIdleLoops","count", "maxIdleLoops" ,
   CmdArg::isOPTVALREQ);
static CmdArgFloat     arg_endQ('e',"endQ","endQ", "stop if that Quality is exceeded.(default: -1 ==
   ignored)" , CmdArg::isOPTVALREQ);
static CmdArgInt       arg_nrOfSequences('s',"nrOfSequences","count", "nrOfSequences (small positive
   integer)" , CmdArg::isOPTVALREQ);
static CmdArgStr       arg_random('r',"random","time|long", "config for randomize" ,
   CmdArg::isOPTVALREQ);

static CmdArgStr       arg_traceFileName('r',"traceFile","filename", "file for trace messages" ,
   CmdArg::isOPTVALREQ);
static CmdArgFloat     arg_crossOverPct('e',"crossOverPct","%float", "how much of the state
   difference is it applied? [0..1] (default = 0.1)" , CmdArg::isOPTVALREQ);


////////////////////////////////////////////
void RandomOptimizerBASE::registerOptions(CmdLine& cmd){
    static bool done = false;

    if(!done){
        done = true;
        cmd << arg_dumpFileName
            << arg_maxTime
            << arg_dumpInterval
            << arg_maxTraceInterval
            << arg_maxIdleLoops
            << arg_maxLoops
            << arg_endQ
            << arg_nrOfSequences
            << arg_random
            << arg_crossOverPct
            ;
    }
};


bool  RandomOptimizerBASE::configure(CmdLineArgIter& UNUSED(cmdIter)){
    long     count;
    double   d;
    MyString s;

    //
    // optiFileName : is part of randomOptimize !!
    //
    if(givenOption(arg_dumpFileName)){
        evalOption(s,"","",arg_dumpFileName);
        if(s != "")
            dumpFileName(s);
    }
    if(givenOption(arg_maxTime)){ // [ option min->intern sec ]
        evalOption(count,60,"",arg_maxTime);
        count *= 60;
        runTime( count );
    }
    if(givenOption(arg_dumpInterval)){ // [ option min->intern sec ]
        evalOption(count,10,"",arg_dumpInterval);
        count *= 60;
        dumpTimeInterval( count );
    }
    if(givenOption(arg_maxLoops)){ // loops
        evalOption(count,100000,"",arg_maxLoops);
        maxLoop( count );
    }
    if(givenOption(arg_maxIdleLoops)){
        evalOption(count,10000,"",arg_maxIdleLoops);
        maxIdleLoops( count );
    }
    if(givenOption(arg_endQ)){
        // ignored if -1
        evalOption(d,-1.0,"",arg_endQ);
        endQ( d );
    }
    if(givenOption(arg_maxTraceInterval)){ // loops
        evalOption(count,1000,"",arg_maxTraceInterval);
        maxTraceInterval( count );
    }
    if(givenOption(arg_nrOfSequences)){
        evalOption(count,4,"",arg_nrOfSequences);
        sequenceCount( count );
    }
    if(givenOption(arg_random)){
        evalOption(s,"time","",arg_random);
        randomSeed( s );
    }
    if(givenOption(arg_crossOverPct)){
        evalOption(d,0.1,"",arg_crossOverPct);
        guidedCrossOverSelectionPct( d );
    }
    return true;
}

////////////////////////////////////////////////////////////////////////
//
RandomOptimizerBASE::~RandomOptimizerBASE(void)
{
}

////////////////////////////////////////////////////////////////
RandomOptimizerBASE::RandomOptimizerBASE(void)
: itsIsDebugStreamOwner(false),itsDebugStream(&cerr),
```

```
  itsDumpFileName(".RandomOptimizerBASEDumpFile"),
  itsDumpReason("none"),
  itsGen(time(0),55), itsIndependent(),
  itsArena(4),
  itsMaxTraceInterval(100), itsIsImproved(false),
  itsEndQ(-1),
  itsGuidedCrossOverSelectionPct( 0.1 ),
  itsPopulationDumpInterval(1000),
  itsRootName("hnco CO-1 #1"),
  itsDebugContextWriter(new AsciiContextWriter(&cerr), IINIT)
{
    for(int i=0; i<sequenceCount(); i++){
        itsArena[i]  = State::Handle(new State,IINIT);
    }
    itsBest          = State::Handle(new State,IINIT);
    itsBestASCount   = 0;
    itsBestLinkCount = 0;
    itsRunTime       = 60 * 60;
    itsEndTime       = time(NULL) + itsRunTime; // nach einer Stunde
    itsNextDumpTime  = time(NULL); // now
    itsInitialTC     = 60;
    itsTC            = 0;
    itsDumpTimeInterval = 15 * 60;
    itsMaxLoopCount  = LONG_MAX;
    itsLastTraceCount= 0;

    itsLastQ = 0.0;  // schlechtester moeglicher Wert, besser == groesser
    itsLastQChanged = 0;
    itsEndQ =    -1; // default is ignore
    itsMaxIdleLoops = 1000;
    for(int i=0; i< countDebugOptions; i++){
        itsDebugFlags[traceQ] = false;
    }
    itsPopulationLoop = 0;
};
//////////////////////////////////////////////////////
//
RandomOptimizerBASE& RandomOptimizerBASE::randomSeed(const MyString& randomText)
{
    MyString t= randomText;
    t.upcase();
    if(t.contains("TIME")){
        itsRndSeed = time(NULL);
    } else if(t.matches(MyRXint)){
        itsRndSeed = atoi(t.chars());
    };
    itsGen = MyACG(itsRndSeed,55);
    return *this;
}
//////////////////////////////////////////////////////
//
void RandomOptimizerBASE::random_shuffle(intVec &x){
    if(x.count()<2)  return;
    for(int i = 0; i<int(float(x.count()) * float(0.76)); i++){
        swap(x[longInRange0(x.count()-1)],x[longInRange0(x.count()-1)]);
    }
}
void RandomOptimizerBASE::random_shuffle(intVec &x, int len){
    if(x.count()<2)  return;
    for(int i = 0; i<int(float(min(x.count(),len)) * float(0.76)); i++){
        swap(x[longInRange0(min(x.count()-1,len-1))],x[longInRange0(min(x.count()-1,len-1))]);
    }
}

//////////////////////////////////////////////////////////////
//: assign an independent
RandomOptimizerBASE& RandomOptimizerBASE::useIndependent(StateIndependent::Handle p ){
    itsIndependent = p;
    for(int i=0; i<sequenceCount();  i++){
        if(!isNull(itsArena[i])) itsArena[i]->useIndependent(itsIndependent);
    }

    itsAtomDist.resize(sequenceCount(),sequenceCount()+1);
    itsQDist.resize(sequenceCount(),sequenceCount()+1);
    itsBest->useIndependent(itsIndependent);
    itsPerAS.resize(protein()->length());
    if(!isNull(usedIndependent()->allCXList()))
        itsPerCX.resize(usedIndependent()->allCXList()->length());
    peakPool()->getList("hnco").length();
    const DLList<Peak::Handle> pList = peakPool()->getList("hnco");
    long maxPID = 0;
    for(Pix px = pList.first(); px ; pList.next(px)){
        maxPID = max( maxPID, long(pList(px)->getID()));
    }
    itsPerBase.resize( maxPID );

    itsAtomDist.fill(0);
    itsQDist.fill(0);
    itsPerAS.fill(0);
    itsPerBase.fill(0);
    itsPerCX.fill(0);
    itsAtomDist.fill(0);
    itsQDist.fill(0);

    return *this;
}
//////////////////////////////////////////////////////
//
RandomOptimizerBASE&  RandomOptimizerBASE::sequenceCount(int l){
    itsArena.resize(l);
    for(int i=0; i<sequenceCount();  i++){
        if(isNull(itsArena[i]))
            itsArena[i] = State::Handle(new State,IINIT);

        if(!isNull(itsIndependent))
            itsArena[i]->useIndependent(itsIndependent);

        itsAtomDist.resize(sequenceCount(),sequenceCount()+1);
        itsQDist.resize(sequenceCount(),sequenceCount()+1);
        itsAtomDist.fill(0);
        itsQDist.fill(0);
    }
    return *this;
};

//////////////////////////////////////////////////////
//
void RandomOptimizerBASE::initState(void)
{
    itsLoopCount    = 0;
    for(int seqIndex = 0; seqIndex < sequenceCount(); ++seqIndex){
        itsArena[seqIndex]->clearState();
    }
    itsBest ->clearState();
}
//////////////////////////////////////////////////////
//virtual !
void RandomOptimizerBASE::restartRun(void)
{
    itsLoopCount    = 1;
    itsEndTime      = time(NULL) + itsRunTime;
    itsNextDumpTime = time(NULL) + itsDumpTimeInterval;
    itsLastTraceCount = 0;
    itsLastQChanged = 0;
    itsLastQ        = 0.0;
    itsTC           = initialTC();
    itsIsImproved   = false;
    itsLastTraceCount = 0;

    itsPopulationLoop= 0;
    itsPerAS.fill(0);
    itsPerBase.fill(0);
    itsPerCX.fill(0);
    itsAtomDist.fill(0);
    itsQDist.fill(0);
}
//////////////////////////////////////////////////////
//
void RandomOptimizerBASE::optimize(void)
{
    /* EMPTY */
}
//////////////////////////////////////////////////////
// (DEBUGGING helper)
```

```
//: teste ob alle erworbenen AS auch Mitglieder des Cache sind
//  wenn nicht, ist eine der INVARIANTEN verletzt worden.
//
//////
// checkUnlockedToo   : true, die Invariante soll auch fuer
//                      die noch nicht erworbenene AS getestet werden
//virtual
#ifdef PRODUCTION
       inline
#endif
void RandomOptimizerBASE::invariante_checkAllCacheMembers(
        int seqIndex,
        bool checkUnlockedToo/* = true */
        )const
{
#ifdef PRODUCTION
    return;
#endif
    DEBUGTracedFrame;
    // discards const on (*this)
    State &S = *(const_cast<Self*>(this)->stateAt(seqIndex));
    ActiveAccount active(S.account());

    ASSERT(1000 > S.asCount());// 1000 >> any reasonable protein
    int stack[1000];
    int top = 0;

    for(int pos = 0; pos < S.asCount(); ++pos){
        if(!isNull(S.asAt(pos))){
            // wenn der CX weder erworben noch
            //   (test fuer nicht erworbene erforderlich)
            //   dann ignoriere diesen CX
            if(!S.asLockAt(pos).isAquired() && ! checkUnlockedToo)
                continue;
            if(!S.asListAt(pos)->contains(S.asAt(pos))){
                stack[top++] = pos+1;
            }
        }
    }
    if( top>0 ){
        ErrMessage err;
        err << error << "INVARIANT VIOLATION: assigned AS not in ASList!"
            << "\nASNR[";
        for(int i=0; i<top; i++){
            err << stack[i];
            if(i+1<top) err << ", ";
        }
        err << "]\n";
    }
}

/////////////////////////////////////////////////////
// (HELPER fuer optimize)
//: suche AS die nicht erworben sind und ersetze sie durch eine leere Stelle
void RandomOptimizerBASE::removeNotLockedAt(int seqIndex)
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    for(int pos = 0; pos < S.asCount(); ++pos){
        if( !isNull(S.asAt(pos)) && !S.asLockAt(pos).isAquired() ){
            S.clearAS(pos,true);
        }
    }
}
/////////////////////////////////////////////////////
// (HELPER fuer optimize)
//: suche AS die nicht erworben sind und ersetze sie durch eine leere Stelle
void RandomOptimizerBASE::removeNotLinkedAt(int seqIndex)
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    if(itsCanRemoveNotLinked){
        for(int pos = 0; pos < S.asCount(); ++pos){
            // entferne jede AS die:
            //   einen Nachbarn hat und nicht mit ihm verknüpft ist.
            //   wenn zwei Nachbarn existieren müssen beide vernuepft sein!
            if(!isNull(S.asAt(pos))){
                if(
#ifdef REMOVE_UNLINKED_COND_BOTH_SIDES_UNLOCKED
                        (pos < S.linkCount()
                         && isNull(S.linkAt(pos)) && !isNull( S.asAt(pos+1 )))
                        &&
                        (pos > 0
                         && isNull(S.linkAt(pos-1))&& !isNull( S.asAt(pos-1 )))
#endif //REMOVE_UNLINKED_COND_BOTH_SIDES_UNLOCKED
#ifdef REMOVE_UNLINKED_COND_ONE_SIDE_UNLOCKED
                        ( pos < S.linkCount()
                          && isNull(S.linkAt(pos))  && !isNull(S.asAt(pos+1)) )
                        ||
                        ( pos > 0
                          && isNull(S.linkAt(pos-1)) && !isNull(S.asAt(pos-1)) )
#endif //REMOVE_UNLINKED_COND_ONE_SIDE_UNLOCKED
                        )
                    {
#ifdef  REMOVE_UNLINKED_RANDOM_KEEP
                    if( getRandomDouble() < 0.5 )
#endif //REMOVE_UNLINKED_RANDOM_KEEP
                        S.clearAS(pos,true);
                    }
            } // if AS at pos not NULL
        }
    } // if removeNotLinked
}
/////////////////////////////////////////////////////////////
//: loesche den CX an position pos (ACTION)
//
void RandomOptimizerBASE::removeRangeAt(int seqIndex)
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    // suche eine zugewiesene position mit cx im bereich [1..len-1]
    int pos;
    if((pos = chooseAS(S,10)) == -1) return ;
    int lastPos = min(pos + 4,S.asCount());
    for(; pos < lastPos; ++pos){
        S.clearAS(pos,true);
    }
}

/////////////////////////////////////////////////////////////
// choose an already assigned  AS position
// in range [1..protein.length-1]
int RandomOptimizerBASE::chooseAS(const State &S, int  maxTry)
{
    int count = maxTry-1;
    int pos = longInRange0(S.asCount()-2)+1;

    for(; count > 0; count--){
        if( S.canChangeAt(pos)      &&
            !isNull(S.asAt(pos)))
        {
            break;
        }
        pos = longInRange0(S.asCount()-2)+1;
    }

    if(count <= 0) return -1;
    if(!isNull(S.asAt(pos))){
        // wie kann ein CX zugeordnet sein und gleichzeitig die
        // Liste der CX leer sein?
        ASSERT(S.asListAt(pos)->length() != 0);
        if(S.asListAt(pos)->length() == 0)
            ASSERT_HOOK();
        return pos;
    } else {
        return -1;
    }
}
/////////////////////////////////////////////////////////////
// choose an EMPTY  AS position which can be assigned
// in range [1..protein.length-1]
int RandomOptimizerBASE::chooseEmptyAS(const State &S, int  maxTry)
```

```
{
    int count = maxTry-1;
    int pos = longInRange0(S.asCount()-2)+1;

    for(; count > 0; count--){
        if( S.canChangeAt(pos)   &&
            isNull(S.asAt(pos))&&
            S.asListAt(pos)->length() != 0 // avoid empty positions
            )
        {
            break;
        }
        pos = longInRange0(S.asCount()-2)+1;
    }

    if(count <= 0) return -1;
    if(isNull(S.asAt(pos))){
        ASSERT(S.asListAt(pos)->length() != 0);
        if(S.asListAt(pos)->length() == 0)
            ASSERT_HOOK();
        return pos;
    } else {
        return -1;
    }
}
////////////////////////////////////////////////////////////
// choose a new AS Assignment (iterator)
// PRECOND:
// don't test AQUIRE
ContextList::Iterator RandomOptimizerBASE::chooseIterAt(const State &S, int pos, int  maxTry)
{
    // iterator in der liste der Cx
    int iterCount;
    ContextList::Iterator cxIter;
    int count = maxTry;
    do {
        count--;
        iterCount = longInRange0(S.asListAt(pos)->length()-1);
        cxIter = S.asListAt(pos)->last();
        for(int i=0; cxIter && i < iterCount; i++, --cxIter);
    } while((!cxIter || isNull(*cxIter)) && count>0);
    return cxIter;
}

////////////////////////////////////////////////////////////
//: bilde eine Zufallszuordnung, (ACTION)
//  das Ergebnis muss nicht linkbar sein.
/////
//  Ausgangszustand: irgendein State
void RandomOptimizerBASE::randomInitAt(int seqIndex)
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    CHECK_LOCK;
    S.clearState();

    const double initialMinAssignedCxCountPct = 0.70;
    int minAssignedCxCount = int(S.asCount() * initialMinAssignedCxCountPct);
    for(int n=0; n< minAssignedCxCount; n++) {
        // position im protein
        int pos = chooseEmptyAS(S);
        if(pos == -1) {
            return;
        }

        S.clearAS(pos,true);

        // iterator in der liste der Cx
        ContextList::Iterator iter = chooseIterAt(S,pos);
        if(!iter) {
            return;
        }

        // weise den Cx zu.
        ContextLock     currASLock;
        currASLock.aquire(const_cast<Context::Handle&>(*iter));
        if(iter && currASLock.isAquired()){
            // setze *iter an diese Position
            currASLock.release(true);
            S.putASAt(pos,*iter);
            S.pushChange(pos);
            evaluateLocalAt(seqIndex,pos);

        }
    }
}

////////////////////////////////////////////////////////////////////////
//: wendet einen Auschnitt eines StateDiffs auf einen State an.
//  dannach werden die neuen CX erworben und dann alle alten Positionen,
//  die im Konflikt mit den neuen stehen, verworfen.
//  Qualität und Locks im Endergebnis sind gültig.(im Gegensatz zum applyDiff)
void     RandomOptimizerBASE::applyDiffRangeAt(
        int destIndex1,

        const StateDiff & diff,
        // erste Aktion im Diff
        int start,
        // Anzahl der Folgeaktionen. (-1 == alle in diff)
        int len ,
        // SWAP's are different from ADD and SUB operations,
        // not all cx combinations are swappable.
        // So, SWAP's are not always applicable.
        // iff they are applied as two sub and two add CX Operations!
        //     NOT as an exchange CX !
        bool apply_swaps /* = true */
        )
{
    DEBUGTracedFrame;
    State &destS1 = *stateAt(destIndex1);

    State &srcS2 = *diff.end(); // state where ADD operations lead to.

#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "applyDiffRangeAt ("
         << destIndex1+1 <<") , range["<< start <<"," << len<<"] <<--- "
         << " Q(dest1)=" << destS1.quality()<<endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q
    CheckLockState checkState1s( &destS1 );
    start = max(start,0);
    if( len < 0 ){
        len = diff.top()-start;
    }
    int end = min(start + len, diff.top());

    intVec isLocked(destS1.asCount()); isLocked.fill(0);
    {ActiveAccount active(destS1.account());
    // unlock all AS
    for(int pos=0; pos<destS1.asCount(); pos++){
        // unlock, don't drop resource
        if(!isNull(destS1.asAt(pos)))
            destS1.asLockAt(pos).release(false);
    }
    // first run, apply SUB and op_sub part of SWAP and ADD
    for(int k = start; k<end; ++k){
        if( diff.at(k).operation() == action_rec::op_sub
            || diff.at(k).operation() == action_rec::op_add)
        {
            destS1.clearAS( diff.at(k).position() );
        } else if(apply_swaps && diff.at(k).operation() == action_rec::op_swap)
        {
            destS1.clearAS( diff.at(k).position() );
            destS1.clearAS( diff.at(k).position2() );
        }
    }
    // second run, apply all additions/swaps
    for(int k = start; k<end; ++k){
        // nur die ADD/SUB/SWAP operationen
        // tragen zu den Veränderungen bei, Links
        // entstehen implizit durch die anderen
        // Operationen.
        if(isLinkOp(diff.at(k))) continue;
```

```
    if(diff.at(k).operation() == action_rec::op_add)
    {
        int pos = diff.at(k).position();
        Context::Handle cx;
        if( pos > -1 ) {
            cx = srcS2.asAt(pos);
        }
        destS1.putASAt(pos,cx);
        isLocked[pos] = 1;
        if(!isNull(cx)){
            ASSERT(destS1.asListAt(pos)->contains(cx));
            destS1.pushChange(pos);
            evaluateLocalAt(destIndex1,pos);
            if(!destS1.asLockAt(pos).isAquired()){
                isLocked[pos] = -1;
            }
        }
    } else if(apply_swaps && diff.at(k).operation() == action_rec::op_swap)
    {
        int pos2 = diff.at(k).position2();
        Context::Handle cx;
        if( pos2 > -1 ) {
            cx = srcS2.asAt(pos2);
        }
        destS1.putASAt(pos2,cx);
        isLocked[pos2] = 1;
        if(!isNull(cx)){
            ASSERT(destS1.asListAt(pos2)->contains(cx));
            destS1.pushChange(pos2);
            evaluateLocalAt(destIndex1,pos2);
            if(!destS1.asLockAt(pos2).isAquired()){
                isLocked[pos2] = -1;
            }
        }


        int pos = diff.at(k).position();
        if( pos > -1 ) {
            cx = srcS2.asAt(pos);
        } else
            cx = Context::Handle();
        destS1.putASAt(pos,cx);
        isLocked[pos] = 1;
        if(!isNull(cx)){
            ASSERT(destS1.asListAt(pos)->contains(cx));
            destS1.pushChange(pos);
            evaluateLocalAt(destIndex1,pos);
            if(!destS1.asLockAt(pos).isAquired()){
                isLocked[pos] = -1;
            }
        }

    }
}

// remove all cx in destS, which collide (can not be simultaneously locked)
// with one of the copied cx
// re-aquire all CX outside copied range
bool repair = true;
for(int pos=0; pos<isLocked.count(); pos++){
    if(isLocked[pos])
        continue;
    if(isNull(destS1.asAt(pos)))
        continue;
    if(!destS1.asLockAt(pos).aquire(const_cast<IMngPointer<Context> &>(destS1.asAt(pos)))){
        destS1.clearAS(pos);
        isLocked[pos] = -2;
    }
}
if( repair ){
    for(int pos=0; pos<isLocked.count(); pos++){
        if(isLocked[pos] != -2)
            continue;
        // wenn es nur einen CX für diese Position gibt
        // und schon einer erfolglos getestet wurde
        // kann es keinen weiteren CX geben der erworben werden kann.
        if(destS1.asListAt(pos)->length()<=1)
            continue;

        // versuche die Position zu reparieren.
        // 1) erst die CX der Eltern.
        // 2) die CX der anderen States im Pool (absteigend).
        // 3) zufällig ODER (impl) bleibt leer.
        ContextLock cxLock;
        bool found = false;

        // versuch 1) Elternteil 1
        Context::Handle currCX = diff.start()->asAt(pos);
        if(cxLock.aquire(currCX))
        {
            found = true;
        }
        currCX = diff.end()->asAt(pos);
        if(!found && cxLock.aquire(currCX))
        {
            found = true;
        }
        for(int sIndex = 0; !found && sIndex<sequenceCount(); ++sIndex){
            currCX = stateAt(sIndex)->asAt(pos);
            if(!found && cxLock.aquire(currCX))
            {
                found = true;
            }
        }
        if( found ){
            cxLock.release(true);
            destS1.putASAt(pos, currCX);destS1.pushChange(pos);
            evaluateLocalAt(destIndex1,pos);
        }
    }
}

} // end ActiveAccount

// should we protect the new areas ?
// foreach pos (firstPos .. firstPos+len-1)
//    dest1.loopCountAt(pos) = dest.epoch() + protect_period;
//    dest2.loopCountAt(pos) = dest.epoch() + protect_period;
//
#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "applyDiffRangeAt ("
         << destIndex1+1 <<") , range["<< start <<"," << len<<"] <<--- "
         <<"  Q(dest1)="<< destS1.quality()
         << endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q
};
////////////////////////////////////////////////////////////////////////////
//:  generates two new States by merging two other States
//   kopiere Stuecke aus srcS1 und srcS2 nach destS1 und destS2
//   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
void     RandomOptimizerBASE::crossOverAt(
        int destIndex1,int destIndex2, // dest  [ 0 .. sequenceCount()-1]
        int srcIndex1 ,int srcIndex2   // src   [-1 .. sequenceCount()-1]
        )
{
    DEBUGTracedFrame;
    ASSERT( destIndex1 != srcIndex1 );
    ASSERT( destIndex1 != srcIndex2 );
    ASSERT( destIndex2 != srcIndex1 );
    ASSERT( destIndex2 != srcIndex2 );
    State &destS1 = *stateAt(destIndex1);
    State &destS2 = *stateAt(destIndex2);
    State &srcS1  = *stateAt(srcIndex1);
    State &srcS2  = *stateAt(srcIndex2);
#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "SEQUENCES["
         << destIndex1+1 <<", "<< destIndex2+1 <<", "
         << srcIndex1+1 <<", "<< srcIndex2+1
         << "] singleCrossOverAt                Q(src1)="
         << srcS1.quality()<<"  Q(src2)="<< srcS2.quality()<< endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q
    CheckLockState checkState1s( &srcS1 );
    CheckLockState checkState2s( &srcS2 );
    CheckLockState checkState1d( &destS1 );
    CheckLockState checkState2d( &destS2 );
```

```
    // waehle einen len Positionen langen Bereich aus
    const int  minLen = 8;
    int        maxLen = int(destS1.asCount()/3);
    int len      = longInRange0(maxLen-minLen)+minLen;
    int firstPos = longInRange0(destS1.asCount()-len-1);

    // kopiere inv-range 1 von seq4 -> seq1
    // kopiere range 1     von seq3 -> seq1
    {ActiveAccount active(destS1.account());
    destS1.copyFrom(srcS2);
    crossCopyRange(destIndex1,   srcIndex1, firstPos, len);
    }

    // kopiere inv-range 1 von seq3 -> seq2
    // kopiere range 1     von seq4 -> seq2
    {ActiveAccount active(destS2.account());
    destS2.copyFrom(srcS1);
    crossCopyRange(destIndex2,  srcIndex2, firstPos, len);
    }
    // should we protect the new areas ?
    // foreach pos (firstPos .. firstPos+len-1)
    //    dest1.loopCountAt(pos) = dest.epoch() + protect_period;
    //    dest2.loopCountAt(pos) = dest.epoch() + protect_period;
    //
#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "singleCrossOver ("
         << destIndex1+1 <<", "<< destIndex2+1 <<" <<--- "
         << srcIndex1+1 <<", "<< srcIndex2+1
         << ")  range("<<firstPos+1<<" ... "<<firstPos+len
         <<")  Q(dest1)="<< destS1.quality()<<"  Q(dest2)="<< destS2.quality()
         << endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q

};
////////////////////////////////////////////////////////////////////////
//:  generates one of two new States by merging two other States
//   kopiere Stuecke aus srcS1 und srcS2 nach destS1 und destS2
//   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
/////
// generateFirst:
//    true generates dest1 , false generates dest2
void     RandomOptimizerBASE::singleCrossOverAt(
        int destIndex1,
        int srcIndex1 ,int srcIndex2,
        bool generateFirst  // true generate dest1 , false generates dest2
        )
{
    DEBUGTracedFrame;
    ASSERT( destIndex1 != srcIndex1 );
    ASSERT( destIndex1 != srcIndex2 );

    State &destS1 = *stateAt(destIndex1);
    State &destS2 = *stateAt(destIndex1);
    State &srcS1  = *stateAt(srcIndex1);
    State &srcS2  = *stateAt(srcIndex2);

#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "SEQUENCES["
         << destIndex1+1 <<" <<--  "
         << srcIndex1+1 <<", "<< srcIndex2+1
         << "] singleCrossOverAt("<<(generateFirst?"FIRST":"SECOND")<<")  Q(src1)="
         << srcS1.quality()<<"  Q(src2)="<< srcS2.quality()<< endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q
    CheckLockState checkState1s( &srcS1 );
    CheckLockState checkState2s( &srcS2 );
    CheckLockState checkState1d( &destS1 );
    CheckLockState checkState2d( &destS2 );
    // waehle einen len Positionen langen Bereich aus
    const int  minLen = 8;
    int        maxLen = int(destS1.asCount()/3);
    int len      = longInRange0(maxLen-minLen)+minLen;
    int firstPos = longInRange0(destS1.asCount()-len-1);


    if(generateFirst)
    // kopiere inv-range 1 von seq4 -> seq1
    // kopiere range 1     von seq3 -> seq1
    {ActiveAccount active(destS1.account());
    destS1.copyFrom(srcS2);
    crossCopyRange(destIndex1,   srcIndex1, firstPos, len);
    }

    if(!generateFirst)
    // kopiere inv-range 1 von seq3 -> seq2
    // kopiere range 1     von seq4 -> seq2
    {ActiveAccount active(destS2.account());
    destS2.copyFrom(srcS1);
    crossCopyRange(destIndex1,  srcIndex2, firstPos, len);
    }
    // should the new areas be protected?
    // foreach pos (firstPos .. firstPos+len-1)
    //    dest1.loopCountAt(pos) = dest.epoch() + protect_period;
    //    dest2.loopCountAt(pos) = dest.epoch() + protect_period;
    //
#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "singleCrossOver("<<(generateFirst?"FIRST":"SECOND")<<") ("
         << destIndex1+1 << " <<--- "
         << srcIndex1+1 <<", "<< srcIndex2+1
         << ")  range("<<firstPos+1<<" ... "<<firstPos+len<<")  Q(dest1)="
         << destS1.quality()
         << endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q

};
////////////////////////////////////////////////////////////////////
// kopiere den Bereich [srcPos .. srcPos+len-1]  nach [destPos .. destPos+len-1]
// innerhalb einer Sequenz
// achtet auf ueberlappte Bereiche
//
// Ueberlappung:
//  getrennt                          iter++
//  ueberlappt,  dest vor src         iter++
//  ueberlappt,  src vor dest         iter--
//
void     RandomOptimizerBASE::copyRange(int seqIndex, int destPos, int srcPos, int len)
{
    DEBUGTracedFrame;
    if(srcPos == destPos) return ;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    ASSERT( srcPos+len-1 <S.asCount() );
    ASSERT( destPos+len-1<S.asCount() );

    Context::Handle cx1;
    Context::Handle cx2;
    int pos1;
    int pos2;
    if(destPos>srcPos && srcPos+len-1 >= destPos) {
        // src startet vor dest && ueberlappt
        for(pos1=srcPos+len-1, pos2=destPos+len-1; pos1>=srcPos; pos1-- , pos2--){
            cx1 = S.asAt(pos1);
            cx2 = S.asAt(pos2);
            S.clearAS(pos1,true);
            S.clearAS(pos2,true);
            if(!isNull(cx1)){
                ASSERT(S.asListAt(pos2)->contains(cx1));
                S.putASAt(pos2,cx1);
            }
            S.pushChange(pos2);
            evaluateLocalAt(seqIndex,pos2);
            if(!isNull(cx2) && S.asListAt(pos1)->contains(cx2)){
                S.putASAt(pos1,cx2);S.pushChange(pos1);
                evaluateLocalAt(seqIndex,pos1);
            }
        }
    } else {
        // getrennte Bereiche oder dest vor src
        for(pos1=srcPos, pos2=destPos; pos1<srcPos+len; pos1++ , pos2++){
            cx1 = S.asAt(pos1);
            cx2 = S.asAt(pos2);
            S.clearAS(pos1,true);
            S.clearAS(pos2,true);
            if(!isNull(cx1)){
                ASSERT(S.asListAt(pos2)->contains(cx1));
```

```
                S.putASAt(pos2,cx1);     S.pushChange(pos2);
                evaluateLocalAt(seqIndex,pos2);
            }
            if(!isNull(cx2) && S.asListAt(pos1)->contains(cx2)){
                S.putASAt(pos1,cx2);S.pushChange(pos1);
                evaluateLocalAt(seqIndex,pos1);
            }
        }
    }
}
////////////////////////////////////////////////////////////////////////////
//: copy one range of AS from one State to another.
void   RandomOptimizerBASE::crossCopyRange(
        int destIndex , // dest   [ 0 .. sequenceCount()-1]
        int srcIndex ,  // src    [-1 .. sequenceCount()-1]
        int firstAS, int len
        )
{
    DEBUGTracedFrame;
    State &destS = *itsArena[destIndex];
    State &srcS  = *stateAt(srcIndex);

    // clear range [firstAS .. firstAS + len -1] in seq (seqIndex1)
    // in srcS active account
    ActiveAccount active(destS.account());
    for(int pos=0; pos<len; pos++){
        destS.clearAS(pos+firstAS);
    }
    // unlock all AS
    for(int pos=0; pos<destS.asCount(); pos++){
        if(!isNull(destS.asAt(pos)))
            destS.asLockAt(pos).release(false); // unlock, don't drop resource
    }
    // copy all cx from srcS
    intVec ids (len);
    for(int pos=0; pos<len; pos++){
        const Context::Handle& cx =  srcS.asAt(pos+firstAS);
        destS.putASAt(pos+firstAS,cx);  destS.pushChange(pos+firstAS);
        if( !isNull(cx) ){
            evaluateLocalAt(destIndex,pos+firstAS);
        }
        ids[pos] = destS.basePeakIDAt(pos+firstAS);
    }
    // remove all cx in destS which collide (can not be simultaneously locked)
    // with one of the copied cx
    // re-aquire all CX outside copied range
    for(int pos=0; pos<destS.asCount(); pos++){
        if(pos==firstAS){
            pos = firstAS+len-1; // exclude range [firstAS .. firstAS+len-1]
            continue;
        }
        if(!destS.asLockAt(pos).aquire(const_cast<IMngPointer<Context> &>(destS.asAt(pos)))){
            destS.clearAS(pos);
        }
    }
}
////////////////////////////////////////////////////////////////////////////
//
//
//////
// return: index of as position or -1
//
int  RandomOptimizerBASE::findFirstMatch(
        int seqIndex,
        int firstPos,   //  anfang des bereichs (as index)
        int asLen,      //  laenge des bereichs (== linkLen+1)
        const intVec& destPos, // alternative positionen (other as index)
        int posLen      // genutzte laenge in destPos
        )
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    for(int as = 0; as < posLen; as++){
        // suche die erste Position im alternativen Bereich
        // die nicht den entsprechenden Cx im Src Bereich
        // aufnehmen kann.
        int p = 0;
        if(destPos[as]+asLen >= S.asCount()) continue;
        for(;
            (p < asLen &&
             S.asListAt(destPos[as]+p)->contains(S.asAt(firstPos+p)));
            p++){
            ASSERT(destPos[as]+p< S.asCount());
            if(p+1< asLen){
                ASSERT(!isNull(S.asAt(firstPos+p+1)));
                ASSERT(!isNull(S.asListAt(destPos[as]+p+1)));
            }
        }

        if(p==asLen){
            // success, range [destPos[as] .. destPos[as]+asLen-1]]
            // can take all cx from range [firstPos .. firstPos+asLen-1]
            for(int i = 0; i< asLen; i++){
                ASSERT(!isNull(S.asAt(firstPos+i)));
                ASSERT(S.asListAt(destPos[as]+i)->length()!=0);
                ASSERT(S.asListAt(destPos[as]+i)->contains(S.asAt(firstPos+i)));
            }
            return as;
        }
    }
    return -1;
}
////////////////////////////////////////////////////////////////////
//
void    RandomOptimizerBASE::localEvaluateInRange(
        int seqIndex,
        int firstPos, int len
        )
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    int pos1;
    for(pos1=firstPos; pos1<firstPos+len; pos1++){
        S.pushChange(pos1);
        evaluateLocalAt(seqIndex,pos1);
    }

}
////////////////////////////////////////////////////////////////////
//: evaluate the quality of one AS
//  will not care about link quality
void RandomOptimizerBASE::evaluateLocalAt(int seqIndex, int pos)
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    if(SHOW_Q_TRACE) cout <<"SEQ " << seqIndex << " AS "<< pos << "  q:=" << S.quality();
    if(SHOW_Q_TRACE) cout <<"  asQ=" << S.asQAt(pos);
    // all local Q was removed from S.quality()
    // now rebuild it
    S.validAS().set(pos,false);
    S.asQAt(pos) = Fuzzy::False;
    if(!isNull(S.asAt(pos))){
        if(S.asLockAt(pos).aquire(const_cast<IMngPointer<Context> &>(S.asAt(pos)))){
            if(SHOW_Q_TRACE) cout <<"  addLockQ " ;
            S.validAS().set(pos,true);
        } else {
            ERRMESSAGE(err);
            err << warning
                << "loop="    << itsLoopCount
                // << " action=" << action2Name(itsLastAction[seqIndex])
                << " can not aquire CX at AS index="<<pos
                << " in SEQ#" <<seqIndex+1
                << eom;
            S.clearAS(pos);
            return ;
        }
        usedIndependent()->calculator()->calculateASQAt(S,pos);

        if(SHOW_Q_TRACE) cout <<"  exprQ=" << S.asQAt(pos) ;
        S.quality() += S.asQAt(pos);
        if(SHOW_Q_TRACE) cout <<"  addExprQ " ;
    }
```

```
    if(SHOW_Q_TRACE) cout <<"  Q:=" << S.quality() << endl;
}

//////////////////////////////////////////////////////////////////
//: evaluateAt:
//  re-evaluates a partially evaluated Sequence.
//  recalculates all CX with changedAS set.
//    recalculation initiates calculation of neighbour links.
//   Q := sum( AS quality ) + sum ( link quality ) + sum( AS is aquired & locked )
//  Thus, unaquirable CX recieve a additional malus.
//
void RandomOptimizerBASE::evaluateAt(int seqIndex)
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    // nothing to do for AS
#ifdef DEBUG_CALCULATION_Q
    cout << "evaluateAt("<<seqIndex<<") START: "<< endl;
    cout <<  "evaluateAt("<<seqIndex<<") vorher  Q:=" << S.quality()<<endl;
    cout <<  "evaluateAt("<<seqIndex<<") changeStack:=[" ;
    for(int i = 0; i < S.changeCount(); ++i){
        int asIndex = S.changeAt(i);
        cout << asIndex << ',';
    }
    cout << "]"<<endl;
#endif
    // collect all links that have to be recalculated.
    // stores all changed links on stack.
    intVec linkStack(S.changeCount()*2); linkStack.fill(-1);
    int    linkStackTop=0;  // == len of used range, == index of first unused
    for(int i = 0; i < S.changeCount(); ++i){
        int asIndex = S.changeAt(i);
        // fuer den linken und rechten Link
#ifdef DEBUG_CALCULATION_Q
        cout <<"evaluateAt("<<seqIndex<<") For AS index=" << asIndex
             <<" is "<<(!isNull(S.asAt(asIndex))?"assigned":"empty");
        if(!isNull(S.asAt(asIndex)))
            cout <<", ("<<(!isNull(S.asAt(asIndex))?"":"NOT")<<" locked)";
        cout <<endl;
#endif
        int linkIndex=asIndex-1;
        if(asIndex==0) linkIndex++; // AS nr 0 has no left side link
        for(; linkIndex<asIndex+1;linkIndex++){
            if(linkIndex==S.linkCount()) continue; // last link has i-1 link only
            if(SHOW_Q_TRACE) cout <<"SEQ " << seqIndex << " linkNr="<<linkIndex;
            if(SHOW_Q_TRACE) cout <<"  seqQ:="  << S.quality();
            if(SHOW_Q_TRACE) cout <<"  asQ=" << S.linkQAt(linkIndex) <<endl;

            if(S.changedAS().get(linkIndex) || S.changedAS().get(linkIndex+1)){
                // wenn beide AS gueltig sind
                if(S.validAS().get(linkIndex) &&
                   (linkIndex+1<S.asCount() && S.validAS().get(linkIndex+1)) ){
#ifdef DEBUG_CALCULATION_Q
                    cout <<"evaluateAt("<<seqIndex<<") push link index " << linkIndex <<endl;
#endif
                    // push on stack if not already in stack
                    int n;
                    for(n=0;n<linkStackTop && linkStack[n] != linkIndex; n++);
                    if(n==linkStackTop) linkStack[linkStackTop++] = linkIndex;
                }
            }
        }
    }
    // recalculate and build all Links on the linkStack
#ifdef DEBUG_CALCULATION_Q
    cout <<"evaluateAt("<<seqIndex<<") vor link  Q:=" << S.quality()<<endl;
#endif
    for(int n=0;n<linkStackTop; n++){
        int linkIndex = linkStack[n];
#ifdef DEBUG_CALCULATION_Q
        cout <<"evaluateAt("<<seqIndex<<") calculate link at index "
             << linkIndex <<endl;
#endif
        usedIndependent()->calculator()->calculateLinkQAt( S, linkIndex );
    }
#ifdef DEBUG_CALCULATION_Q
    cout <<"evaluateAt("<<seqIndex<<") nachher Q:=" << S.quality()<<endl;
    cout <<"evaluateAt("<<seqIndex<<") FINISH: "<< endl;
#endif
    if(SHOW_Q_TRACE) cout <<"  Q:=" << S.quality();
    if(SHOW_Q_TRACE) cout <<endl;

    while( S.changeCount() ){
        S.declareUnchanged(S.popChange());
    }
    if(SHOW_Q_TRACE) cout <<" Q:="<<S.quality()<<endl;
};
// berechne itsBestQ ... neu
void RandomOptimizerBASE::rebuildBestQ(void){
    rebuildQAt(-1);  // -1 == mapps to itsBest
    if(itsLastQ > itsBest->quality()){
        itsLastQ = 0; // ::evaluateArena will recalculate it
    }
};
//////////////////////////////////////////////////////////////////
//: berechne die Qualitaet voellig neu.
//  verlaesst sich nur auf  itsAS
//
void RandomOptimizerBASE::rebuildQAt(int seqIndex)
{
    DEBUGTracedFrame;
    State &S = *stateAt(seqIndex);
    ActiveAccount active(S.account());
    // setze alles auszer itsAS auf 0
    if(SHOW_Q_TRACE) cout <<"rebuildQ ("<<seqIndex<<")"<< endl;
    Context::Handle cxNULL;
    S.usedCount() = 0; // requires explicit handling of usedCount !
    for(int pos = 0; pos < S.asCount(); ++pos){
        Context::Handle cx = S.asAt(pos); // backup data
        S.clearAS(pos,true);              // cleanup
        S.putASAt(pos,cx);                // put it back
        if(!isNull(cx)) S.usedCount()++;  // putASAt(...) does not cover this case. s.o.
    }
    S.quality()   = 0.0;
    S.clearChanges();
    for(int pos = 0; pos < S.asCount(); ++pos){
        S.pushChange(pos);
        evaluateLocalAt(seqIndex,pos);
    }
    evaluateAt(seqIndex);
};

//////////////////////////////////////////////////////////////////
//: decide which objects should stay
void  RandomOptimizerBASE::evaluateArena(void){
    // noch wissen wir nicht ob eine Verbesserung eingtreten ist.
    itsIsImproved = false;
};
//////////////////////////////////////////////////////////////////
bool RandomOptimizerBASE::abbruchbedingung(void)
{
    itsLoopCount++;
#ifdef  WITH_TIME_STAMPS
    debugTraceLoop();
#endif // WITH_TIME_STAMPS
    if( itsTC-- <= 0){
        // if some time (loops) has elapsed,
        // recheck system time.
        itsTC = initialTC(); // some small loop count, NOT seconds !
        time_t currTime = time(NULL);
        if( itsEndTime < currTime){
            exitReason("max time reached");
            return true;
        }
        if( currTime - itsDumpTimeInterval >=  itsNextDumpTime){
            exitReason("dump intervall reached");
            dumpState();
            if( itsNextDumpTime < itsDumpTimeInterval){
                itsNextDumpTime = currTime;
            } else {
                while(currTime - itsDumpTimeInterval >=  itsNextDumpTime)
                    itsNextDumpTime += itsDumpTimeInterval;
            }
```

```
        }
    }
    if( itsLoopCount  >= itsMaxLoopCount ){
        exitReason("max loop count reached");
        return true;
    }
    const double epsilon = 1e-4;
    if( itsEndQ > 0 && (itsBest->quality()-itsEndQ) >= -epsilon){
        exitReason("quality exceeds end quality");
        return true;
    }
    if(abbruchbedingung_qualityChange())
        return true;
    return false;
}
bool RandomOptimizerBASE::abbruchbedingung_qualityChange(void)
{
    if(itsLastQChanged < itsLoopCount - itsMaxIdleLoops * 2
       && itsLastQ >= itsBest->quality()
       ){
        exitReason("no quality improvement for (2*maxIdleLoop) loops");
        return true;
    }
    return false;
}
//////////////////////////////////////////////////////////////////////
//
bool RandomOptimizerBASE::loadConfig(istream & is)
{
    enum{ ok = 0 , fail = 1 };
    int errPos=0;
    AssertHook noStackDump(NULL);

    ReadBlock  block2(is,"CONFIG");
    ReadBlock  block3(is,"ACTIONCONFIG");
    if(!block2.open())       ERROR_RECOVER;

    if(!block2.skipToEnd())  ERROR_RECOVER;

    if(!block3.open())      ERROR_RECOVER;

    if(!block3.skipToEnd()) ERROR_RECOVER;
    return ok;
  error_recover:
    ErrMessage err;
    err << error
        << " parsing optimizer seed failed. (code =  " << errPos
        << " at line " << streampos2lineNr(is,is.tellg())
        << ")"  << eom;
    return fail;
}

//////////////////////////////////////////////////////////////////////
//
bool RandomOptimizerBASE::loadState(
        const MyString& name, PeakPool::Handle pool, PeakNet::Handle net)
{
    enum{ ok = 0 , fail = 1 };
    int errPos=0;
    AssertHook noStackDump(NULL);

    fstream is(name, ios::in|ios::nocreate);
    if(!is){
        ErrMessage err;
       err << error << "loadState:"<< "can't open config file \"" <<name<<"\"";
        return fail;
    }

    ReadBlock  block1(is,"RANDOMSEQUENCEOPTIMIZERARENA");
    // block 2 und 3 in loadConfig
    ReadBlock  block4(is,"BEST");
    ReadBlock  block5(is,"ARENA");
    MyString version;
    int arenaLen;

    if(!block1.open())  ERROR_RECOVER;
    if(parseDQField(is, "VERSION",version)) ERROR_RECOVER;
    if(version >= "1.01"){
        MyString temp;
        if(parseDQField(is, "DUMPTIME", temp )) ERROR_RECOVER;
        if(version >= "1.02"){
            if(parseDQField(is, "DUMPREASON", temp )) ERROR_RECOVER;
        }
    }

    if(loadConfig(is)) ERROR_RECOVER;


    if(!block4.open())  ERROR_RECOVER;
    itsBest = State::Handle(new State,IINIT);
    if(itsBest->createState(is, itsIndependent, *pool, *net))  ERROR_RECOVER;
    if(!block4.close()) ERROR_RECOVER;

    if(!block5.open())  ERROR_RECOVER;
    if(parseField(is, "LENGTH", arenaLen) ) ERROR_RECOVER;
    itsArena.resize(arenaLen);
    for(int i=0; i< arenaLen; i++){
        int pos;
        if(parseField(is, "STATE", pos) || pos-1 != i) ERROR_RECOVER;
        itsArena[i] = State::Handle(new State,IINIT);
        if(itsArena[i]->createState(is, itsIndependent, *pool, *net))  ERROR_RECOVER;
    }
    if(!block5.close())  ERROR_RECOVER;

    if(!block1.close())  ERROR_RECOVER;
    return ok;
error_recover:
    ErrMessage err;
    err << error
        << " parsing optimizer seed failed. (code =  " << errPos
        << " at line " << streampos2lineNr(is,is.tellg())
        << ")"  << eom;
    return fail;
}
//////////////////////////////////////////////////////////////////////
//
void RandomOptimizerBASE::dumpConfig(ostream& os)
{
    enum{ ok = 0 , fail = 1 };
    {WriteBlock  block2(os,"CONFIG");
    if(block2){
    }
    }
    {WriteBlock  block3(os,"ACTIONCONFIG");
    if(block3){
    }
    }
}

void RandomOptimizerBASE::dumpState(void)
{
    enum{ ok = 0 , fail = 1 };
    debugTraceDumps();

    rebuildBestQ();
    for(int i=0; i<sequenceCount(); i++){
        rebuildQAt(i);
    }

    fstream os(itsDumpFileName, ios::out);
    if(!os){
        ErrMessage err;
       err << error<< "RandomOptimizerBASE::loadState: can't open config file \""
            << itsDumpFileName <<"\"";
        return; // fail;
    }
    {WriteBlock  block1(os,"RandomSequenceOptimizerARENA");
    if(block1){
        os << indent << "VERSION    \"1.03\"\n";
        os << indent << "DUMPTIME   \""<<now<<"\"\n";
        os << indent << "DUMPREASON \""<<exitReason()<<"\"\n";

        dumpConfig(os);
```

```
        {WriteBlock  block4(os,"BEST");
        if(block4){
            if(!isNull(itsBest)) itsBest->dumpState(os);
        }
        }

        {WriteBlock  block5(os,"ARENA");
        if(block5){
            os << indent << "LENGTH  " << sequenceCount() << endl;
            for(int i=0; i<sequenceCount(); i++){
                os << indent << "STATE  " << i+1 << endl;
                if(!isNull(itsArena[i])) itsArena[i]->dumpState(os);
            }
        }
        }
    }
    } // block
    os.close();
}
////////////////////////////////////////////////////////////////////
//
RandomOptimizerBASE& RandomOptimizerBASE::debug(const MyString& options){
    itsDebugOptions = options;

    itsDebugFlags[traceQ] = options.contains("TRACEQ")!=0;
    itsDebugFlags[traceEachQ] = options.contains("TRACEEACHQ")!=0;
    if(itsDebugFlags[traceQ] && itsDebugFlags[traceEachQ]){
        ErrMessage err;
        err << warn << "both options activated \"traceQ\" and \"traceEachQ\"\n"
            << "\"traceQ\" now disabled "<< eom;
        itsDebugFlags[traceQ] = false;
    }
    itsDebugFlags[tracePopulationStat] = options.contains("TRACEPOPULATIONSTAT")!=0;
    itsDebugFlags[tracePopulationDiff] = options.contains("TRACEPOPULATIONDIFF")!=0;
    if(options.contains("TRACEPOPULATION") &&
       !itsDebugFlags[tracePopulationStat] && !itsDebugFlags[tracePopulationDiff])
    {
        itsDebugFlags[tracePopulationStat] = true;
        itsDebugFlags[tracePopulationDiff] = true;
    }
    itsDebugFlags[traceActions] = options.contains("TRACEACTIONS")!=0;
    itsDebugFlags[traceActionTableChange]
        = options.contains("TRACEACTIONTABLECHANGE")!=0;
    itsDebugFlags[traceDumps] = options.contains("TRACEDUMPS")!=0;

    // trace params if any tracing is active
    itsDebugFlags[traceParams] = false;
    for(int i=0;i<countDebugOptions;i++){
        if(itsDebugFlags[i]){
            itsDebugFlags[traceParams] = true;
            break;
        }
    }
    return *this;
};
////////////////////////////////////////////////////////////////////
RandomOptimizerBASE&  RandomOptimizerBASE::debugStream(ostream& os){
    if(!isNull(itsDebugStream)) itsDebugStream->flush();
    itsDebugStream = &os; itsDebugContextWriter->setStream(&os);
    return *this;
};

////////////////////////////////////////////////////////////////////
//
RandomOptimizerBASE& RandomOptimizerBASE::debugTraceQ(void){
    if((itsIsImproved
        || (itsMaxTraceInterval > 0
            && itsMaxTraceInterval < (itsLoopCount - itsLastTraceCount))))
    {
        itsLastTraceCount = itsLoopCount;
        if(itsDebugFlags[traceQ]){
            double best=0.0, worst=itsArena[0]->quality(), mid=0.0;
            for(int i=0; i<sequenceCount(); i++){
                double currQ =itsArena[i]->quality();
                if(best<currQ) best  = currQ;
                if(worst>currQ)worst = currQ;
                mid += currQ;
            }
            mid /=sequenceCount();
            const char sep = ';';
            (debugStream()) << "TRACE_Q"<<sep<< itsLoopCount << sep
                            << itsLastQ << sep
                            << itsLastQ  << sep
                            << itsBest->quality() << sep
                            << itsBestASCount << sep
                            << itsBestLinkCount << sep;

            (debugStream()) << best << sep << (mid) << sep << worst << sep;
            (debugStream()) << endl;
        } else {
            debugTraceEachQ();
        }
    }
    return *this;
};
////////////////////////////////////////////////////////////////////
//
RandomOptimizerBASE& RandomOptimizerBASE::debugTraceEachQ(void){
    if(itsDebugFlags[traceEachQ]){
        const char sep = ';';
        (debugStream()) << "TRACE_EACHQ"<<sep<< itsLoopCount << sep;
        double bestQ=0.0, worstQ=itsArena[0]->quality(), midQ=0.0;
        for(int i=0; i<sequenceCount(); i++){
            double currQ =itsArena[i]->quality();
            if(bestQ<currQ) bestQ  = currQ;
            if(worstQ>currQ)worstQ = currQ;
            midQ += currQ;
        }
        midQ /=sequenceCount();

        (debugStream())
            << itsLastQ << sep
            << itsBest->quality() << sep
            << itsBestASCount << sep
            << itsBestLinkCount << sep;
        (debugStream())
            << bestQ << sep
            << midQ << sep
            << worstQ << sep
            << '*' << sep;
        for(int i=0; i<sequenceCount(); i++){
            (debugStream())  << itsArena[i]->quality()<< sep;
        }
        (debugStream()) << "\n";
    }
    return *this;
};

////////////////////////////////////////////////////////////////////
// gib alle Statistikwerte vor Beendigung des Optis aus
RandomOptimizerBASE& RandomOptimizerBASE::debugTraceFinalPopulationStat(void)
{
    // ensure that debugTracePopulationStat() will dump if it is enabled.
    itsPopulationLoop = 0;

    debugTracePopulationStat();
    return *this;
}

////////////////////////////////////////////////////////////////////
// Statistik der benutzten CX in der Population
RandomOptimizerBASE& RandomOptimizerBASE::debugTracePopulationUsedCX(void)
{
    // UNUSED FUNCTION !!!! ( dead code )
    if(itsPopulationDumpInterval < 0)  return *this;
    const int len = sequenceCount();

    itsAtomDist.fill(0);
    itsQDist.fill(0);

    ///////////////////////
    for(int i= 0; i< len; ++i){
        for(int n= -1; n< len; ++n){
            if( n < i ){
```

```
                StateDiff diff( stateAt( i ), stateAt( n ));

                diff.finalizePath();

                itsAtomDist [ i ][ n+1 ] = diff.atomDistance();
                itsQDist[ i ][ n+1 ]     = diff.quality();

                if( n != -1){
                    // spiegelung der Werte, beim Spiegeln
                    // uebernimmt i die Rolle von n, daher i+1
                    // Qualitaet * -1 , Schritte bleiben gleich.
                    itsAtomDist [ n ][ i+1 ] =  itsAtomDist [ i ][ n+1 ];
                    itsQDist[ n ][ i+1 ]     = -itsQDist[ i ][ n+1 ];
                }
            }
        }
    }
    return *this;
}

RandomOptimizerBASE& RandomOptimizerBASE::debugTracePopulationDiff(void)
{
    if(itsPopulationDumpInterval < 0)  return *this;
    if(!itsDebugFlags[tracePopulationDiff]) return *this;
    const int len = sequenceCount();

    itsAtomDist.fill(0);
    itsQDist.fill(0);

    ////////////////////
    for(int i= 0; i< len; ++i){
        for(int n= -1; n< len; ++n){
            if( n < i ){
                StateDiff diff(
                        stateAt( i ),
                        stateAt( n ));

                diff.finalizePath();

                itsAtomDist [ i ][ n+1 ] = diff.atomDistance();
                itsQDist[ i ][ n+1 ] = diff.quality();

                if( n != -1){
                    // spiegelung der Werte, beim Spiegeln
                    // uebernimmt i die Rolle von n, daher i+1
                    // Qualitaet * -1 , Schritte bleiben gleich.
                    itsAtomDist [ n ][ i+1 ] =  itsAtomDist [ i ][ n+1 ];
                    itsQDist[ n ][ i+1 ]     = -itsQDist[ i ][ n+1 ];
                }
            }
        }
    }

    const char sep = ';';
    for(int i= 0; i< sequenceCount(); ++i){ // y index
        (debugStream())
            << "TRACE_DIFF"<<sep<< itsLoopCount << sep
            << "ATOMS" << sep  << i+1 << sep;
        for(int n= -1; n< sequenceCount(); ++n){ // x index
            (debugStream()) << itsAtomDist [ i ][ n+1 ]<< sep;
        }
        (debugStream()) << endl;
    }

    (debugStream())
        << setiosflags(ios::fixed|ios::right|ios::showpoint)
        << setprecision(3);
    for(int i= 0; i< sequenceCount(); ++i){ // y index
        (debugStream())
            << "TRACE_DIFF"<<sep<< itsLoopCount << sep
            << "Q" << sep  << i+1 << sep;
        for(int n= -1; n< sequenceCount(); ++n){ // x index
            (debugStream()) << itsQDist [ i ][ n+1 ]<< sep;
        }
        (debugStream()) << endl;
    }
    return *this;
}
RandomOptimizerBASE& RandomOptimizerBASE::debugTracePopulationStat(void)
{
    if(itsPopulationDumpInterval < 0)  return *this;
    if(!itsDebugFlags[tracePopulationStat]) return *this;

    const int len = sequenceCount();

    ////////////////////
    // nur States mit Verbesserungen werden in die Statistik
    // aufgenommen.
    for(int i=0; i< len; ++i){
        if( stateAt( i )->quality() >stateAt( -1 )->quality()){
            StateDiff diff(
                    stateAt( -1 ),
                    stateAt( i ));

            diff.finalizePath();

            countChanges( diff, itsPerAS, itsPerCX, itsPerBase,rootName());
        }
    }

    if( (itsLoopCount - itsPopulationLoop) > itsPopulationDumpInterval )
    {
        const char sep = ';';
        (debugStream())
            << "TRACE_STAT"<<sep<< itsLoopCount << sep << "AS" << sep;
        for(int i=0; i<itsPerAS.count(); i++){
            (debugStream()) << itsPerAS[i] << sep;
        }
        (debugStream()) << endl;
        (debugStream())
            << "TRACE_STAT"<<sep<< itsLoopCount << sep << "CX" << sep;
        for(int i=0; i<itsPerCX.count(); i++){
            (debugStream()) << itsPerCX[i] << sep;
        }
        (debugStream()) << endl;
        (debugStream())
            << "TRACE_STAT"<<sep<< itsLoopCount << sep << "BASE" << sep;
        for(int i=0; i<itsPerBase.count(); i++){
            (debugStream()) << itsPerBase[i] << sep;
        }
        (debugStream()) << endl;

        itsPopulationLoop = itsLoopCount;
        itsPerAS.fill(0);
        itsPerCX.fill(0);
        itsPerBase.fill(0);
    }

    return *this;
};

//////////////////////////////////////////////////////////////////
//
RandomOptimizerBASE& RandomOptimizerBASE::debugParams(void){
    if(itsDebugFlags[traceParams]){
        (debugStream())
            << "OPTI_PARAMS;"
            << "  dumpfile="<< itsDumpFileName
            << "  nextDumpTime="<<itsNextDumpTime
            << "  dumpTimeInterval="<<itsDumpTimeInterval
            << "  endTime="<<itsEndTime
            << "  maxLoopCount="<<itsMaxLoopCount
            << "  maxIdleLoops="<<itsMaxIdleLoops
            << "  arenaSize="<<sequenceCount()
            << "  randomSeed="<<itsRndSeed
            << "  debugFlags=\""<<itsDebugOptions<<"\""
            << "  maxTraceInterval="<< itsMaxTraceInterval
            << endl;
        (debugStream())
            << "OPTI_DEFINES; "
            << "WITH_VETOPERIOD="
#ifdef     WITH_VETOPERIOD
            << "(defined="<<#WITH_VETOPERIOD<<")"
```

```
#else  //  WITH_VETOPERIOD
            << "(undef)"
#endif //  WITH_VETOPERIOD
            << " removeNotLocked="<<(itsCanRemoveNotLocked?"true":"false")
            << " removeNotLinked="<<(itsCanRemoveNotLinked?"true":"false")
            << " canFillEmpty="<<(itsCanFillEmpty?"true":"false")
            << " initialMinAssignedCxCountPct="<<initialMinAssignedCxCountPct
            << endl;
    }
    return *this;
}
////////////////////////////////////////////////////////////////////
//
RandomOptimizerBASE& RandomOptimizerBASE::debugTraceDumps(void)
{
    const char sep = ';';
    if(itsDebugFlags[traceDumps]){
        (debugStream())
            << "TRACE_DUMPS;"
            << now
            << sep << itsLoopCount
            << endl;
    }
    return *this;
}
#ifdef WITH_TIME_STAMPS
////////////7777/////////////////////////////////////////////////////
// Produziert eine Zeitmarke alle 1000 Loops
// Die Zeitmessung  ist relativ zur Startzeit in Sekunden
RandomOptimizerBASE& RandomOptimizerBASE::debugTraceLoop(void)
{
    const char sep = ';';
    static long   lastCount = 0L;
    static time_t timeStart = 0;
    // setze die startzeit neu wenn der loop zähler zurückgesetzt wurde.
    if(loopCount() < lastCount)  lastCount = 0;
    if(lastCount == 0 || lastCount+10<loopCount()){
        static time_t lastTime  = 0;
        time_t currtime = time(NULL);
        if(lastTime == currtime)
            return *this;
        if(lastCount == 0) timeStart=currtime;
        lastTime=time(NULL);
        (debugStream())
            << "TRACE_TIME"
            << sep << itsLoopCount
            << sep << (currtime-timeStart)
            << endl;
        lastCount = loopCount();
    }
    return *this;
}
#endif // WITH_TIME_STAMPS
////////////////////////////////////////////////////////////////////

#include "Array.cc"
template class Array<IMngObject<State> >;
```

N.7 Quelltext: RandomOpti_Calc.h

```
#ifndef   __RANDOM_OPTI_CALC_H__
#define   __RANDOM_OPTI_CALC_H__  1
//
//
// $Id: RandomOpti_Calc.h,v 1.7 2000/05/29 16:23:56 va Exp $
//
#include "macros.h"
#include "Matrix.h"
#include "long.Vec.h"
#include "Context.h"

class StateIndependent;
class State;
#include "LinkBuilder.h"
////////////////////////////////////////////////////////////
//: internal data structures of caching variants.
struct LinkCacheRec;
struct CollisionCacheRec;
class RandomOpti_Calc_noCache;
class RandomOpti_Calc_withCache;
class RandomOpti_Calc_withCache_AS;

#include "AVLMap.h"
////////////////////////////////////////////////////////////
struct LinkCacheRec
{
public:
    long              linkeCXID;
    long              rechteCXID;
    Context::Handle   linkH;
    double            q;

private:
    friend class RandomOpti_Calc_noCache;
    friend class RandomOpti_Calc_withCache;
    friend class RandomOpti_Calc_withCache_AS;

    ////////////////////////// INTERNAL DATA !
    LinkCacheRec * toEnd;
    LinkCacheRec * toFront;

public:
    friend inline bool operator==(const LinkCacheRec& , const LinkCacheRec&);
    friend inline bool operator!=(const LinkCacheRec& , const LinkCacheRec&);

    class  CompareIndex; friend class CompareIndex ;
    class CompareIndex: public CompOp< LinkCacheRec* >
    {
      public:
        typedef LinkCacheRec* key_t;
        virtual int operator()(const  key_t& x, const key_t& y) const
        {
            if( x->linkeCXID!=y->linkeCXID )
                return x->linkeCXID  - y->linkeCXID;
            if( x->rechteCXID!=y->rechteCXID )
                return x->rechteCXID - y->rechteCXID;
            return ( 0 );
        }
    };
    class  EqualIndex; friend class EqualIndex ;
    class  EqualIndex: public CompOp< LinkCacheRec* >
    {
    public:
        typedef LinkCacheRec* key_t;
        virtual int operator()(const  key_t& x, const key_t& y) const
        {
            return (x->linkeCXID==y->linkeCXID && x->rechteCXID==y->rechteCXID);
        }
    };
};

inline bool operator==(const LinkCacheRec& A , const LinkCacheRec& B)
{
    // die Indices reichen aus um einen record zu identifizieren.
    return ( A.linkeCXID  == B.linkeCXID &&
             A.rechteCXID == B.rechteCXID );
}
inline bool operator!=(const LinkCacheRec&A , const LinkCacheRec&B)
{
    return !(A==B);
}

struct CollisionCacheRec
{
    long              linkeCXID;
    long              rechteCXID;
    bool              val;
```

```
    // timing info
    long              lastUsed;
};

////////////////////////////////////////////////////////////////
//:
class RandomOpti_Calc_noCache : public RCObject
{
public:
    typedef RandomOpti_Calc_noCache Self;
    typedef IMngObject<Self>  Handle;

    ////////////
    static Handle make(const MyString& className);

    ////////////
    virtual ~RandomOpti_Calc_noCache(void);

    ////////////
    RandomOpti_Calc_noCache(void);

    ////////////
    //: resize and removes all previously cached information
    virtual void  clearCache(void);
    //: call this once before you use the cache.
    virtual void  setupCache(StateIndependent *indep);

    virtual void  calculateASQAt(State& s, int pos);
    virtual void  calculateLinkQAt(State& s, int pos);

    //: successfull/missed lookups in link cache
    inline double linkCacheHitRatio(void)const;

protected:
    Self& linkBuilder(const MyString& name);
    const MyString&    linkBuilder(void)const;

    //: calculate link q factor , part of setupCache
    void          setupLinkQ(void);
    inline double linkQFactorAt(int lPos)const;

    inline long&  cacheASUsage(int cx);
    inline long&  cacheASBadUse(int cx);
    inline long&  cacheASGoodUse(int cx);

    Context::Handle buildLink(State &S, int leftAS );

    //: handle the AS Collision Cache
    virtual int   lookupCollision(CollisionCacheRec& rec);
    virtual void  storeCollision( CollisionCacheRec& rec);
    //: handle the link CX Cache
    int           lookupLink(LinkCacheRec& rec);
    void          storeLink( LinkCacheRec& rec );

    ////////////////////
    //: independent data
    StateIndependent     *itsIndependent;

    //: defines how to build a link
    MyString             itsLinkBuilderName;
    LinkBuilder::Handle  itsLinkBuilder;

    //: default ascii context writer
    IMngPointer<ContextWriter>  itsDebugContextWriter;

    //: quality factor for link calculation
    //  len = linkCount
    doubleVec            itsLinkQFactor;

    ////////////////////
    //: maintain statistik for link cache lookups ?
    static const bool profileCacheHits = true;
    //: count of link cache lookups
    long             itsLinkCacheHits;
    //: count of successfull cache hits
    long             itsLinkCacheHitSuccess;
    //: count of failed lookups
    long             itsLinkCacheHitFail;
    //: count of recycled records
    long             itsLinkCacheHitDrop;
    //: count of dropped && recalculated records
    long             itsLinkCacheHitRecalc;
};

////////////////////////////////////////////////////////////////
//:
class RandomOpti_Calc_withCache : public RandomOpti_Calc_noCache
{
public:
    virtual ~RandomOpti_Calc_withCache(void);
    RandomOpti_Calc_withCache(void);

    //: resize and removes all previously cached information
    virtual void    clearCache(void);
    //: call this once before you use the cache.
    virtual void    setupCache(StateIndependent *indep);

    //: as q cache
    virtual void  calculateASQAt(State&   s, int pos);
    virtual void  calculateLinkQAt(State& s, int pos);

protected:
    inline long&    cacheASUsage(int   cx);
    inline long&    cacheASBadUse(int  cx);
    inline long&    cacheASGoodUse(int cx);

    //: caches the as type independent quality information
    //  c21w.hf will use ~ 3.5  MB.
    //  (6400 x 148 x (sizeof(float)==4)/1024/1024)
    inline float&  cacheASQ(int cx, int pos);
    //: call this once before you use the cache.
    virtual void    setupASQCache(void);
    //: sets up the as Q Cache
    virtual void    setupLinkQCache(void);

    //: is the link Q cache usable ?
    //  c21w.hf will use ~ 151 MB if you set this true.
    //  (6400 x 6400 x (sizeof(float)==4)/1024/1024)
    //  compile time flag
    static const bool  useLinkCache = true;
    //: lookup link quality,
    // returns -1    if not yet calculated ,
    //          0    if calculated and not linkable
    //         > 0  if calculated and a valid link
    inline float&      cacheLinkQ(int cx_left,int cx_right);
    //: handle the link CX Cache
    int       lookupLink(LinkCacheRec& rec);
    void      storeLink( LinkCacheRec& rec );
    //: how many additional link records should stay in link cache?
    //  additional records are all link records that do not refere
    //  to an currently used link.
    float             itsLinkCacheOverhead;
    //: max number of items in link cache.
    //  uses env var   LINK_CACHE_SIZE if set.
    int               itsLinkCacheMaxCount;

    // internal data of link cache
    // preconditions:
    //  *1) link cache much larger than ( count of states * protein )
    //  *2) active entries are regularly updated before they reach the end.
    //       (done in rebuildQ and every other time they are looked up)
    //: start of double linked list,
    //  entries with higher lastUsed are inserted at Front
    //  removable items are at itsEnd, ( or NULL if empty )
    LinkCacheRec * itsFront; // to youngest
    LinkCacheRec * itsEnd;   // to oldest (recycled in next insert)
    //: number of used items in itsLinkCache
    int            itsLinkCacheCurrCount;

    //
    Array<LinkCacheRec>                   itsLinkCache;
    AVLMap<LinkCacheRec*,LinkCacheRec*>   itsLinksByIndex;
    //: is the collision cache usable ?
    // c21w.hf will use ~ 37 MB if you set this true.
    // (6400 x 6400 x (sizeof(float)==1)/1024/1024)
    // only 18.8 MB with an upper triangular matrix!
    // (which is NOT the current implementation !)
```

```
    // requires ( cx1 < cx2 ), f(cx1,cx2) == f(cx2,cx1)
    static const bool  useCollisionCache = false;
    inline bool&       cacheASCollision(int cx1,int cx2);


    //: quality cache for AS ( len == nr of object in all-list)
    // (~ 6300x148x4bytes == 3.6 MB ! fuer c21w.hf)
    Matrix<float>                       itsASQCache;
    //: usage accounting for AS ( len == nr of object in all-list)
    longVec                             itsASQCacheUsage;
    //: usage accounting for AS ( len == nr of object in all-list)
    longVec                             itsASQCacheBadUse;
    //: usage accounting for AS ( len == nr of object in all-list)
    longVec                             itsASQCacheGoodUse;

    //: quality cache for Links ( len == nr of object in all-list)
    //  xlen == ylen ! ( ~ 6300x6300x4bytes == 151.4 MB ! fuer c21w.hf)
    Matrix<float>                      itsLinkQCache;

    //: collision cache  AS <-> AS
    //  CX's collide if if(m[lowerID][higherID] == true)
    //  148x148x1 ~ 21 KB
    Matrix<bool>                        itsASCollisionsCache;
};

//////////////////////////////////////////////////////////////
//:
class RandomOpti_Calc_withCache_AS : public RandomOpti_Calc_withCache
{
public:
    virtual ~RandomOpti_Calc_withCache_AS(void);

    //: resize and removes all previously cached information
    virtual void    clearCache(void);
    //: call this once before you use the cache.
    virtual void    setupCache(StateIndependent *indep);

    virtual void  calculateLinkQAt(State& s, int pos);
protected:
    //: sets up the as Q Cache
    virtual void    setupLinkQCache(void);
};

//////////////////////////////////////////////////////
//
inline const MyString&    RandomOpti_Calc_noCache::linkBuilder(void)const{
    return itsLinkBuilderName;
};
inline double RandomOpti_Calc_noCache::linkCacheHitRatio(void)const{
    return (itsLinkCacheHitFail>0L?
            ( double(itsLinkCacheHitSuccess) / double(itsLinkCacheHitFail) )
            :-1.0L);
};
//////////////////////////////////////////////////////
//
inline double        RandomOpti_Calc_noCache::linkQFactorAt(int lPos)const
{
    return itsLinkQFactor[lPos];
}
//////////////////////////////////////////////////////
//
inline float&             RandomOpti_Calc_withCache::cacheASQ(
        int cx,int pos)
{
    return  itsASQCache[cx][pos];
}
inline long&              RandomOpti_Calc_withCache::cacheASUsage(int i)
{
    return  itsASQCacheUsage[i];
}
inline long&              RandomOpti_Calc_withCache::cacheASBadUse(int i)
{
    return  itsASQCacheBadUse[i];
}
inline long&              RandomOpti_Calc_withCache::cacheASGoodUse(int i)
{
    return  itsASQCacheGoodUse[i];
}
inline bool&              RandomOpti_Calc_withCache::cacheASCollision(
        int i, int n){
    return  itsASCollisionsCache[i][n];
}
inline float&             RandomOpti_Calc_withCache::cacheLinkQ(
        int i,int n)
{
    return  itsLinkQCache[i][n];
}
#endif //__RANDOM_OPTI_CALC_H__
```

N.8 Quelltext: RandomOpti_Calc.cc

```
#include "RandomOpti_Calc.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizer.debug.h"

#include "AsciiContextReader.h"

//////////////////////////////////////////////////////////////
//static
RandomOpti_Calc_noCache::Handle RandomOpti_Calc_noCache::make(
        const MyString& className)
{
    if(className == "none"){
        return  RandomOpti_Calc_noCache::Handle(new RandomOpti_Calc_noCache,IINIT);
    } else if(className == "as_link"){
        return RandomOpti_Calc_noCache::Handle(new RandomOpti_Calc_withCache,IINIT);
    } else if(className == "as"){
        return RandomOpti_Calc_noCache::Handle(new RandomOpti_Calc_withCache_AS,IINIT);
    }
    ErrMessage err;
    err << fatal
        << "no  RandomOpti Calculator with name \""<< className<<"\""
        << eom;
    return  RandomOpti_Calc_noCache::Handle(new RandomOpti_Calc_noCache,IINIT);
}

//////////////////////////////////////////////////////////////
// static member variables
const bool   RandomOpti_Calc_noCache::profileCacheHits;

const bool   RandomOpti_Calc_withCache::useLinkCache;
const bool   RandomOpti_Calc_withCache::useCollisionCache;

//////////////////////////////////////////////////////////////
RandomOpti_Calc_noCache::~RandomOpti_Calc_noCache(void)
{
    /* EMPTY */
};
RandomOpti_Calc_withCache::~RandomOpti_Calc_withCache(void)
{
    if( useLinkCache ) {
        if ( profileCacheHits ){
            cerr <<"LINK_CACHE: size:"<<itsLinkCacheCurrCount
                 <<", lookups:"<<itsLinkCacheHits
                 <<", success:" <<itsLinkCacheHitSuccess
                 <<", fail:" << itsLinkCacheHitFail
                 <<", drop:" << itsLinkCacheHitDrop
                 <<", recalc:" << itsLinkCacheHitRecalc
                 <<", ratio:"<< linkCacheHitRatio()
                 << endl;
            long neverHit   = 0; // still -1
            long invalidHit = 0; // == 0
            for(int x=0; x<itsLinkQCache.xlength(); ++x){
                for(int y=0;y<itsLinkQCache.ylength(); ++y){
                    if( itsLinkQCache[x][y] == -1.0){
                        neverHit ++;
                    } else if( itsLinkQCache[x][y] == 0.0){
                        invalidHit ++;
                    }
                }
            }
```

```
            long all = (itsLinkQCache.ylength()*itsLinkQCache.xlength());

            cerr <<"AS_CACHE: combinations:"<<all
                 <<", never visited:"<< neverHit
                 <<", invalid combination:"<< invalidHit
                 <<", valid&used:" << ( all -  neverHit -  invalidHit )
                 << endl;
        }
    }
};
RandomOpti_Calc_withCache_AS::~RandomOpti_Calc_withCache_AS(void)
{
    /* EMPTY */
};
////////////////////////////////////////////////////////////
RandomOpti_Calc_noCache::RandomOpti_Calc_noCache(void)
:   itsDebugContextWriter( new AsciiContextWriter(&cerr), IINIT)
    ,itsLinkQFactor(0)
    ,itsLinkCacheHits( 0 )
    ,itsLinkCacheHitSuccess( 0 )
    ,itsLinkCacheHitFail( 0 )
    ,itsLinkCacheHitDrop(0)
    ,itsLinkCacheHitRecalc(0)
{
    linkBuilder("c21w.hf");
};

////////////////////////////////////////////////////////////////
static const LinkCacheRec::CompareIndex  linkCacheIndexComp;
static const LinkCacheRec::EqualIndex    linkCacheIndexEqual;

RandomOpti_Calc_withCache::RandomOpti_Calc_withCache(void)
:   itsLinkCacheOverhead(1.0)
    ,itsFront(NULL), itsEnd(NULL)
    ,itsLinkCacheCurrCount( 0 )
    ,itsLinkCache(0)
    ,itsLinksByIndex(NULL, &linkCacheIndexComp, &linkCacheIndexEqual)
{
    /* EMPTY */
};
////////////////////////////////////////////////
//
RandomOpti_Calc_noCache&   RandomOpti_Calc_noCache::linkBuilder(
        const MyString& name)
{
    if(name != itsLinkBuilderName){
        itsLinkBuilderName = name;
        itsLinkBuilder = LinkBuilder::make(name);
    }
    return *this;
};
////////////////////////////////////////////////////////////
void    RandomOpti_Calc_noCache::clearCache(void)
{
    /* EMPTY */
}
void    RandomOpti_Calc_withCache::clearCache(void){
    // passe die grössen an
    const ContextList &allCXList = *itsIndependent->allCXList();
    const Protein&    protein    = *itsIndependent->protein();

    itsLinkQFactor.resize(protein.length()-1);
    itsASQCache.resize(allCXList.length(),protein.length());
    itsASQCacheUsage.resize(allCXList.length());
    itsASQCacheBadUse.resize(allCXList.length());
    itsASQCacheGoodUse.resize(allCXList.length());
    if(useCollisionCache)
        itsASCollisionsCache.resize(allCXList.length(),allCXList.length());
    if(useLinkCache){
        itsLinkQCache.resize(allCXList.length(),allCXList.length());
        // if cache is already in use, deallocate all cached members!
        itsLinksByIndex.clear();
        if( itsLinkCacheCurrCount > 0 ){
            // release all cached links
            Context::Handle cxNull;
            for(LinkCacheRec* curr = itsEnd; curr!=NULL; curr=curr->toFront){
                curr->linkH = cxNull;
            }
            itsFront = NULL;
            itsEnd   = NULL;
            itsLinkCacheCurrCount = 0;
        }
        itsLinkCacheHits       = 0;
        itsLinkCacheHitSuccess = 0;
        itsLinkCacheHitFail    = 0;
        itsLinkCacheHitDrop    = 0;
        itsLinkCacheHitRecalc  = 0;
    }

    // und lösche alle Daten.
    itsASQCache.fill(-1.0);
    itsASQCacheUsage.fill(0);
    itsASQCacheBadUse.fill(0);
    itsASQCacheGoodUse.fill(0);
    if(useCollisionCache)
        itsASCollisionsCache.fill(false);
    if(useLinkCache)
        itsLinkQCache.fill(-1);

}
void    RandomOpti_Calc_withCache_AS::clearCache(void){
    // passe die grössen an
    const ContextList &allCXList = *itsIndependent->allCXList();
    const Protein&    protein    = *itsIndependent->protein();

    itsASQCache.resize(allCXList.length(),protein.length());
    itsASQCacheUsage.resize(allCXList.length());
    itsASQCacheBadUse.resize(allCXList.length());
    itsASQCacheGoodUse.resize(allCXList.length());
    if(useCollisionCache)
        itsASCollisionsCache.resize(allCXList.length(),allCXList.length());

    // und lösche alle Daten.
    itsASQCache.fill(-1.0);
    itsASQCacheUsage.fill(0);
    itsASQCacheBadUse.fill(0);
    itsASQCacheGoodUse.fill(0);
    if(useCollisionCache)
        itsASCollisionsCache.fill(false);
}
////////////////////////////////////////////////////////////
void  RandomOpti_Calc_noCache::setupLinkQ(void)
{
    const Protein&    protein   = *itsIndependent->protein();
    const doubleVec & asQFactor = itsIndependent->asQFactor();
    // calulate Link Q Factor
    itsLinkQFactor.resize(protein.length()-1);
    for(int i=0; i<protein.length()-1; ++i){
        itsLinkQFactor[i] = (asQFactor[i] + asQFactor[i+1])/2.0;
    }
}
////////////////////////////////////////////////////////////
void  RandomOpti_Calc_noCache::setupCache(StateIndependent *indep)
{
    itsIndependent = indep;
    // calulate Link Q Factor
    setupLinkQ();
}
void  RandomOpti_Calc_withCache::setupCache(StateIndependent *indep )
{
    itsIndependent = indep;
    clearCache();
    setupLinkQ();
    setupASQCache();
    setupLinkQCache();
}
void  RandomOpti_Calc_withCache_AS::setupCache(StateIndependent *indep)
{
    itsIndependent = indep;
    clearCache();
    setupLinkQ();
    setupASQCache();
}
////////////////////////////////////////////////////////////
```

```
void  RandomOpti_Calc_withCache::setupLinkQCache(void)
{
    //:größe des link caches
    //  nebenbedingungen:
    //  Die Anzahl der Records im Cache ist größer als die Anzahl
    //  der gleichzeitig berechneten Links!
    //  d.h    minimum = states * (as count + 1)
    //
    ///// c21w merged == 6500 as cx
    //      --> 6500 * 6500 == 42.250.000 links possible
    //
    //    20 (states) * 150 (protein)  == 3000
    //
    ///// CHEY (merged.allPeaks, net2)
    //
    // mögliche links 31.292.863    100.0%
    // berechnete        476.659      1.5%
    // nicht null        103.246      0.3%
    //
    // Abschätzung c21w.hf   0.3% == 195.000
    //
    const int setupCache  = 20;
    itsLinkCacheMaxCount  = int(
            itsIndependent->protein()->length()
            * setupCache  * (1.0+itsLinkCacheOverhead))+1;

    itsLinkCacheMaxCount = max(
            itsLinkCacheMaxCount,
            itsIndependent->allCXList()->length());
    itsLinkCacheMaxCount *= itsLinkCacheMaxCount;

    // LINK_CACHE_SIZE
    const char* p = getenv("LINK_CACHE_SIZE");
    int sz        = int(itsLinkCacheMaxCount * 0.003);
    if( p != NULL ){
        int t = atoi(p);
        if( t > sz ){
            sz=t;
        }
    }
    itsLinkCacheMaxCount = sz;

    itsLinkCacheCurrCount = 0;

    itsLinkCache.resize( itsLinkCacheMaxCount );
}
//////////////////
void  RandomOpti_Calc_withCache_AS::setupLinkQCache(void)
{
    /* EMPTY */
}
//////////////////////////////////////////////////////////////

void  RandomOpti_Calc_withCache::setupASQCache(void)
{
    const ResourceName rightTypePrefix
        = itsIndependent->getRightTypeNamePrefix();
    const ResourceName leftTypePrefix
        = itsIndependent->getLeftTypeNamePrefix();

    ResourceNameVec leftResName(_max_amino);
    ResourceNameVec rightResName(_max_amino);
    for(int i=0;i<_max_amino;i++){
        strstream namestream1;
        namestream1 << leftTypePrefix << name3char(i) <<ends;
        leftResName[i] =ResourceName(namestream1.str());namestream1.freeze(0);
        strstream namestream2;
        namestream2 << rightTypePrefix<< name3char(i) <<ends;
        rightResName[i]=ResourceName(namestream2.str());namestream2.freeze(0);
    }

    const ContextList &allCXList = *itsIndependent->allCXList();
    Protein&    protein          = *itsIndependent->protein();
    const doubleVec &asQFactor   = itsIndependent->asQFactor();

    doubleVec ltype(_max_amino);
    doubleVec rtype(_max_amino);
    const Resource *r;
    for(ContextList::Iterator iter=allCXList.first(); iter; ++iter){
        Context& cx = **iter;
        ltype.fill(0);
        rtype.fill(0);
        for(int i=0;i<_max_amino;i++){
            r = cx.Lookup(leftResName[i]);
            if(r==NULL || !r->isType(ltype[i])){
                if(r != NULL){
                    ERRMESSAGE(err);
                    err << error << "wrong type of resource in context\n"
                        << "name=\""<<stream << leftResName[i]<<"\"" ;
                    err << eom;
                }
                ltype[i] = Fuzzy::False;
            }
            r= cx.Lookup(rightResName[i]);
            if(r==NULL || !r->isType(rtype[i])){
                if(r != NULL){
                    ERRMESSAGE(err);
                    err << error << "wrong type of resource in context\n"
                        << "name=\""<<stream << rightResName[i] <<"\"" ;
                    err << eom;
                }
                rtype[i] = Fuzzy::False;
            }
        }
        double prevStructQ = 0.0;
        for(int pos = 0; pos<protein.length(); pos++){
            double    currQ = 0.0;
            if(pos == 0 ||
               itsIndependent->asExpressionAt(pos-1)!=itsIndependent->asExpressionAt(pos))
            {
                prevStructQ =
   convert_cast<Fuzzy::zahl_t>(itsIndependent->asExpressionAt(pos)->eval(cx));
            }
            currQ +=  prevStructQ;
            if(pos>0)
                currQ +=  ltype[protein[pos-1].aa()];
            currQ +=  rtype[protein[pos].aa()];
            cacheASQ(cx.id(),pos) = currQ * asQFactor[pos];
        }
    }
}


//////////////////////////////////////////////////////
//:
void RandomOpti_Calc_withCache::calculateASQAt(State& S, int pos)
{
    S.asQAt(pos) =  cacheASQ( S.asAt(pos)->id(), pos );
}
void RandomOpti_Calc_noCache::calculateASQAt(State& S, int pos)
{
    S.buildASTypeQAt(pos); // changeASAt invalidated it
    S.asQAt(pos) =  itsIndependent->asQFactorAt(pos) * (
            convert_cast<Fuzzy::zahl_t>(S.asExpressionAt(pos)->eval(*S.asAt(pos)))
            + convert_cast<Fuzzy::zahl_t>(S.asTypeQAt(pos)));
}
//////////////////////////////////////////////////////////////
//:
Context::Handle RandomOpti_Calc_noCache::buildLink(State &S, int leftAS )
{
    // State &S = *stateAt(seqIndex);
    Context::Handle linkH;
    ASSERT( leftAS >= 0 && leftAS < S.asCount()-1);

    { ActiveAccount active(S.account());

    S.clearLink(leftAS,true);
    if(isNull(S.asAt(leftAS)) || isNull(S.asAt(leftAS+1)))
        return linkH;
    linkH = Context::Handle(new Context(itsIndependent->globalCx(),
                                        Context::Containing),IINIT);
    linkH = itsLinkBuilder->buildLink(S.asAt(leftAS),S.asAt(leftAS+1),
                                      itsIndependent->globalCx());
```

```
    if( linkH->length(Context::local) == 0 )
        return Context::Handle();
#ifdef BUILD_LINK_SHOW_LINKS
    itsDebugContextWriter->setStream(&cerr);
    cerr << "calc LINK from  AS index("<<leftAS << ") to AS index("<<leftAS+1<<")\n";
    cerr << "##################Context for AS("<<leftAS << ") is:\n";
    itsDebugContextWriter->write(*S.asAt(leftAS));
    cerr << "##################Context for AS("<<leftAS+1 << ") is:\n";
    itsDebugContextWriter->write(*S.asAt(leftAS+1));
    cerr << "##################First Context Link is:\n";
    itsDebugContextWriter->write(*linkH);
#endif // BUILD_LINK_SHOW_LINKS
    // muss das Muster die Peaks erwerben koennen?
    // testPattern == JA, erwirbt HNCO beider Seiten. verhindert link zu sich selbst
    const bool patternNeedsUnlockedAS = true;
    bool leftLocked = false;
    bool rightLocked = false;
    if(patternNeedsUnlockedAS){
        leftLocked = S.asLockAt(leftAS).isAquired();
        rightLocked = S.asLockAt(leftAS+1).isAquired();
        // temporary unlock left and right AS, pattern wants to lock them
        if(leftLocked) S.asLockAt(leftAS).release(false);
        if(rightLocked) S.asLockAt(leftAS+1).release(false);
    }
    {
        // test this Context
        ASSERT(!isNull(S.linkPatternAt(leftAS)));
        ConditionState::Handle HLS = S.linkPatternAt(leftAS)->createState();

        ASSERT(!isNull(HLS));
        // reourcen sind schon gebunden.
        S.linkPatternAt(leftAS)->BindResources(*linkH,*HLS,Condition::findNext);

        if(HLS->isExhausted()){
            //linkH             = Context::Handle();
            S.linkAt(leftAS) = Context::Handle();
        } else {
            S.linkAt(leftAS) = linkH;
        }

#ifdef BUILD_LINK_SHOW_LINKS
        cerr << "--- link is "<<( S.linkAt(leftAS)!=NULL?"not null":"NULL")<<endl;
        cerr << ( S.linkAt(leftAS)!=NULL?"AKZEPTIERT":"nicht AKZEPTIERT")<<endl;
#endif // BUILD_LINK_SHOW_LINKS

        // teste ob das Muster nur einen Zustand zurueckgibt.
        // testCount: restrict the number of tests
        // -1 == test forever
        static int testCount = 0;
        if((testCount > 0 || testCount == -1) && !HLS->isExhausted()){
            testCount --;
            Context firstResult(*linkH); // makes a backup
            while(!HLS->isExhausted()){
                S.linkPatternAt(leftAS)->BindResources(*linkH,*HLS,Condition::findNext);
                // pattern can return more then one state
                // ok as long as all state are the same
                if(stableComp(firstResult,*linkH)!=0){
                    // pattern is wrong.
                    // it returns (calculates) more then one state ,
                    // therefore we need a different algorithm. bad .
                    // document it and die.
                    ErrMessage err;
                    err << fatal << "While building a unique Link from AS("<<leftAS+1
                        << ") to AS("<<  leftAS+2 <<") the link pattern returns "
                        "(calculates) more then one state. This is illegal.\n";
                    itsDebugContextWriter->setStream(&err.stream());
                    err << "##################Context for AS("<<leftAS << ") is:\n";
                    itsDebugContextWriter->write(*S.asAt(leftAS));
                    err << "##################Context for AS("<<leftAS+1 << ") is:\n";
                    itsDebugContextWriter->write(*S.asAt(leftAS+1));

                    err << "##################First Context Link is:\n";
                    itsDebugContextWriter->write(firstResult);
                    err << "##################Second Context Link is:\n";
                    itsDebugContextWriter->write(*linkH);

                    err << "##################Link Pattern is:\n";
                    S.linkPatternAt(leftAS)->print(err.stream());
                }
            }
            *linkH = firstResult; // restore old value
        }
        HLS = ConditionState::Handle();  // destroy State, releases all resources
    }
    if(patternNeedsUnlockedAS){
        // re-aquire AS
        if(leftLocked)  S.asLockAt(leftAS).aquire();
        if(rightLocked) S.asLockAt(leftAS+1).aquire();
    }
    } ////////// active Account

    linkH = S.linkAt(leftAS);

    return linkH;
};
//////////////////////////////////////////////////////////////
//: is this collision in cache ?
//
//// return:
// 0     no,
// 1     yes
int RandomOpti_Calc_noCache::lookupLink(LinkCacheRec& UNUSED(rec))
{
    return 0; // not found ( no cache !)
}
void RandomOpti_Calc_noCache::storeLink( LinkCacheRec& UNUSED(rec))
{
    /* EMPTY */
}
//////////////////////////////////////////////////////////////
//: is this Link in cache ?
//
//// return:
// 0     no,            build link and calculate quality.
// 1     yes, partially, copy quality from record, build link if q > 0
// 2     yes, complete,  copy quality and link from record
int     RandomOpti_Calc_withCache::lookupLink(LinkCacheRec& rec)
{
    if( !useLinkCache )
        return 0;
    if( profileCacheHits ){
        itsLinkCacheHits++;
        if(itsLinkCacheHits>=MAXINT-1){
            const long f = MAXINT/100;
            itsLinkCacheHits       /= f;
            itsLinkCacheHitSuccess /= f;
            itsLinkCacheHitFail    /= f;
            itsLinkCacheHitDrop    /= f;
            itsLinkCacheHitRecalc  /= f;
        };
    }
    // try to find the quality for this record.
    rec.q = itsLinkQCache[rec.linkeCXID][rec.rechteCXID];
    if(rec.q==0){
        // this combination was already calculated
        // and will not give a valid link cx! (and is therefore not cached)
        // simulate a full record.
        rec.linkH = Context::Handle(); // assign NULL
        return 2; // simulate a full record.
    } else if( rec.q > 0.0 ){
        if( ! itsLinksByIndex.contains(&rec) ){
            if( profileCacheHits ) itsLinkCacheHitRecalc++;
            if( profileCacheHits ) itsLinkCacheHitFail++;
            return 1; // link was calculated, but is no longer in cache
        }
        if( profileCacheHits ) itsLinkCacheHitSuccess ++;
        rec.linkH = itsLinksByIndex[&rec]->linkH;
        return 2;// q found in cache, and link cx is available.
    }
    if( profileCacheHits ) itsLinkCacheHitFail++;
    return 0; // nothing is known about this index combination.
}

void    RandomOpti_Calc_withCache::storeLink( LinkCacheRec& rec )
{
    if( !useLinkCache )
```

```
        return;

    // update link quality cache
    cacheLinkQ(rec.linkeCXID, rec.rechteCXID)=rec.q;

    if( rec.q <= 0.0 )
        return; // don't store unlinkable cx in link-cx-cache !

    LinkCacheRec* curr;
    LinkCacheRec* y = itsFront; // the youngest or NULL
    if( itsFront != NULL ) {
        if( itsLinksByIndex.contains( &rec )){
            curr = itsLinksByIndex[&rec];
            // already in cache, relink it to front!

            // is curr already in front ? then skip relinking.
            if( curr != y ){
                // is curr the last object in list? update header info
                if( curr == itsEnd )
                    itsEnd = curr->toFront;
                // aus der liste heraus nehmen
                if( curr->toEnd   != NULL )
                    curr->toFront->toEnd = curr->toEnd;
                if( curr->toFront != NULL )
                    curr->toEnd->toFront = curr->toFront;

                // neu an der spitze einfügen.
                curr->toFront = NULL;
                curr->toEnd   = itsFront;

                // link the previous front
                y->toFront    = curr;

                // update header info
                itsFront      = curr;
            }
            return ;
        }

        // this rec is not in cache, create a new entry!
        if (itsLinkCacheCurrCount < itsLinkCacheMaxCount) {
            // there is still space left in cache, grow cache.

            // get the entry  and copy it
            curr = &itsLinkCache[itsLinkCacheCurrCount++];
            *curr = rec;

            curr->toFront = NULL;
            curr->toEnd   = itsFront;

            y->toFront    = curr;

            itsFront      = curr;
        } else {
            if( profileCacheHits ) itsLinkCacheHitDrop++;
            // ein record muss recycled werden.
            // record aufräumen und aus dem index cache löschen.
            LinkCacheRec* o = itsEnd;   // the oldest, recycle target
            itsLinksByIndex.del(o);
            // unlink old record from end, terminate list at second oldest.
            if(o->toFront!=NULL)
                o->toFront->toEnd = o->toEnd;

            // get the entry and copy it
            curr = o;
            *curr = rec;

            curr->toFront = NULL;
            curr->toEnd   = itsFront;

            y->toFront    = curr;

            itsFront = curr;
        }
    } else {
        // initialize list header and first entry
        // called once, for the first element

        // get the entry and copy it
        curr = &itsLinkCache[itsLinkCacheCurrCount++];
        *curr = rec;

        curr->toEnd    = NULL;
        curr->toFront  = NULL;

        itsFront = curr;
        itsEnd   = curr;
    }
    itsLinksByIndex[curr] = curr;

}
////////////////////////////////////////////////////////////
int   RandomOpti_Calc_noCache::lookupCollision(CollisionCacheRec& UNUSED(rec))
{
    return 0; // not found
}
void    RandomOpti_Calc_noCache::storeCollision( CollisionCacheRec& UNUSED(rec))
{
    /* EMPTY */
};
////////////////////////////////////////////////////////////
//:
void    RandomOpti_Calc_noCache::calculateLinkQAt(State& S, int linkIndex)
{
    // q below a value which a link is concidered to be invalid
    // even if it builds as a non-empty cx
    const float epsilon = 1e-4;

    LinkCacheRec rec;

    rec.linkeCXID  = S.asAt(linkIndex)->id();
    rec.rechteCXID = S.asAt(linkIndex+1)->id();
    // ist der Link schon im cache ?
    int found = lookupLink(rec);

    if(found == 2) {
        // rec.q == 0 is handled here !
        ASSERT( rec.q != 0.0  && !isNull(rec.linkH) );
        ASSERT( rec.q == 0.0  && isNull(rec.linkH)  );
        // nothing to do
    } else { // (found == 1), (found == 0)
        ASSERT( found == 0 || (found == 1 && rec.q != 0.0 ));
        rec.linkH = buildLink(S, linkIndex);
        // S.linkAt(linkIndex)  = rec.linkH; // already done in buildLink!
        // case found == 1:
        //    q > 0  ==> a valid cx must exist!
        //    ( valid == not NULL with at least one item)
        if( found == 0 ){
            // calculate an link for the first time
            if(isNull(rec.linkH) || rec.linkH->length(Context::local)==0){
                // empty CX or NULL, ==> q = 0
                rec.q     = 0.0;
            } else {
                rec.q =
  convert_cast<Fuzzy::zahl_t>(S.linkExpressionAt(linkIndex)->eval(*S.linkAt(linkIndex)));
                if(SHOW_Q_TRACE) cout <<" exprQ=" << rec.q;
                if(rec.q > epsilon)
                    rec.q = 4.0 * rec.q + Fuzzy::True;
                else // q very small,
                    rec.q = 0.0;
            }
        }
        if ( rec.q == 0.0 ){
            rec.linkH = Context::Handle(); // assign NULL
        }
        storeLink( rec );
    }

    S.linkQAt(linkIndex) = rec.q * linkQFactorAt(linkIndex) ;
    S.linkAt(linkIndex)  = rec.linkH;

    if( rec.q > 0 ){
#ifdef DEBUG_CALCULATION_Q
        cout << "calculateLinkQ("<<seqIndex<<") add link "
```

```
            << linkIndex << " += " <<  S.linkQAt(linkIndex) << endl;
#endif
        S.validLink().set(linkIndex,true);
        // update Q
        S.quality() += S.linkQAt(linkIndex);
        if(SHOW_Q_TRACE) cout <<"  add Q " ;
    }
}
////////////////////////////////////////////////////////////
//:
void    RandomOpti_Calc_withCache::calculateLinkQAt(State& S, int linkIndex)
{
    // epsilon: quality threshold below which a link is concidered to be invalid
    // even if it builds as a non-empty cx
    const float epsilon = 1e-4;

    LinkCacheRec rec;

    rec.linkeCXID  = S.asAt(linkIndex)->id();
    rec.rechteCXID = S.asAt(linkIndex+1)->id();
    // ist der Link schon im cache ?
    int found = lookupLink(rec);

    if(found == 2) {
        // rec.q == 0 is handled here !
        ASSERT( (rec.q != 0.0  && !isNull(rec.linkH)) || (rec.q == 0.0  && isNull(rec.linkH)  ));
        // nothing to do
    } else { // (found == 1), (found == 0)
        ASSERT( found == 0 || (found == 1 && rec.q != 0.0 ));
        rec.linkH = buildLink(S, linkIndex);
        // S.linkAt(linkIndex)  = rec.linkH; // already done in buildLink!
        // case found == 1:
        //    q > 0  ==> a valid cx must exist!
        //    ( valid == not NULL with at least one item)
        if( found == 0 ){
            // calculate an link for the first time
            if(isNull(rec.linkH) || rec.linkH->length(Context::local)==0){
                // empty CX or NULL, ==> q = 0
                rec.q     = 0.0;
            } else {
                rec.q =
   convert_cast<Fuzzy::zahl_t>(S.linkExpressionAt(linkIndex)->eval(*S.linkAt(linkIndex)));
                if(SHOW_Q_TRACE) cout <<" exprQ=" << rec.q;
                if(rec.q > epsilon)
                    rec.q = 4.0 * rec.q + Fuzzy::True;
                else // q very small,
                    rec.q = 0.0;
            }
        }
        if ( rec.q == 0.0 ){
            rec.linkH = Context::Handle(); // assign NULL
        }
        storeLink( rec );
    }

    S.linkQAt(linkIndex) = rec.q * linkQFactorAt(linkIndex);
    S.linkAt(linkIndex)  = rec.linkH;

    if( rec.q > 0 ){
#ifdef DEBUG_CALCULATION_Q
        cout << "calculateLinkQ("<<seqIndex<<") add link "
             << linkIndex << " += " <<  S.linkQAt(linkIndex) << endl;
#endif
        S.validLink().set(linkIndex,true);
        // update Q
        S.quality() += S.linkQAt(linkIndex);
        if(SHOW_Q_TRACE) cout <<"  add Q " ;
    }
}
////////////////////////////////////////////////////////////
void    RandomOpti_Calc_withCache_AS::calculateLinkQAt(State& S, int linkIndex)
{
    RandomOpti_Calc_noCache::calculateLinkQAt( S, linkIndex );
}
////////////////////////////////////////////////////////////
#include "CompOp.cc"
#include "Map.cc"
#include "AVLMap.cc"
AVLMAP_INSTANCE( LinkCacheRec* ,LinkCacheRec* );
AVLMAP_COMPOP_INSTANCES(LinkCacheRec* );

#include "Array.cc"
template class Array<LinkCacheRec>;
```

N.9 Quelltext: RandomOpti_StateIndependent.h

```
#ifndef   __RANDOM_OPTI_STATE_INDEPENDENT_H__
#define   __RANDOM_OPTI_STATE_INDEPENDENT_H__  1
//
//
// $Id: RandomOpti_StateIndependent.h,v 1.15 2000/06/12 12:08:51 va Exp $
//
//
//
//#define DONT_TRACE 1

#include "TraceObj.h"
#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "Cache.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "Protein.h"
#include "PeakPool.h"
#include "PeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "Expression.h"
#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "ParseCondition.h"

#include "RandomOpti_Calc.h"
#define  WITH_VETOPERIOD  1
#undef   WITH_VETOPERIOD

////////////////////////////////////////
// :suche in Bereich [0.. x.count()] nach dem Element == key
//////
// return:
//       gefunden:  index des Elements
// nicht gefunden:     -1
int binsearch(const intVec &x, int key);

// suche in Bereich [0..len] nach dem Element == key
//////
// return:
//       gefunden:  index des Elements
// nicht gefunden:     -1
int binsearchInRange(const intVec &x, int key, int len);

////////////////////////////////////////
class StateIndependent;
class State;
class RandomOptimizer;
class RandomOpti_Calc_noCache;
////////////////////////////////////////////////////////////
class StateIndependent : public RCObject
{
```

```
public:
    typedef IMngObject< StateIndependent > Handle;
    ~StateIndependent(void);
    StateIndependent(void);

    StateIndependent& globalCx(Context::Handle & g);
    StateIndependent& protein( Protein::Handle& p);
    StateIndependent& asListAt(int seqIndex, ContextList::Handle& p);
    StateIndependent& asExpressionAt(int seqIndex, Expression<Fuzzy>::Handle& p);
    StateIndependent& linkExpressionAt(int seqIndex, Expression<Fuzzy>::Handle& p);
    StateIndependent& linkPatternAt(int seqIndex, Condition::Handle& p);
    StateIndependent& endTime( time_t p);
    StateIndependent& dumpTimeInterval(time_t p);
    StateIndependent& maxLoop( long p);

    StateIndependent&    allCXListName(const MyString & name);
    const MyString&      allCXListName(void)const;
    ContextList::Handle  allCXList(void)const;

    Context::Handle      globalCx(void)const;
    Protein::Handle      protein( void)const;
    ContextList::Handle  asListAt(int seqIndex)const;
    Expression<Fuzzy>::Handle asExpressionAt(int seqIndex)const;
    Expression<Fuzzy>::Handle linkExpressionAt(int seqIndex)const;
    Condition::Handle    linkPatternAt(int seqIndex)const;
    const ResourceName&  basePeakResName(void)const;
    const ResourceName&  getLeftResName(int seqIndex)const;
    const ResourceName&  getRightResName(int seqIndex)const;
    const DLList<Peak::Handle>&  baseList(void)const;
    int                  extractBaseID(const Context::Handle & cx)const;
    const intVec&        asForPeak(int id);
    const intVec&        peakAtAS(int id);

    inline double&       setASQFactorAt(int lPos);
    inline double        asQFactorAt(int lPos)const;
    inline const doubleVec&  asQFactor(void)const;

    RandomOpti_Calc_noCache::Handle  calculator(void);

    StateIndependent&    calculatorName(const MyString& name);
    const MyString&      calculatorName(void)const;

    StateIndependent&    getLeftTypeNamePrefix(const MyString& prefix);
    const MyString&      getLeftTypeNamePrefix(void)const;
    StateIndependent&    setRightTypeNamePrefix(const MyString& prefix);
    const MyString&      getRightTypeNamePrefix(void)const;

    static StateIndependent::Handle loadState(
            const MyString& name,
            PeakPool::Handle poolH,
            PeakNet::Handle net,
            Context::Handle& globalContext);

protected:
    void resize(int l);

    StateIndependent&  buildResName(void);

    void               setupAS2PeakMapping(void);
    virtual void       setupCXIDs(void);
    virtual void       setupCache(void);

    int                loadAllCXList(
            PeakPool::Handle poolH,
            PeakNet::Handle PN);

    //: das Protein zu dem die Sequenzen gehoeren (used in dump)
    MyString                          itsProteinName;

    //: das Protein zu dem die Sequenzen gehoeren
    Protein::Handle                   itsProtein;

    //: the "all" List if it is defined
    ContextList::Handle               itsAllCXList;
    MyString                          itsAllCXListName;

    //: Liste der moeglichen As Contexte je AS
    //  length==  protein.length()
    Array< ContextList::Handle    >       itsASList;

    //: Expressions for evaluation of AS Contexts
    //  length==  protein.length()
    Array< Expression<Fuzzy>::Handle    >  itsASExpr;

    //: Expressions for evaluation of Link Contexts
    //  length==  protein.length()-1
    Array< Expression<Fuzzy>::Handle    >  itsLinkExpr;

    //: ConditionList for Creation of Link Contexts
    //  length ==  protein.length()-1
    Array< Condition::Handle            >  itsLinkCond;

    //: AminoAcid dependend resource name
    //  of the acid type eg: "GRIE #1 Ala"
    Array< ResourceName                 >  itsLeftResName;
    Array< ResourceName                 >  itsRightResName;

    //: as q factor
    doubleVec                             itsASQFactor;

    ////////////////////////////////////////
    //: Name des Basispeaks
    ResourceName                          itsBasePeakResName;

    //: Name der BasisListe
    SharedString                          itsBaseListName;

    //: die Basisliste
    DLList<Peak::Handle>                  itsBaseList;

    //: ID's der Basispeaks je Aminosaeure
    // possible Peaks for AS(index+1) = itsPeaksForAS[index]
    // index    == asID -1
    // contents == peakIndex
    Array< intVec >                       itsPeaksForAS;

    //: Nummmer der Aminosaeure je Basispeak
    // possible AS for Peak(index+1) = itsASForPeak[index]
    // index    == peakID-1
    // contents == asIndex
    Array< intVec >                       itsASForPeak;

    //: flag, setting of cx ids, initially false.
    bool                                  itsCXIDsAreSetup;

    //: prefix for AS type names "GRIE #1" | "GRIE #2"
    // left == i-1 , right == i
    MyString                              itsLeftTypeNamePrefix;
    MyString                              itsRightTypeNamePrefix;
    Context::Handle                       itsGlobalCx;

    //: calculation & caching
    MyString                              itsCalculatorName;
    RandomOpti_Calc_noCache::Handle       itsCalc;
};

////////////////////////////////////////////////////////////////
class State : public RCObject
{
public:
    typedef IMngObject< State > Handle;

    State(void);
    ~State(void);

    State& copyFrom(const State& src);

    inline int asCount(void)const;
    inline int linkCount(void)const;

    State& clearState(void);
    void clearAS( int asIndex, bool bookKeeping = true);
    void clearLink( int linkIndex, bool bookKeeping = true);
```

```
    inline Protein::Handle protein(void)const;
    inline ContextList::Handle       asListAt(int seqIndex)const;
    inline Expression<Fuzzy>::Handle asExpressionAt(int seqIndex)const;
    inline Expression<Fuzzy>::Handle linkExpressionAt(int seqIndex)const;
    inline Condition::Handle linkPatternAt(int seqIndex)const;

    State& useIndependent(StateIndependent::Handle p );
    inline StateIndependent::Handle usedIndependent( void)const;

    //: get the Account Object.
    inline Account&          account(void);
    //: overall quality,
    inline double&           quality(void);
    inline double            quality(void)const;
    //: count of assigned AS
    inline int&              usedCount(void);
    inline int               usedCount(void)const;
    //: nr of Links
    inline int               usedLinks(void)const;

    inline const Context::Handle& asAt(int i)const;
    inline void              putASAt(int i, const Context::Handle& cx);
    inline double&           asQAt(int i);
    inline double            asQAt(int i)const;
    inline double&           asTypeQAt(int i);
    inline double            asTypeQAt(int i)const;

    void                     buildASTypeQAt(int i);
    double                   extractASTypeQAt(int i, const ResourceName &name);

    inline ContextLock&      asLockAt(int i);
    inline StatArray<bool>& validAS(void);
    inline StatArray<bool>& changedAS(void);
    inline MyString          asFlagsAt(int i)const;
    inline int               basePeakIDAt(int i)const;

    inline Context::Handle& linkAt(int i);
    inline double&           linkQAt(int i);
    inline StatArray<bool>& validLink(void);
    inline MyString          linkFlagsAt(int i)const;


    inline long  loopCountAt(int i)const;
    inline long& loopCountAt(int i);
    inline bool  canChangeAt(int i)const;

    inline State& nextEpoch(void);
    inline State& setEpoch(long e);

    inline int getTTL(void)const;
    inline State& setTTL(int ttl);

    //: insert another Change
    inline State& pushChange( int pos );
    //: remove toplevel change ( does not reset itsChangedAS )
    inline int    popChange( void );
    //: how many changes are there?
    inline int    changeCount(void) const ;
    //: show position of the N th change
    inline int    changeAt(int i)const;
    // position of the top level change
    inline int    topChange(void)const;
    //: drop changes at POS (include itsChangedAS)
    inline State& declareUnchanged(int pos);
    //: drop all changes (include itsChangedAS)
    inline State& clearChanges(void);

    void          baseIDsInRange(intVec& baseID, int firstPos, int len);
    int           countLinksInRange( int firstPos, int len );

    bool createState(istream& is, StateIndependent::Handle& ind,
                     const PeakPool& Pool, const PeakNet& Net
                     );
    void dumpState(ostream& os);

    bool OK(void)const;    // state invariant
```



```
    /// STATICS
    //: how many loops should an already altered AS stay unchanged
    static long  vetoPeriod(void);
    static void  vetoPeriod(long v);
private:
    ////////////////////////////////////////////////////////
    //: die unveraenderlichen Variablen
    StateIndependent::Handle              itsIndependent;

    ////////////////////////////////////////////////////////
    //: Account -- die Account gruppe.
    Account                               itsAccount;

    ////////////////////////////////////////////////////////
    //: time to live  [ 0 ... ( min start = 5 .. 50) ]
    int                                   itsTTL;

    ////////////////////////////////////////////////////////
    //: overall quality
    double                                itsQ;

    ////////////////////////////////////////////////////////
    //:stack of changed AS positions  (len == protein.count())
    intVec                                itsChangedASStack;

    //: count of used elements in itsChangedASStack
    int                                   itsChangedASCount;

    ////////////////////////////////////////////////////////
    //: nr of assigned positions
    int                                   itsAssignedCount;

    ////////////////////////////////////////////////////////
    //: zugewiesener AS Context je AS, oder NULL
    //  length==  protein.length()
    Array< Context::Handle    >           itsAS;
    //: lock object for the assigned AS
    Array< ContextLock        >           itsASLock;
    //: Quality of AS Contexts
    doubleAVec                            itsASQ;
    //: Quality of type match
    doubleAVec                            itsTypeQ;
    //: AS wurde veraendert und noch nicht neubewertet.
    //  gesetzt wenn AS(pos) neu zugewiesen.
    StatArray<bool>                       itsChangedAS; //(itsProtein.count());
    //: gueltige AS, (pos ist belegt, erwerbbar, bewertbar Q>0)
    StatArray<bool>                       itsValidAS; // (itsProtein.count());

    //: zugewiesener Link Context je AS, oder NULL
    //  length==  protein.length()-1
    //  der Cx fuer AS(i)->AS(i+1) ist in itsLink[i]
    Array< Context::Handle    >           itsLink;
    //: Quality of Link Contexts
    //  length==  protein.length()-1
    doubleAVec                            itsLinkQ;
    //: gueltiger Link, (pos ist belegt (AS(pos) und AS(pos+1) sind guelting (s.o.)),
    //                  , bewertbar Q>0)
    StatArray<bool>                       itsValidLink;//(itsProtein.count()-1);

    ////////////////////////////////////////////////////////
    //: loop count, independent from opti loop count
    //  this is internal data !! do not publish!!
    long                                  itsCount;

    ////////////////////////////////////////////////////////
    //: loop count of the last successfull change.
    //  internal data, synchronous with itsCount!
    longVec                               itsLoopCount;
    //: how long is a subsequent change of an already changed AS forbidden.
    static long                           theVetoPeriod;

    bool asOK(void)const;
    bool linkOK(void)const;
};

//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
inline StateIndependent& StateIndependent::protein(
```

```
        Protein::Handle& p)
{
    itsProtein =p;
    resize(p->length());
    buildResName();

    return *this;
};
inline StateIndependent& StateIndependent::allCXListName(
        const MyString& name)
{
    itsAllCXListName=name;
    return *this;
};
inline StateIndependent& StateIndependent::asListAt(
        int seqIndex, ContextList::Handle& p)
{
    itsASList[seqIndex]=p;
    return *this;
};
inline StateIndependent& StateIndependent::asExpressionAt(
        int seqIndex, Expression<Fuzzy>::Handle& p)
{
    itsASExpr[seqIndex]=p;
    return *this;
};
inline StateIndependent& StateIndependent::linkExpressionAt(
        int seqIndex, Expression<Fuzzy>::Handle& p)
{
    itsLinkExpr[seqIndex]=p;
    return *this;
};
inline StateIndependent& StateIndependent::linkPatternAt(
        int seqIndex, Condition::Handle& p)
{
    itsLinkCond[seqIndex]=p;
    return *this;
};
inline StateIndependent&  StateIndependent::globalCx(
        Context::Handle & g)
{
    itsGlobalCx = g; return *this;
};

inline Protein::Handle StateIndependent::protein(void)const{
    return itsProtein;
};
inline const MyString& StateIndependent::allCXListName(void)const{
    return itsAllCXListName;
}
inline ContextList::Handle StateIndependent::allCXList(void)const{
    return itsAllCXList;
};
inline ContextList::Handle StateIndependent::asListAt(int seqIndex)const{
    return itsASList[seqIndex];
};
inline Expression<Fuzzy>::Handle StateIndependent::asExpressionAt(int seqIndex)const{
    return itsASExpr[seqIndex];
};
inline Expression<Fuzzy>::Handle StateIndependent::linkExpressionAt(int seqIndex)const{
    return itsLinkExpr[seqIndex];
};
inline Condition::Handle StateIndependent::linkPatternAt(int seqIndex)const{
    return itsLinkCond[seqIndex];
};
inline Context::Handle StateIndependent::globalCx(void)const{
    return itsGlobalCx;
};
inline const DLList<Peak::Handle>&  StateIndependent::baseList(void)const
{
    return itsBaseList;
}
inline const ResourceName& StateIndependent::basePeakResName(void)const{
    return itsBasePeakResName;
};
inline const ResourceName& StateIndependent::getLeftResName(int seqIndex)const{
    return itsLeftResName[seqIndex];
};
inline const ResourceName& StateIndependent::getRightResName(int seqIndex)const{
    return itsRightResName[seqIndex];
};

inline StateIndependent& StateIndependent::getLeftTypeNamePrefix(
        const MyString& prefix)
{
    itsLeftTypeNamePrefix = prefix;
    return *this;
}
inline const MyString&  StateIndependent::getLeftTypeNamePrefix(void)const
{
    return itsLeftTypeNamePrefix;
}
inline StateIndependent& StateIndependent::setRightTypeNamePrefix(
        const MyString& prefix)
{
    itsRightTypeNamePrefix = prefix;
    return *this;
}
inline const MyString&  StateIndependent::getRightTypeNamePrefix(void)const
{
    return itsRightTypeNamePrefix;
}

inline const intVec& StateIndependent::asForPeak(int id){
    return itsASForPeak[id];
};
inline const intVec& StateIndependent::peakAtAS(int id){
    return itsPeaksForAS[id];
};
////////////////////////////////////////////////////
//
inline double&             StateIndependent::setASQFactorAt(int pos){
    return  itsASQFactor[pos];
}
inline double              StateIndependent::asQFactorAt(int pos)const{
    return  itsASQFactor[pos];
}
inline const doubleVec&          StateIndependent::asQFactor(void)const{
    return  itsASQFactor;
}
////////////////////////////////////////////////////
//
inline const MyString&      StateIndependent::calculatorName(void)const{
    return itsCalculatorName;
};
////////////////////////////////////////////////////
//
inline RandomOpti_Calc_noCache::Handle StateIndependent::calculator(void){
    return itsCalc;
};
////////////////////////////////////////////////////
// STATE
////////////////////////////////////////////////////
// static
inline long State::vetoPeriod(void){
    return theVetoPeriod;
}
// static
inline void  State::vetoPeriod(long i){
    theVetoPeriod = i;
}
////////////////////////////////////////////////////
inline State::~State(void){
    ActiveAccount ac(itsAccount); // used in lock dtor!
    for(int i=0; i<itsASLock.count(); i++){
        itsASLock[i].release(true);
    }
};
inline StateIndependent::Handle State::usedIndependent( void)const{
    return itsIndependent;
}
inline int State::getTTL(void)const{
    return itsTTL;
```

```
};
inline State& State::setTTL(int ttl){
    itsTTL = ttl;
    return *this;
};
inline State& State::setEpoch(long e){
    if(itsCount<e) itsCount = e; return *this;
};
inline Account&         State::account(void){
    return itsAccount;
};

inline int State::asCount(void)const
{
    return itsAS.count();
}

inline int State::linkCount(void)const
{
    return itsLink.count();
}
inline long  State::loopCountAt(int i)const{
    return itsLoopCount[i];
};
inline long& State::loopCountAt(int i){
    return itsLoopCount[i];
};

inline State& State::pushChange( int pos ){
    if(!itsChangedAS.get(pos)){
        itsChangedASStack[itsChangedASCount++] = pos;
        itsChangedAS.set(pos,true);
        loopCountAt(pos) = itsCount;
    }
    return *this;
}
inline int State::popChange( void ){
    return itsChangedASStack[--itsChangedASCount];
}
int State::changeCount(void) const {
    return itsChangedASCount ;
}
inline int State::changeAt(int i)const{
    return itsChangedASStack[i];
}
inline int State::topChange(void)const{
    return itsChangedASStack[itsChangedASCount -1];
}
inline State& State::declareUnchanged(int pos){
    itsChangedAS.set(pos,false); return *this;
}
inline State& State::clearChanges(void){
    itsChangedAS.fill(false);
    itsChangedASCount = 0;
    return *this;
}

inline int   State::usedCount(void)const{
    return itsAssignedCount;
}
inline int&   State::usedCount(void){
    return itsAssignedCount;
}
inline int    State::usedLinks(void)const{
    return itsValidLink.countValid();
};
#ifdef WITH_VETOPERIOD
inline bool  State::canChangeAt(int i)const{
    return (loopCountAt(i)+State::vetoPeriod() <= itsCount);
}
#else
inline bool  State::canChangeAt(int UNUSED(i))const{
    return true;
}
#endif // WITH_VETOPERIOD

inline State& State::nextEpoch(void){
    itsCount++; itsTTL--; return *this;
}
inline const Context::Handle& State::asAt(int i)const{
    return itsAS[i];
}
inline void State::putASAt(int i, const Context::Handle& cx){
    if(!isNull(itsAS[i])) usedCount()--;
    itsAS[i] = const_cast<IMngPointer<Context> &>(cx);
    if(!isNull(cx)) usedCount()++;
}
inline double& State::asQAt(int i){
    return itsASQ[i];
}
inline double State::asQAt(int i)const{
    return itsASQ[i];
}
inline double& State::asTypeQAt(int i){
    return itsTypeQ[i];
}
inline double State::asTypeQAt(int i)const{
    return itsTypeQ[i];
}
inline int              State::basePeakIDAt(int i)const
{
    return usedIndependent()->extractBaseID(asAt(i));
}
inline Context::Handle& State::linkAt(int i){
    return itsLink[i];
}
inline double& State::linkQAt(int i){
    return itsLinkQ[i];
}
inline ContextLock& State::asLockAt(int i){
    return itsASLock[i];
}
inline StatArray<bool>& State::validAS(void){
    return itsValidAS;
}
inline StatArray<bool>& State::validLink(void){
    return itsValidLink;
}
inline StatArray<bool>& State::changedAS(void){
    return itsChangedAS;
}
inline double& State::quality(void)
{
    return itsQ;
}
inline double State::quality(void)const
{
    return itsQ;
}
inline Protein::Handle State::protein( void)const{
    return itsIndependent->protein();
};
inline ContextList::Handle State::asListAt(int asIndex)const{
    return itsIndependent->asListAt(asIndex);
};
inline Expression<Fuzzy>::Handle State::asExpressionAt(int asIndex)const{
    return itsIndependent->asExpressionAt(asIndex);
};
inline Expression<Fuzzy>::Handle State::linkExpressionAt(int asIndex)const{
    return itsIndependent->linkExpressionAt(asIndex);
};
inline Condition::Handle State::linkPatternAt(int asIndex)const{
    return itsIndependent->linkPatternAt(asIndex);
};
#endif // __RANDOM_OPTI_STATE_INDEPENDENT_H__
```

N.10 Quelltext: RandomOpti_StateIndependent.cc

```
//
//
// $Id: RandomOpti_StateIndependent.cc,v 1.21 2000/06/12 12:08:51 va Exp $
//
```

```
#define DONT_TRACE 1

#include "RandomOpti_StateIndependent.h"
#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include "NoDump.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "Cache.h"
#include "GetInput.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "gr_prob.h" // enableAS( )
#include "Protein.h"
#include "PeakPool.h"
#include "PeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "AsciiContextReader.h"
#include "ExpressionBuilder.h"
#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "SelectiveFrequencyContextCompOp.h"
#include "StructureContextCompOp.h"
#include "RandomOpti_Calc.h"

//////////////////////////////////////////////////////////////
#define  WITH_VETOPERIOD 1
#undef   WITH_VETOPERIOD


// define DONT_IGNORE_MISSING_AS_TYPE, if an AS type entry like "GRIE #x Ala" is missing
// it should trigger an error message. otherwise missing types are handled as quality 0.0.
// default is (not defined)
//#define DONT_IGNORE_MISSING_AS_TYPE
#undef  DONT_IGNORE_MISSING_AS_TYPE

// enable all print statements for calculation tracing
// #define DEBUG_CALCULATION_Q 1
// enable the detection of wrong calculations
// #define DEBUG_TRACE_CALCULATION_Q 1
// #define   moveLinkedRange_DEBUG_CALCULATION_Q
#define   crossOverAt_DEBUG_CALCULATION_Q 1
#define  SHOW_Q_TRACE   false

#ifdef     DEBUG_CALCULATION_Q
#define   moveLinkedRange_DEBUG_CALCULATION_Q 1
#define   crossOverAt_DEBUG_CALCULATION_Q 1
#define   DEBUG_TRACE_CALCULATION_Q 1
#endif  // DEBUG_CALCULATION_Q


//////////////////////////////////////////////////////////////
// RandomOptimizer::evaluateArena
#define FOLLOW_BEST_Q    1
//#define FOLLOW_MID_Q     1

#define  EVALUATE_ARENA_REPLACE_ALL_BELOW_THRESHOLD
////////////////////////////////////////////////////////////////
////////////////////////////////////////
//
//
int binsearch(const intVec &x, int key){
    const bool notFoundReturnsNeg = true;
    int len = x.count();
    int l = 0; int r= len-1; int k=0;
    if(x.count()<1) return -1;
    if(key<x[0])    return -1;
    if( notFoundReturnsNeg ) {
        if(key>x[len-1]) return -1;
    } else {
        if(key>x[len-1]) return len;
    }
    while(r>=l){
   k = (l+r)/2;
   if(key<x[k])r = k-1; else l = k+1;
   if(key==x[k]) return (k);
    };

    if( notFoundReturnsNeg ) {
        if(x[k]!=key) return -1;
    } else {
        if(x[k]>key) return k-1;
        if(x[k]<key) return k;
    }
    return 0;
};
//////////////////////////////////////
// suche in Bereich [0..len] nach dem Element == key
//////
// return:
//       gefunden:  index des Elements
// nicht gefunden:    -1
int binsearchInRange(const intVec &x, int key, int len){
    const bool notFoundReturnsNeg = true;
    int l = 0; int r= len-1; int k=0;
    if(x.count()<1) return -1;
    if(key<x[0])    return -1;
    if( notFoundReturnsNeg ) {
        if(key>x[len-1]) return -1;
    } else {
        if(key>x[len-1]) return len;
    }
    while(r>=l){
   k = (l+r)/2;
   if(key<x[k])r = k-1; else l = k+1;
   if(key==x[k]) return (k);
    };

    if( notFoundReturnsNeg ) {
        if(x[k]!=key) return -1;
    } else {
        if(x[k]>key) return k-1;
        if(x[k]<key) return k;
    }
    return 0;
};
//////////////////////////////////////////////////////////////
StateIndependent::StateIndependent(void)
:  itsProtein(NULL), itsBasePeakResName("hnco CO-1 #1"),
   itsBaseListName("hnco"),
   itsCXIDsAreSetup(false),
   itsLeftTypeNamePrefix( "GRIE #1 "),
   itsRightTypeNamePrefix("GRIE #2 "),
   itsCalculatorName("as_link"),
   itsCalc(RandomOpti_Calc_noCache::make(itsCalculatorName))
{
    /* EMPTY */
};
StateIndependent::~StateIndependent(void)
{
    delete itsProtein;
}
//////////////////////////////////////////////////////////////
void StateIndependent::resize(int l){
    itsASList.resize(l);
    itsASExpr.resize(l);
    itsLinkExpr.resize(l);
    itsLinkCond.resize(l);
    itsLeftResName.resize(l);
    itsRightResName.resize(l);
    itsASQFactor.resize(l);
```

```
};
//////////////////////////////////////////////////////
//
StateIndependent& StateIndependent::calculatorName(const MyString& name){
    if(name != itsCalculatorName){
        itsCalculatorName = name;
        itsCalc = RandomOpti_Calc_noCache::make(name);
        if( !isNull(allCXList()) && allCXList()->length()>0)
            itsCalc->setupCache(this);
    }
    return *this;
};
//////////////////////////////////////////////////////////////
//: return the i-1 name for each AS
StateIndependent& StateIndependent::buildResName(void){
    itsLeftResName[0] = SharedString::null();
    for(int i=0; i<protein()->length(); i++){
        // generate 3char as name with first char upcase, second + third lower case
        // eg: "Ala",     ==> Grie modul name OR aa_info
        strstream namestream1;
        if(i>0){
            namestream1 << itsLeftTypeNamePrefix << protein()->get_3char(i-1) << ends;
            itsLeftResName[i] = ResourceName(namestream1.str()); namestream1.freeze(0);
        }
        strstream namestream2;
        namestream2 << itsRightTypeNamePrefix << protein()->get_3char(i) << ends;
        itsRightResName[i] = ResourceName(namestream2.str()); namestream2.freeze(0);
    }
    return *this;
};
//////////////////////////////////////////////////////////////
void                StateIndependent::setupAS2PeakMapping(void){
    ASSERT( baseList().length() > 0);
    ASSERT( protein() != NULL);

    itsPeaksForAS.resize(protein()->length());
    // bilde das Mapping  as -> peaks
    int lastPeakID = baseList()(baseList().last())->getID();
    itsASForPeak.resize(lastPeakID);   // first peak in pos 0 (index == ID-1)
    intVec peaks(lastPeakID);          // peaks of as         (index == ID-1)
    // loop from (1 .. protein.length)
    // pos AS 1 can't exist ( with patterns expecting i-1 peaks/frequencies )
    for(int asPos=1; asPos<protein()->length(); asPos++){
        peaks.fill(0);
        const ContextList::Handle& asList = asListAt(asPos);
        int count=0;
        for(ContextList::Iterator iter=asList->first(); iter; ++iter){
            int peakIndex = extractBaseID(*iter)-1;
            if(peakIndex > 0) {
                if(peaks[peakIndex]==0) count++; // zaehle jeden basis peak nur einmal
                peaks[peakIndex] ++;
            }
        }
        // erzeuge eine sortierte liste der peak ids, (unique entries)
        itsPeaksForAS[asPos].resize(count);
        int top = 0;
        for(int i=0; i<peaks.count();  i++){
            if(peaks[i] > 0) itsPeaksForAS[asPos][top++] = i;
        }
    }
    // bilde das inverse mapping  peak -> as
    intVec as(protein()->length()); // as of peak
    for(int p = 0; p<itsASForPeak.count(); p++){
        as.fill(0);
        int count = 0;
        for(int asPos=1; asPos<protein()->length(); asPos++){
            int index = binsearch(itsPeaksForAS[asPos],p);
            if(index != -1){
                if(as[asPos] == 0) count ++;
                as[asPos]++;
            }
        }
        // erzeuge eine sortierte liste der peak ids, (unique entries)
        itsASForPeak[p].resize(count);
        int top = 0;
        for(int i=0; i<as.count();  i++){
            if(as[i] > 0)   itsASForPeak[p][top++] = i;
        }
    }
};
//////////////////////////////////////////////////////////////
void                StateIndependent::setupCXIDs(void){
//     const ContextCompOp       compCX;
//     const ContextEqual        equalCX;
//     AVLMap<Context::Handle, long> cx2id(
//             -1,  &compCX, &equalCX
//             );
    long    nextItemID = 0; // start with 0, used as index!
    for(ContextList::Iterator iter=allCXList()->first(); iter; ++iter){
        if( (*iter)->id() != -1){
            ErrMessage err;
            err << fatal << "some cx ids are not == -1 before registration!"
                << endl;
        }
        (*iter)->id(nextItemID++);
    }
    itsCXIDsAreSetup = true;
}
//////////////////////////////////////////////////////////////
void  StateIndependent::setupCache(void)
{
    itsCalc->setupCache(this);
}
//////////////////////////////////////////////////////////////
int                 StateIndependent::extractBaseID(
        const Context::Handle & cx)const
{
    int id = 0;
    if(!isNull(cx)) {
        PeakNet::NodeIterator nIter;
        const Resource *r= cx->Lookup(basePeakResName());
        if(r==NULL || !r->isType( nIter )){
            if(r != NULL){
                ERRMESSAGE(err);
                err << error << "wrong type of resource\n"
                    << "name=\""<< stream << basePeakResName() <<"\"" ;
                err << eom;
            } else {
                ERRMESSAGE(err);
                err << error << "no such resource in context\n"
                    << " resource name=\""<< stream << basePeakResName()<<"\"";
                err << eom;
            }
            id = 0;
        } else {
            id = (*nIter)->getPeak()->getID();
        }
    } else {
        id = 0;
    }
    return id;
};

//////////////////////////////////////////////////////////////
int    StateIndependent::loadAllCXList(
        PeakPool::Handle poolH, PeakNet::Handle PN
        )
{
    enum{ ok = 0 , fail = 1 };

    ContextList::Handle list(new ContextList,IINIT);
    if(!loadContextList(*list , allCXListName() , *poolH, *PN,
                        globalCx()))
    {
        ErrMessage err;
        err << error
            << " while creating the ALL-CX as context list from file "
            << allCXListName() << eom;
        return fail;
    }
    itsAllCXList= list;
    return ok;
};
```

```
////////////////////////////////////////////////////////////
//static
StateIndependent::Handle StateIndependent::loadState(
        const MyString& name, PeakPool::Handle poolH, PeakNet::Handle PN,
        Context::Handle& globalContext )
{
    enum{ ok = 0 , fail = 1 };
    int errPos=0;
    AssertHook noStackDump(NULL);

    StateIndependent::Handle sH(new StateIndependent, IINIT);
    StateIndependent& S = *sH;

    S.globalCx(globalContext);


    fstream is(name, ios::in|ios::nocreate);
    if(!is){
        ErrMessage err;
   err << error<< "loadState:"<< "can't open config file";
        return StateIndependent::Handle();
    }

    MyStringVec  asListName;
    MyStringVec  asExprName;
    MyStringVec  linkExprName;
    MyStringVec  linkPatternName;
    MyString     allCXListName;
    MyString     asQFactorListName;

    bool hasAllCX = false;

    ReadBlock  block1(is,"RandomSequenceOptimizerInfo");
    ReadBlock  block2(is,"ASINFO");
    ReadBlock  block3(is,"LINKINFO");
    MyString headerVersion, linkInfoVersion, asInfoVersion, cacheDir;
    int asLen, linkLen;
    const char sep = ',';
    char  c;

    asInfoVersion   = "1";
    linkInfoVersion = "1";

    if(!block1.open()) ERROR_RECOVER;
    if(parseDQField(is, "VERSION",headerVersion)) ERROR_RECOVER;
    ////////////////////////////////////
    // headerVersionen
    //
    // 1   nur Protein & cachedir im header
    // 2   use_allcx_list && allcx_list im header
    // 3   mit asQFactorListName (== nutzungsfaktor je as)
    //     und Versionsangabe im  im ASINFO/LINKINFO TEIL

    if( headerVersion != "1" &&
        headerVersion != "2" &&
        headerVersion != "3") ERROR_RECOVER;

    if(parseDQField(is, "PROTEIN", S.itsProteinName)) ERROR_RECOVER;
    if(parseDQField(is, "CACHEDIR", cacheDir)) ERROR_RECOVER;
    if(headerVersion >= "2"){
        int x;
        if(parseField(is, "USE_ALLCX_LIST", x)) ERROR_RECOVER;
        hasAllCX = x;
        if(parseDQField(is, "ALLCX_LIST",   allCXListName)) ERROR_RECOVER;
        if(headerVersion >= "3"){
            if(parseDQField(is, "AS_QFACTOR_LIST",   asQFactorListName)) ERROR_RECOVER;
        }
    }
    if(!block2.open()) ERROR_RECOVER;
    if( headerVersion >= "2" ){
        if(parseDQField(is, "VERSION",asInfoVersion)) ERROR_RECOVER;
    }
    if(parseField(is, "LENGTH", asLen) ) ERROR_RECOVER;

    yy_setfile(S.itsProteinName);
    if(S.itsProtein == NULL){
        S.itsProtein = Protein::make();
    }
    if(parse(*S.itsProtein)){
        ErrMessage err;
err << error<< "loadState:"
            << "cant parse Protein file " << S.itsProteinName;
        return StateIndependent::Handle();
    }
    yy_finish();

    enableAS(*S.itsProtein);
    if(asLen != S.protein()->length()) {
        ErrMessage err;
        err << error << "seed has wrong size. found="<< asLen
            << "  protein="<<S.protein()->length()<<eom;
        ERROR_RECOVER;
    }

    asListName.resize(asLen);
    asExprName.resize(asLen);
    for(int i=0; i< asLen; i++){
        int pos;

        if(!is || !(is >>ws>> pos >>ws>> c) || sep != c) ERROR_RECOVER;
        if(!pos-1 == i)  ERROR_RECOVER;

        // format v1: asid, "cx-list", "evaluator-name"
        if(asInfoVersion == "1"){
            if(!readDQString(is,asListName[i])) ERROR_RECOVER;
            if(!is || !(is >>ws>> c) || sep != c) ERROR_RECOVER;
            if(!readDQString(is,asExprName[i])) ERROR_RECOVER;
        }

    }
    if(!block2.close())  ERROR_RECOVER;
    if(!block3.open()) ERROR_RECOVER;
    if( headerVersion >= "2" ){
        if(parseDQField(is, "VERSION",linkInfoVersion)) ERROR_RECOVER;
    }
    if(parseField(is, "LENGTH", linkLen) || linkLen != asLen-1) ERROR_RECOVER;
    linkExprName.resize(linkLen);
    linkPatternName.resize(linkLen);
    for(int i=0; i< linkLen; i++){
        int pos;
        if(!is || !(is >>ws>> pos >>ws>> c) || sep != c) ERROR_RECOVER;
        if(!pos-1 == i)  ERROR_RECOVER;

        if(!readDQString(is,linkExprName[i])) ERROR_RECOVER;
        if(!is || !(is >>ws>> c) || sep != c) ERROR_RECOVER;
        if(!readDQString(is,linkPatternName[i])) ERROR_RECOVER;
    }
    if(!block3.close())  ERROR_RECOVER;
    if(!block1.close())  ERROR_RECOVER;

    /////////////// AS Q Factor , init
    S.itsASQFactor.resize(asLen);
    S.itsASQFactor.fill(1.0);
    /////////////// read AS Q Factor if avail (v >= 3)
    if( headerVersion >= "3" ){
        fstream f(asQFactorListName, ios::in);
        if(!f)
            ERROR_RECOVER;
        MyString asQFactorVersion;
        int len;
        ReadBlock  block4(f,"AS_QFACTOR");

        if(!block4.open()) ERROR_RECOVER;
        if(parseDQField(f, "VERSION",asQFactorVersion)) ERROR_RECOVER;
        if(parseField(f, "LENGTH", len) || len != asLen) ERROR_RECOVER;

        for(int i=0; i< asLen; i++){
            int pos; // as position (index  is  pos-1)
            if(!f || !(f >>ws>> pos >>ws>> c) || sep != c) ERROR_RECOVER;
            if(!pos-1 == i)  ERROR_RECOVER;

            if(!f || !(f >>ws>> S.itsASQFactor[i]) || sep != c) ERROR_RECOVER;
        }
```

```
    if(!block4.close())  ERROR_RECOVER;
}

//////////////////// POST READ initialization
{   // start -- post read block, covers variables,
    // makes GOTO in ERROR_RECOVER easier
    // (less ctor-dtor handling for compiler)
    S.resize(S.protein()->length());
    S.buildResName();

    const char dirSep = '/';
    ContextList::Handle   allH(new ContextList, IINIT);
    if(hasAllCX){
        S.allCXListName(allCXListName);
        if(S.loadAllCXList( poolH, PN )){
            return  StateIndependent::Handle();
        };
        allH = S.allCXList();
    }
    ContextList& allCx = *allH;
    S.itsAllCXList       = allH;

    for(int i=0; i<asLen; i++){
        // handle special names like [empty,leer, all]
        if( asListName[i] == "leer" || asListName[i] == "empty" ){
            ContextList::Handle list2(new ContextList, IINIT);
            S.asListAt(i,list2);
            cerr << "context list for AS#"<< i+1 << " is empty"<<endl;
            continue;
        } else if(asListName[i] == "all" ){
            // Oops, how will we get a "merged-all" list, if all AS use "all"?
            // A) one list names a merged-all version of all others, the rest uses "all"
            S.asListAt(i,allH);
            cerr << "context list for AS#"<< i+1 << " is \"merge-all\" list"<<endl;
            continue;
        }
        // load cx list from file
        strstream nameStream;
        nameStream << cacheDir << dirSep << dirForASNr( i+1 )
                   << dirSep <<  asListName[i]<< ends;
        MyString fileName = nameStream.str(); nameStream.freeze(0);
        ContextList::Handle list(new ContextList,IINIT);
        if(!loadContextList(*list , fileName, *poolH, *PN)){
            ErrMessage err;
            err << error
                << " while creating as context list for as # "
                << i+1 <<" from file "<< fileName
                << eom;
            return  StateIndependent::Handle();
        }
        ContextList::Handle list2(new ContextList, IINIT);
        for(ContextList::Iterator iter=list->first(); iter; ++iter){
            Context::Handle temp = *iter;
            temp->rw().setContainedContext(S.globalCx());
            if(allCx.contains(temp)){
                temp    = *allCx.find(temp);
            } else {
                allCx.add(*iter);
                temp    = *allCx.find(*iter);
            }
            list2->add(temp);
        }
        S.asListAt(i,list2);
    }
    StatArray<bool> flag(asLen);
    ExpressionDebugWrapper*         builder=NULL;

    flag.fill(false);
    for(int i=0; i<asLen; i++){
        if(!flag.get(i)){
            if((builder = ExpressionDebugWrapper::make(asExprName[i]))
               ==NULL)
            {
                ErrMessage err;
                err << error << "while creating as expression for as #"
                    << i+1 << " from file "
                    << asExprName[i] << eom;
                return  StateIndependent::Handle();
            };
            builder -> build();
            Expression<Fuzzy>::Handle expressionH
                = Expression<Fuzzy>::Handle(builder,IINIT);

            for(int n=i; n<asLen; n++){
                if(!flag.get(n) && asExprName[i] == asExprName[n]){
                    flag.set(n,true);
                    S.asExpressionAt(n,expressionH);
                }
            }
        }
    }

    flag.fill(false);
    for(int i=0; i<linkLen; i++){
        if(!flag.get(i)){
            if((builder = ExpressionDebugWrapper::make(linkExprName[i]))
               ==NULL)
            {
                ErrMessage err;
                err << error << "while creating link expression for link #"
                    << i+1 << " from  "
                    << linkExprName[i] << eom;
                return  StateIndependent::Handle();
            }
            builder -> build();
            Expression<Fuzzy>::Handle expressionH
                = Expression<Fuzzy>::Handle(builder,IINIT);

            for(int n=i; n<linkLen; n++){
                if(!flag.get(n) && linkExprName[i] == linkExprName[n]){
                    flag.set(n,true);
                    S.linkExpressionAt(n,expressionH);
                }
            }
        }
    }
    flag.fill(false);
    for(int i=0; i<linkLen; i++){
        if(!flag.get(i)){
            Condition::Handle currPatternH;
            if(!loadPattern(currPatternH, linkPatternName[i], PN)){
                ErrMessage err;
                err << error << "while creating link pattern for link #"
                    << i+1 << " from file  "
                    << linkPatternName[i] << eom;
                return  StateIndependent::Handle();
            }
            currPatternH->attachToPeakNet(PN);
            for(int n=i; n<linkLen; n++){
                if(!flag.get(n)
                   && linkPatternName[i] == linkPatternName[n])
                {
                    flag.set(n,true);
                    S.linkPatternAt(n,currPatternH);
                }
            }
        }
    }
    S.itsBaseList = poolH->getList(S.itsBaseListName);
    if(S.baseList().length() == 0){
        ErrMessage err;
    err << error << "base peak list \""
        << stream<<S.itsBaseListName<<"\"is empty ";
    err << eom;
    ERROR_RECOVER;
    }

    S.setupAS2PeakMapping();
    S.setupCXIDs();
    S.setupCache();
  } // end -- post read block

  return sH;
error_recover:
```

```
    ErrMessage err;
    err << error
        << " parsing OptimizerInfo \""<< name <<"\" failed. (code =  " << errPos
        << " at line " << streampos2lineNr(is,is.tellg())
        << ")"  << eom;
    return StateIndependent::Handle();
};

////////////////////////////////////////////////////////
// static
long State::theVetoPeriod = 5;

////////////////////////////////////////////////////////
//
State::State(void)
: itsIndependent(), itsTTL(10), itsQ(0.0),
  itsChangedASStack(),itsChangedASCount(0),itsAssignedCount(0),
  itsAS(), itsASLock(), itsASQ(), itsTypeQ(), itsChangedAS(), itsValidAS(),
  itsLink(), itsLinkQ(), itsValidLink(), itsCount(0), itsLoopCount()
{
    /* EMPTY */
};
//////////////////////////////////////////////////////////////
double State::extractASTypeQAt(int i, const ResourceName &name){
    ASSERT(!isNull(asAt(i)));
    double q = 0.0;
    const Resource *r= asAt(i)->Lookup(name);
    if(r==NULL || !r->isType( q )){
        if(r != NULL){
            ERRMESSAGE(err);
            err << error << "wrong type of resource in context of as#"<<i+1 << "\n"
                << "name=\""<<stream << name <<"\"" ;
            err << eom;
        } else {
#ifdef DONT_IGNORE_MISSING_AS_TYPE
            ERRMESSAGE(err);
            err << error << "no such resource in context of as#"<<i+1 << "\n"
                << "name=\""<<stream << name <<"\"";
            err << eom;
#endif // DONT_IGNORE_MISSING_AS_TYPE
        }
        return Fuzzy::False;
    }
    return q;
}
void State::buildASTypeQAt(int i){
   itsTypeQ[i] = 0.0;
   if(!isNull(asAt(i))) {
       itsTypeQ[i] +=  extractASTypeQAt(i,usedIndependent()->getLeftResName(i));
       itsTypeQ[i] +=  extractASTypeQAt(i,usedIndependent()->getRightResName(i));
   }
}

State& State::copyFrom(const State& src){
    if(&src != this){
        itsIndependent    = src.itsIndependent; // shared
        itsAccount.copyFrom(src.itsAccount);
        itsQ              = src.itsQ;
        itsChangedASStack = src.itsChangedASStack;
        itsChangedASCount = src.itsChangedASCount;
        itsAssignedCount  = src.itsAssignedCount;
        itsAS             = src.itsAS;
        {ActiveAccount active(itsAccount);
        for(int i=0; i<asCount(); i++){
            itsASLock[i].dupFrom(src.itsASLock[i]);
        }
        } // end of ActiveAccount
        itsASQ            = src.itsASQ;
        itsTypeQ          = src.itsTypeQ;
        itsChangedAS      = src.itsChangedAS ;
        itsValidAS        = src.itsValidAS;
        itsLink           = src.itsLink;
        itsLinkQ          = src.itsLinkQ;
        itsValidLink      = src.itsValidLink;
        itsCount          = src.itsCount;
        itsLoopCount      = src.itsLoopCount;
        itsTTL            = src.itsTTL;
    }
    return *this;
};

//: clears all search state variables
State& State::clearState(void){
    itsQ = Fuzzy::False;

    ActiveAccount ac(itsAccount); // called in dtor!

    Context::Handle cxNULL;
    itsAS.fill( cxNULL );
    itsAssignedCount = 0;
    itsASQ.fill(Fuzzy::False);
    itsTypeQ.fill(Fuzzy::False);
    itsValidAS.fill(false);
    itsChangedAS.fill(false);
    for(int i=0; i<itsASLock.count(); i++){
        itsASLock[i].release(true);
    }
    itsLink.fill( cxNULL );
    itsLinkQ.fill(Fuzzy::False);
    itsValidLink.fill(false);
    itsCount          = 0;
    itsLoopCount.fill(0);
    itsTTL            = 10;
    return *this;
}

//: clear AS at asIndex,  clears links at asIndex and asIndex-1
//  maintains itsAssignedCount and if requested Q
void State::clearAS( int asIndex, bool bookKeeping = true)
{
    if(SHOW_Q_TRACE) cout <<"CLEARAS " << asIndex ;
    if(SHOW_Q_TRACE) cout <<" qS="<<itsQ << " ";
    if(SHOW_Q_TRACE) cout <<" isNull=" << (isNull(asAt(asIndex))) << " ";
    if(SHOW_Q_TRACE) cout <<" qA="<< (asQAt(asIndex));
    if(SHOW_Q_TRACE) cout <<" lock="<< (itsASLock[asIndex].isAquired());
    if(!isNull(asAt(asIndex))){
        if(bookKeeping ){
            quality() -= asQAt(asIndex);
            if(SHOW_Q_TRACE) cout <<" dropASQ " ;
        }
        if(asIndex-1>0)          clearLink(asIndex-1,bookKeeping);
        if(asIndex<linkCount()) clearLink(asIndex,bookKeeping);

        itsValidAS.set(asIndex,false);
        putASAt(asIndex, Context::Handle());
        asLockAt(asIndex).release(true);

        pushChange(asIndex);
    }
    asQAt(asIndex)          = Fuzzy::False;
    asTypeQAt(asIndex)      = Fuzzy::False;

    if(abs(itsQ) < 1E-3) itsQ = 0.0;
    if(SHOW_Q_TRACE) cout <<"  qS:=" << itsQ << endl;
}
void State::clearLink( int linkIndex, bool bookKeeping = true)
{
    if(SHOW_Q_TRACE) cout <<" clear link index="<<linkIndex;
    if(SHOW_Q_TRACE) cout <<" qL="<<linkQAt(linkIndex);
    if(bookKeeping && !isNull(linkAt(linkIndex))){
        quality()  -=  linkQAt(linkIndex);
        if(SHOW_Q_TRACE) cout << " dropLinkQ " ;
    }
    itsLink[linkIndex] = Context::Handle();
    itsValidLink.set(linkIndex,false);
    itsLinkQ[linkIndex] = Fuzzy::False;
}
//////////////////////////////////////////////////////////////
void     State::baseIDsInRange(intVec& baseID, int firstASPos, int len)
{
    ASSERT(len<=baseID.count());
    for(int i=0; i<min(len,asCount()-firstASPos); i++){
        if(!isNull(asAt(i+firstASPos))){
```

```
            baseID[i] = int(basePeakIDAt(i+firstASPos)-1);
        } else {
            baseID[i] = -1;
        }
    }
}
//////////////////////////////////////////////////////////////////
int     State::countLinksInRange(
        int firstPos, int len
        )
{
    int count = 0;
    for(int pos=firstPos; pos<min(firstPos+len,linkCount()); pos++){
        if(!isNull(linkAt(pos))){
            ASSERT(!isNull(asAt(pos)));
            count++;
        } else {
            return count;
        }
    }
    return count;
}

//////////////////////////////////////////////////////////////////
bool State::OK(void)const{
    int failed = false;
    int lineNr = -1;
    int asIndex = -1;
    int as      = -1;
#define CHECK_FAILED   { failed = false; lineNr = __LINE__; asIndex = as; }
    if(!isNull(itsIndependent)){
        if(!failed && (itsAS.count() != itsIndependent->protein()->length())) CHECK_FAILED;
        if(!failed && (itsLink.count() != itsIndependent->protein()->length()-1)) CHECK_FAILED;
        if(!failed && (linkCount()+1 != asCount())) CHECK_FAILED;
    }
    if(!failed && (const_cast<State*>(this)->usedCount()<0 || const_cast<State*>(this)->usedCount()
   > asCount())) CHECK_FAILED;
    if(!failed && (const_cast<State*>(this)->changeCount()<0 ||
   const_cast<State*>(this)->changeCount() > asCount())) CHECK_FAILED;
    int change=0;
    int i=0;
    for(i= 0; i<asCount(); i++)
        if(const_cast<State*>(this)->changedAS().get(i)) change++;
    if(!failed && (const_cast<State*>(this)->changeCount()!=change)) CHECK_FAILED;
    int use=0;
    for(i= 0; i<asCount(); i++)
        if(!isNull(const_cast<State*>(this)->asAt(i))) use++;
    if(!failed && (const_cast<State*>(this)->usedCount()!=use)) CHECK_FAILED;
    if(!failed && (change>use)) CHECK_FAILED;
    if(!failed && (! asOK())) CHECK_FAILED;
    if(change == 0){
        // unveraendert == korrekte Q
    }

    if(failed){
        ERRMESSAGE(err);
        err << warning << "invariant failure for sequence!! failed test at line "
            << lineNr;
        if(i!=asCount()){
            err << " in iteration i="<<i;
        }
        err << eom;
        return false;
    }

    return true;
};
bool State::asOK(void)const{

    // moegliche Zustaende der AS Variablen
    // Variablen sind itsAS, itsASLock, itsASQ, itsValidAS, itsChangedAS
    // Zeitpunkt [leer == nach dem initialisieren]
    //   itsAS == NULL , !itsASLock.hasContext(), itsASQ == 0,
    //   itsValidAS == false, itsChangedAS == false
    // Zeitpunkt [nach einer Veraenderung, vor dem Berechnen ]
    // entweder leer UND itsChangedAS == false oder :
    // wenn itsChangedAS == true
    //   itsAS != NULL, itsASLock.hasContext() && itsASLock.isAquired(), itsASQ == 0,
    //   itsValidAS == false,
    // Zeitpunkt [nach dem Berechnen == alle AS+Link leer oder belegt und gueltig]
    //   itsAS == NULL , !itsASLock.hasContext(), itsASQ == 0,
    //   itsValidAS == false, itsChangedAS == false
    int failed = false;
    int lineNr = -1;
    int asIndex = -1;
#define CHECK_FAILED  { failed = false; lineNr = __LINE__; asIndex = as; }
    if(failed){
        ErrMessage err;
        err << warning << "invariant failure!! failed test at line "
            << lineNr << "  for AS # " << asIndex +1 << eom;
    }
#undef CHECK_FAILED
    return false;
};
bool State::linkOK(void)const{
    // moegliche Zustaende der Link Variablen
    // Variablen sind itsLink, itsLinkQ, itsValidLink
    // Zeitpunkt [leer == nach dem initialisieren]
    // Zeitpunkt [nach einer Veraenderung, vor dem Berechnen ]
    // Zeitpunkt [nach dem Berechnen == alle AS+Link leer oder belegt und gueltig]
    //
    return true;
};

//////////////////////////////////////////////////////////////////
State& State::useIndependent(StateIndependent::Handle p ){
    itsIndependent = p;

    itsChangedASStack.resize(protein()->length());

    itsAS.resize(protein()->length());
    itsASLock.resize(protein()->length());
    itsASQ.resize(protein()->length());
    itsTypeQ.resize(protein()->length());
    itsChangedAS.resize(protein()->length());
    itsValidAS.resize(protein()->length());
    itsLoopCount.resize(protein()->length());

    itsLink.resize(protein()->length()-1);
    itsLinkQ.resize(protein()->length()-1);
    itsValidLink.resize(protein()->length()-1);

    return *this;
}
MyString State::asFlagsAt(int i)const{
    MyString flags;
    char f[32];
    char * p = f; *p++ = '[';
    *p++ =(isNull(const_cast<State*>(this)->asAt(i)))? 'E':'_';  // empty
    //*p++ =( const_cast<State*>(this)->validAS(i) )? 'V':'_';
    *p++ = ']'; *p++ = '\0';
    return MyString(f);
};
MyString State::linkFlagsAt(int i)const{
    MyString flags;
    char f[32];
    char * p = f; *p++ = '[';
    *p++ =( isNull(const_cast<State*>(this)->linkAt(i)) )? 'E':'_'; // empty
    //*p++ =( const_cast<State*>(this)->validLink(i) )? 'V':'_';
    //*p++ =( asAt(i) == NULL )? 'E':'_';
    *p++ = ']'; *p++ = '\0';
    return MyString(f);
};

//////////////////////////////////////////////////////////////////

bool State::createState(
        istream& is, StateIndependent::Handle& ind,
        const PeakPool& Pool, const PeakNet& Net
        )
{
    AsciiContextReader Reader(&is,Net,Pool);
    enum {ok = 0, fail = 1};
    int errPos = 0;
    MyString s,s1;
```

```
#undef  ERROR_RECOVER
#define ERROR_RECOVER  { errPos = __LINE__; goto create_error_recover; }

    ActiveAccount ac(itsAccount);

    useIndependent(ind);
    clearState();

    is >> noskipws;
    MyString version;
    MyString protName;

    ReadBlock block(is,"RandomSequenceOptimizerState");
    ReadBlock block2(is,"AS");
    ReadBlock block3(is,"LINK");
    MyRegex rxASFlags("[Ee_]");
    MyRegex rxLinkFlags("[Ee_]");
    MyString flags;
    int nrOfContexts;
    int nrOfLinks;
    const char sep = ',';
    bool failed = false;
    Context::Handle cxNULL;
    ContextLock currASLock;
    AsciiContextWriter Writer(&cerr);
    SelectiveFrequencyContextEqual funcEq(1.0);
    // HACK: freqency names should come from a (global) config object.
    // HACK: freqency distances should come from a (global) config object.
    // (???)
    funcEq.add("CA #1",0.55);
    funcEq.add("CA-1 #1",0.55);
    funcEq.add("CB #1",0.55);
    funcEq.add("CB-1 #1",0.55);
    SelectiveFrequencyContextDistance distanceOp(1.0);
    distanceOp.add("CA #1",0.55);
    distanceOp.add("CA-1 #1",0.55);
    distanceOp.add("CB #1",0.55);
    distanceOp.add("CB-1 #1",0.55);

    if(!block.open())  ERROR_RECOVER;
    if(parseDQField(is,"VERSION", version)) ERROR_RECOVER;
    if(parseDQField(is,"PROTEINNAME", protName)) ERROR_RECOVER;
    if(parseField(is,"QUALITY",quality())) ERROR_RECOVER;
    if(version > "1"){
        int i;
        if(parseField(is,"USED_AS",i)) ERROR_RECOVER;   // seit version 1.01
        if(parseField(is,"USED_LINKS",i)) ERROR_RECOVER; // seit version 1.01
    }

    if(!block2.open()) ERROR_RECOVER;
    if(parseField(is,"LENGTH", nrOfContexts))  ERROR_RECOVER;
    usedCount() = 0;

    for(int i=0; i<nrOfContexts; i++){
        int pos=-1; char c; double Q;
        if(!is || !(is >>ws>> pos) || pos-1 != i) ERROR_RECOVER;
        if(!is || !(is >>ws>> c) || sep != c)  ERROR_RECOVER;
        if(!is || parseBrace(is,flags,'[',']',false) )  ERROR_RECOVER;
        if(!is || !(is >>ws>> c) || sep != c) ERROR_RECOVER;
        if(!is || !(is >>ws>> Q ) ) ERROR_RECOVER;
        if(!is || !(is >>ws>> c) || sep != c) ERROR_RECOVER;

        flags.upcase();
        asQAt(i)=Q;

        putASAt(i, cxNULL);
        asLockAt(i).release();
        if(!flags.matches(rxASFlags)) ERROR_RECOVER;

        if(!flags.contains("E")) // && asQAt(i) == Fuzzy::False)
        {
            Context *tempC = new Context(ind->globalCx(),Context::Containing);
            Reader.read(*tempC);
            Context::Handle currAS(tempC,IINIT);

// suche cx in ind->asListAt(i)
// fehler wenn nicht vorhanden.
// sonst setze cx aus ind->asListAt(i)
bool found=false;
for(ContextList::Iterator iter=asListAt(i)->first();iter;++iter){
    if(stableComp(**iter,*currAS)==0){
        currAS = *iter;
        found = true;
        break; // foreach cx in asListAt(i)
    };
}
if(!found){
    ErrMessage err;
    AsciiContextWriter Writer(&err.stream());
    err << error << "context list for AS# "<<i+1
        << " contains no such context\n" ;
    Writer.write(*currAS);
    // err << eom;

    Context::Handle orgAS( new Context(*tempC), IINIT);
    // suche nach einem aehnlichen CX
    // funktionaler Sub oder EQ CX

    err << " ..looking for structual equivalent CX ";
    for(ContextList::Iterator iter=asListAt(i)->first();iter;++iter)
    {
        if(isStructuralEqual(*iter,orgAS)){
            currAS = *iter;
            found = true;
            break; // foreach cx in asListAt(i)
        };
    }
    err << (found ? " FOUND": "NOT FOUND")<<"\n";
    if( ! found ){
        err << " ..looking for functional equal CX ";
        // dist : die niedrigste Distanz, init als maximaler Abstand.
        double dist = MAXDOUBLE;
        for(ContextList::Iterator iter=asListAt(i)->first();iter;++iter)
        {
            if(funcEq(*iter,orgAS)){
                double currDist = distanceOp(*iter,orgAS);
                if( !found || currDist<dist){
                    currAS = *iter;
                    dist   = currDist;
                }
                found = true;
            };
        }
        err << (found ? " FOUND": "NOT FOUND")<<"\n";
    }
    failed = true;
    // alternative: fuege currAS in die aktuelle Liste ein und
    //  akzeptiere ihn.
    if(found){
        err << " ...accepted CX:"<<"\n";
        Writer.write(*currAS);
    } else {
        // loesche diese Position
        asQAt(i)= Fuzzy::False;
        err << " ... did not accept a CX!"<<"\n";
        continue; // next entry , for(i<nrOfContexts)
    }
}
asLockAt(i).aquire(currAS);
if(!asLockAt(i).isAquired()){
    ErrMessage err;
    AsciiContextWriter Writer(&err.stream());
    err << warning
        << "can't aquire Context of AS#"<<i+1<<", dropped.\n";
    Writer.write(*currAS);
    err << eom;
    clearAS(i);

    asQAt(i)= Fuzzy::False;

} else {
    putASAt(i,currAS);
```

```
                pushChange(i);
            }
        } else {
            // cx == NULL und Q == Fuzzy::False
            // asAt(i) bleibt leer
            asQAt(i)= Fuzzy::False;
        }
    }  //   for (i<nrOfContexts)
    if(!block2.close()) ERROR_RECOVER;
    if(failed){
        bool b = true;
        getInput(b, "Cache and State are incompatible, ignore error == continue loading?" , b);
        if(!b) ERROR_RECOVER;
    }

    if(!block3.open()) ERROR_RECOVER;
    if(parseField(is,"LENGTH", nrOfLinks)) ERROR_RECOVER;
    for(int i=0; i<nrOfLinks; i++){
        int pos; double Q; char c;
        if(!is || !(is >> ws >> pos >> ws >> c) || sep != c) ERROR_RECOVER;
        if(!pos-1 == i)  ERROR_RECOVER;
        if(!is || parseBrace(is,flags,'[',']',false) )  ERROR_RECOVER;
        if(!is || !(is >> ws >> c) || sep != c) ERROR_RECOVER;
        if(!is || !(is >> ws >> Q >> ws >> c) || sep != c) ERROR_RECOVER;
        flags.upcase();

        linkQAt(i)=Q;
        linkAt(i) = cxNULL;

        if(!flags.matches(rxLinkFlags)) ERROR_RECOVER;

        // lies den link CX, egal ob er Nachbarn hat oder nicht.
        if(!flags.contains("E")){
            Context *tempC = new Context(ind->globalCx(),Context::Containing);
            Reader.read(*tempC);
            Context::Handle currLink(tempC,IINIT);
            // wenn flags und linker(i) und rechter(i+1) Nachbar
            // einen cx enthalten .
            if(i+1<asCount() && !isNull(asAt(i)) && !isNull(asAt(i+1))) {
                // bilde den link Context und vergleiche ihn mit dem
                // gelesenen.
                linkAt(i) = currLink;
            } else {
                linkQAt(i) = Fuzzy::False;
            }
        } else {
            // cx == NULL und Q == Fuzzy::False
            // linkAt(i) bleibt leer
            linkQAt(i) = Fuzzy::False;
        }
    }

    if(!block3.close()) ERROR_RECOVER;
    if(!block.close())  ERROR_RECOVER;
    // unlock
    for(int i=0; i<nrOfContexts; i++){
        asLockAt(i).release(false); // nur entsperren, nicht loslassen
    }

    return ok;
  create_error_recover:
    ErrMessage err;
    err << error
        << " parsing seed failed. (code =  " << errPos
        << " at line " << streampos2lineNr(is,is.tellg())
        << ")"  << eom;
    return fail;
};

void State::dumpState(ostream& os){
    const char *sep = ",";
    AsciiContextWriter Writer(&os);
    WriteBlock block(os,"RandomSequenceOptimizerState");
    if(block){
    IndentLevel ilevel(os);
    os << indent << "VERSION  \"1.01\"" << endl;
        os << indent << "PROTEINNAME  \"" << protein()->name() <<'\"'<< endl;
        os << indent << "QUALITY    " << quality() << endl;
        int a =0, l= 0;
        for(int as =0; as < asCount(); as++){
            if(!isNull(asAt(as))) a++;
        }
        for(int link =0; link < linkCount(); link++){
            if(!isNull(linkAt(link))) l++;
        }
        os << indent << "USED_AS     " << a << endl;
        os << indent << "USED_LINKS  " << l << endl;
        {
            WriteBlock block2(os,"AS");
            if(block2){
                os << indent << "LENGTH  " << asCount() << endl;
                for(int as =0; as < asCount(); as++){
                    os << indent << as+1 << sep
                       << asFlagsAt(as) << sep
                       << asQAt(as) << sep;
                    if(!isNull(asAt(as)))
                        Writer.write(*asAt(as));
                    else
                        os << endl;
                }
            }
        }
        {
            WriteBlock block2(os,"LINK");
            if(block2){
                os << indent << "LENGTH  " << linkCount() << endl;
                for(int link =0; link < linkCount(); link++){
                    os << indent << link+1 << sep
                       << linkFlagsAt(link) << sep
                       << linkQAt(link) << sep;
                    if(!isNull(linkAt(link)))
                        Writer.write(*linkAt(link));
                    else
                        os << endl;
                }
            }
        }
    }
};

#include "Array.cc"
template class Array<Context::Handle>;
template class Array<ContextList::Handle >;
template class Array<intVec>;
template class Array<SharedString>;
template class Array<Expression<Fuzzy>::Handle>;
template class Array<bool>;
template class Array<Condition::Handle>;

#include "Matrix.cc"
template class Matrix<bool>;
```

N.11 Quelltext: LinkBuilder.h

```
#ifndef  __LinkBuilder_H__
#define  __LinkBuilder_H__ 1

#include "RCObject.h"
#include "iptr.h"
#include "Context.h"
#include "ResourceName.h"
#include "AVLMap.h"
#include "CompOp.h"

class LinkBuilder : virtual public RCObject
{
public:
    typedef IMngObject<LinkBuilder> Handle;

    //: creates link builders based on their name
    static LinkBuilder::Handle make(const MyString& name);
```

```
    // LinkBuilder(const LinkBuilder& b);
    // LinkBuilder operator=(const LinkBuilder& b);

    //: makes this class abstract
    virtual ~LinkBuilder(void)=0;

    //: generates links
    virtual Context::Handle buildLink(
            const Context::Handle &leftCx,
            const Context::Handle &rightCx,
            const Context::Handle &globalCx)const=0;

protected:
    virtual void init(void)=0;
    LinkBuilder(void);
private:
};


#endif // __LinkBuilder_H__
```

N.12 Quelltext: LinkBuilder.cc

```
#include "LinkBuilder.h"
#include "macros.h"
#include "Debug.h"
#include "ErrMessage.h"

////////////////////////////////////////////////////////////
class TranslatingLinkBuilder : virtual public LinkBuilder
{
public:
    friend class LinkBuilder; // for ::make(name)
    typedef IMngObject<LinkBuilder> Handle;

    virtual ~TranslatingLinkBuilder(void)=0;

    virtual Context::Handle buildLink(
            const Context::Handle &leftCx,
            const Context::Handle &rightCx,
            const Context::Handle &globalCx)const;

protected:
    TranslatingLinkBuilder(void);
    virtual void initNameMaps(const char* left[], const char *right[]);
private:
    ////////////////////////////////////////
    //
    const CompOp<SharedString> resCompOp;
    const Equal<SharedString>  resEqual;
    AVLMap< ResourceName, ResourceName >  itsLeftNameMap;
    AVLMap< ResourceName, ResourceName >  itsRightNameMap;
};

////////////////////////////////////////////////////////////
class LinkBuilder_c21w_hf : virtual public TranslatingLinkBuilder
{
public:
    friend class LinkBuilder; // for ::make(name)
    typedef IMngObject<LinkBuilder> Handle;

    virtual ~LinkBuilder_c21w_hf(void);

protected:
    virtual void init(void);
    LinkBuilder_c21w_hf(void);
};


////////////////////////////////////////////////////////////
// creates the apropriate link builder based on name
//
/////
// MAINTAINANCE: add further link builders here
//
// static
LinkBuilder::Handle LinkBuilder::make(const MyString& name){
    if(name == "c21w.hf"){
        return  LinkBuilder::Handle(new LinkBuilder_c21w_hf,IINIT);
    }
    ErrMessage err;
    err << error << "unknown link builder \""<<name<<"\"" << endl;
    return LinkBuilder::Handle();
};
////////////////////////////////////////////////////////////
//
LinkBuilder::~LinkBuilder(void)
{
    /* EMPTY */
};
LinkBuilder::LinkBuilder(void)
{
    /* EMPTY */
};

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
//pure virtual
TranslatingLinkBuilder::~TranslatingLinkBuilder(void)
{
    /* EMPTY */
};

TranslatingLinkBuilder::TranslatingLinkBuilder()
: itsLeftNameMap(SharedString::null(),  &resCompOp, &resEqual),
  itsRightNameMap(SharedString::null(), &resCompOp, &resEqual)
{
   /* EMPTY */    // can't call (virtual) init (of Derived) !! CTOR !!
};

void TranslatingLinkBuilder::initNameMaps(const char*left[], const char* right[])
{
    int from, to;
    from = 0, to =1;
    while( left[from]!=NULL && left[to]!=NULL){
        itsLeftNameMap[SharedString(left[from])] = SharedString(left[to]);
        from+=2; to+=2;
    }
    from = 0, to =1;
    while( right[from]!=NULL && right[to]!=NULL){
        itsRightNameMap[SharedString(right[from])] = SharedString(right[to]);
        from+=2; to+=2;
    }
}
// virtual
Context::Handle TranslatingLinkBuilder::buildLink(
        const Context::Handle &leftCx,
        const Context::Handle &rightCx,
        const Context::Handle & globalCx) const
{
    Context::Handle linkH;
    if(isNull(leftCx) && isNull(rightCx))
        return linkH;

    linkH = Context::Handle(new Context(globalCx,
                                        Context::Containing),IINIT);
    Context::RW     rw(*linkH);
    // iterate over map keys and copy if exists
    // store under translated name (key,res) -> (value,res)
    for(Pix px = itsLeftNameMap.first(); !isNull(leftCx) && px ; itsLeftNameMap.next(px)){
        const Resource * res = leftCx->Lookup(itsLeftNameMap.key(px));
        if(res != NULL)
           rw.Assign(itsLeftNameMap.contents(px),*res);
    }
    for(Pix px = itsRightNameMap.first(); !isNull(rightCx) && px ; itsRightNameMap.next(px)){
        const Resource * res = rightCx->Lookup(itsRightNameMap.key(px));
        if(res != NULL)
            rw.Assign(itsRightNameMap.contents(px),*res);
    }

    return linkH;
```

```
};

//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
//virtual
LinkBuilder_c21w_hf::~LinkBuilder_c21w_hf(void)
{
    /* EMPTY */
};
LinkBuilder_c21w_hf::LinkBuilder_c21w_hf(void)
: TranslatingLinkBuilder()
{
    init(); // ! *always*  calls  LinkBuilder_c21w_hf::init ! (never derived::init)
};
// virtual
void LinkBuilder_c21w_hf::init(void)
{
    const char* left[] = {
        // from      ----> to
        // "(any experiment) CA-1 #1" , NULL,  dropped
        // "(any experiment) CB-1 #1" , NULL,  dropped
        "hnca CA #1"   , "hnca CA #1",
        "cbcanh CA #1" , "cbcanh CA #1",
        "cbcanh CB #1" , "cbcanh CB #1",
        "hnco CO-1 #1" , "hnco CO-1 #1",
        NULL, NULL // terminate Array
    };
    const char* right[] = {
        // from       ---->    to
        "hnca CA-1 #1"      , "hnca CA-1 #2",
        "cbcanh CA-1 #1"    , "cbcanh CA-1 #2",
        "cbcanh CB-1 #1"    , "cbcanh CB-1 #2",
        "cbca_conh CA-1 #1" , "cbca_conh CA-1 #2",
        "cbca_conh CB-1 #1" , "cbca_conh CB-1 #2",
        "hnco CO-1 #1"      , "hnco CO-1 #2" ,

        "GRIE #2 Gly"      , "GRIE #2 Gly",
        NULL, NULL // terminate Array
    };
    initNameMaps(left, right);
};

#include "Map.cc"
#include "AVLMap.cc"
template class AVLMap<ResourceName, ResourceName>;
```

N.13 Quelltext: StateMetrics.h

```
#ifndef __StateMetrics_H__
#define __StateMetrics_H__ 1

#include "Array.h"
#include "Context.h"
#include "RandomOpti_StateIndependent.h"

class action_rec;
class StateDiff;
//////////////////////////////////////////
class action_rec
{
public:
    typedef  action_rec Self;

    enum operations_e {
        op_noop=0,
        link_op_min=1, // the first link operation

        opl_sub=link_op_min, opl_add=link_op_min+1, // operations on links
        opl_sum=link_op_min+2,

        link_op_max=link_op_min+2 , // the last link operation
        as_op_min  =link_op_max+1,  // first as operation

        op_sub=as_op_min, op_add=as_op_min+1, // operations on
        op_swap=as_op_min+2,

        as_op_max=as_op_min+2,  // last as operation

        op_count=as_op_max+1  // count of operations
    };

    inline action_rec(void);
    inline action_rec(
            operations_e            e,
            int                     pos1,
            long                    cxID1,
            double                  q,
            int                     pos2  = -1,
            long                    cxID2 = -1
            );

    inline operations_e operation(void)const;
    inline operations_e inv_operation(void)const;
    inline Self&  operation(operations_e op);

    //:
    inline bool isLinkOp(void)const;
    inline bool isASOp(void)const;

    inline long   cxID(void)const;
    inline Self&  cxID(long cxID);

    inline int    position(void)const;
    inline Self&  position(int pos);

    inline long   cxID2(void)const;
    inline Self&  cxID2(long cxID);

    inline int    position2(void)const;
    inline Self&  position2(int pos);

    inline double quality(void)const;
    inline Self&  quality(double q);

    static const char* asString(operations_e e);

private:
    operations_e itsOp;
    int          itsPos;
    long         itsCxID;
    double       itsQ;
    int          itsPos2;
    long         itsCxID2;

    static const char *       itsOpNames[op_count];
    static const operations_e itsInverse[op_count];

    friend inline bool operator==(const action_rec& A, const action_rec& B);
    friend inline bool operator!=(const action_rec& A, const action_rec& B);
};
ostream& operator<<(ostream& os, const action_rec& a);

//////////////////////////////////////////
class StateDiff : public RCObject
{
public:
    typedef StateDiff Self;

    StateDiff(void);

    StateDiff(
            State::Handle fromH,
            State::Handle toH);


    const action_rec & at(int)const;
    action_rec & at(int);

    //: position of next unused position and count of used positions
    int          top(void)const;
    action_rec & pop(void);
    void         push(const action_rec & a);
```

```
    double quality(void)const;
    //: Die Anzahl der atomaren Aktionen, die in diesem Diff
    //  enthalten sind.
    int    atomDistance(void)const;
    //: Die Anzahl der CX, die von diesem Diff betroffen sind.
    int    cxDistance(void)const;


    const State::Handle & start(void)const;
    const State::Handle & end(void)const;
    Self&  start(const State::Handle &s);
    Self&  end(const State::Handle &s);

    //: bilde den inversen Pfad
    void invert(void);

    // bilde den pfad der operationen, die
    // dem state from in den den state to verwandeln.
    void pathTo(
            State::Handle fromH, State::Handle toH
            );

    // remove all noop link pairs .
    Self&   finalizePath(void);

    // try to replace some (--++) groups with swap operations
    Self&  reduceSwaps(void);

    // extract base peak ids in path, returns the top index
    int  baseIDs(
            longVec&   vec,
            //       State::Handle stateH,
            const ResourceName& root,
            const bool containFrom  = true,
            const bool containTo    = true,
            const bool ignoreNullCX = true
            );
    // extract all peak ids in path, returns the top index
    int  peakIDs(
            longVec&   vec,
            //       State::Handle stateH,
            const ResourceNameVec& root,
            const bool containFrom  = true,
            const bool containTo    = true,
            const bool ignoreNullCX = true
            );
    friend inline void swap( StateDiff& d1, StateDiff& d2 );
protected:
    void  convert(
            longVec&       vec,
            int&           top,
            State::Handle  stateH,
            const bool     ignoreNullCX = true
            );

    ////////////////////////////////////////
    int                itsTop;
    Array<action_rec> itsActionPath;

    State::Handle     itsStart;
    State::Handle     itsEnd;
};
inline void swap( StateDiff& d1, StateDiff& d2)
{
    swap(d1.itsTop, d2.itsTop);
    swap(d1.itsActionPath, d2.itsActionPath);
    swap(d1.itsStart, d2.itsStart);
    swap(d1.itsEnd, d2.itsEnd);
}
////////////////////////////////////////
// :apply
//     apply a diff object to an state
//
// State can be any State Object which uses the same Independent!
// It is not required to be one of the original State pair
// which formed the Diff Object!
//
// Although it sets the contexts to those of the Diff, the resulting State
// is incomplete because the routine can't call opti->evaluateAt()
// without a random opti object.  So, state->quality() is in transition.
// and you have to call opti->rebuildQAt( ) afterwards!
//
// The second type of defect after this routine is the lack
// of ->aquire() ->release() correctness, but opti->rebuildQAt( )
// fixes that, too! ( by possibly removing some cx's from the State )
//
// *Parameter apply_swaps
// SWAP  is different from ADD an SUB operations.
// Not all cx combinations that result from one State
// are swappable in another State. So, swaps are not always
// applicable.
//
// If apply_swaps is true, any SWAP in the Diff Object
// is treated as two sequential SUB and ADD operations!
// SWAP is *not* applied as an pair wise exchange operation
// of the cx's in the State, *instead* it replaces the cx's in the State
// with the cx's which were swapped in the original State pair!
//
void apply(
        State::Handle&  s,
        StateDiff&      diff,
        bool            apply_swaps = true);
////////////////////////////////////////////////////////////
//
// addiere die Veränderungen je Operation zu den Vektoren
//
void countChanges(
        StateDiff & diff,
        longVec&  perAS,longVec& perCX, longVec& perPeak,

        const ResourceName& root,
        // Sollen nur die hinzugekommenen CX
        // gezählt werden oder auch die gelöschten?
        const bool countDropped = false
        );


////////////////////////////////////////
long extractPeakID(
        const Context::Handle &HC,
        const ResourceName& root,
        bool  acceptMissingResource = true);

////////////////////////////////////////
inline bool isLinkOp(  const action_rec::operations_e& e)
{
    return  ( action_rec::link_op_min <= e &&
              e <= action_rec::link_op_max );
}
inline bool isLinkOp(  const action_rec& A)
{
    return  isLinkOp(A.operation());
}

////////////////////////////////////////
inline const char* action_rec::asString(operations_e e){
    return action_rec::itsOpNames[int(e)];
};
//////////////
inline action_rec::action_rec(void)
:itsOp(action_rec::op_noop),itsPos(-1),itsCxID(-1),
 itsPos2(-1),itsCxID2(-1)
{
    /* EMPTY */
};
inline action_rec::action_rec(
        action_rec::operations_e e,
        int                      pos,
        long                     cxID,
        double                   q,
        int                      pos2,
        long                     cxID2
        )
:itsOp(e),itsPos(pos),itsCxID(cxID), itsQ(q),
```

```
 itsPos2(pos2),itsCxID2(cxID2)
{
    /* EMPTY */
};
//////////////
inline action_rec::operations_e action_rec::operation(void)const
{ return itsOp;
}
inline action_rec::operations_e action_rec::inv_operation(void)const
{ return action_rec::itsInverse[itsOp];
}
inline action_rec&  action_rec::operation(operations_e op)
{ itsOp= op; return *this;
}

inline bool action_rec::isLinkOp(void)const{
    return (link_op_min <= operation() && operation() <=link_op_max );
};
inline bool action_rec::isASOp(void)const{
    return (as_op_min <= operation() && operation() <= as_op_max );
};

inline long action_rec::cxID(void)const
{
    return itsCxID;
}
inline action_rec& action_rec::cxID(long cxID)
{
    itsCxID = cxID; return *this;
}
inline long action_rec::cxID2(void)const
{
    return itsCxID2;
}
inline action_rec& action_rec::cxID2(long cxID)
{
    itsCxID2 = cxID; return *this;
}

inline int action_rec::position(void)const
{ return itsPos;
}
inline action_rec&  action_rec::position(int pos)
{
    itsPos=pos;    return *this;
}
inline int action_rec::position2(void)const
{ return itsPos2;
}
inline action_rec&  action_rec::position2(int pos)
{
    itsPos2=pos;    return *this;
}


inline double action_rec::quality(void)const
{
    return itsQ;
}
inline action_rec&  action_rec::quality(double q)
{
    itsQ=q;    return *this;
};

////////////////////////////
inline bool operator==(const action_rec& A, const action_rec& B)
{
    return (A.itsOp == B.itsOp
            && A.itsPos == B.itsPos
            && A.itsCxID == B.itsCxID
            && A.itsPos2 == B.itsPos2
            && A.itsCxID2 == B.itsCxID2
            && A.itsQ == B.itsQ);
}
inline bool operator!=(const action_rec& A, const action_rec& B)
{
    return !( A == B);
}

////////////////////////////
inline StateDiff::StateDiff(void)
: itsTop(0)
{
    /* EMPTY */
};

inline const State::Handle & StateDiff::start(void)const
{
    return itsStart;
};
inline const State::Handle & StateDiff::end(void)const
{
    return itsEnd;
}
inline StateDiff&  StateDiff::start(const State::Handle &s)
{
    itsStart = s;    return *this;

};
inline  StateDiff&  StateDiff::end(const State::Handle &s)
{
    itsEnd = s;    return *this;
}

inline const action_rec & StateDiff::at(int i)const{
    return itsActionPath[i];
};
inline action_rec & StateDiff::at(int i){
    return itsActionPath[i];
};

inline action_rec & StateDiff::pop(void)
{
    return itsActionPath[itsTop--];
};
inline void StateDiff::push(const action_rec & a){
    if(itsActionPath.count()<top()+1)
        itsActionPath.resize(top()*1.5+1);
    itsActionPath[itsTop++] = a;
};

inline double StateDiff::quality(void)const
{
    double q=0.0;
    for(int i=0; i<top(); i++){
        q += at(i).quality();
    }
    return q;
}

inline int    StateDiff::atomDistance(void)const
{
    int count = 0;
    for(int k=0;k<top(); ++k){
        if( !isLinkOp(at(k)))
            count++;
    }
    return count;
};

inline int    StateDiff::top(void)const
{
    return itsTop;
}

#endif //  __StateMetrics_H__ 1
```

N.14 Quelltext: StateMetrics.cc

```
#include "StateMetrics.h"

//////////////
```

```
// static
const char *  action_rec::itsOpNames[action_rec::op_count] =  {
    "noop",     "addLink",    "subLink",    "sumLinkQ",    "add",    "sub",    "swap"
};
const action_rec::operations_e
action_rec::itsInverse[action_rec::op_count] ={
    action_rec::op_noop,

    action_rec::opl_sub,
    action_rec::opl_add,
    action_rec::opl_sum,

    action_rec::op_sub ,
    action_rec::op_add ,
    action_rec::op_swap
};

//////////////////////////////////////////
void  convert(longVec& vec, State::Handle stateH)
{
    vec.resize(stateH->asCount());
     for(int i = 0; i<vec.count(); i++){
         if(!isNull(stateH->asAt(i))){
             vec[i] = stateH->asAt(i)->id();
         } else {
             vec[i] = -1;
         };
     }
}
//////////////////////////////////////////
/* aus Kerninghan/Ritchie "Programmieren in C" P.106  (Kap. 5.6)*/
void sortByPosition(Array<action_rec>& a, int left, int right)
{
    int last;
    if(left >= right)
         return;

     swap(a[left],a[(left+right)/2]);
     last = left;
     for(int i=left+1; i<= right; i++)
         if( a[i].position() < a[left].position() ){
             swap(a[++last],a[i]);
         }
     swap(a[left],a[last]);
     sortByPosition(a, left,   last-1);
     sortByPosition(a, last+1, right);
}

//////////////////////////////////////////
inline bool lessCX_Pos_OP( const action_rec& A, const action_rec& B)
{
    if( A.cxID() != B.cxID() )
    {
        return ( A.cxID() < B.cxID() );
    } else if( A.position() != B.position())
    {
        return ( A.position()< B.position() );
    } else if( int(A.operation()) < int(B.operation()) )
    {
        return true;
    }
    return false;
 }
/* aus Kerninghan/Ritchie "Programmieren in C" P.106  (Kap. 5.6)*/
void sortByCXAndPositionAndOperation(Array<action_rec>& a, int left, int right)
{
    int last;
    if(left >= right)
        return;

    swap(a[left],a[(left+right)/2]);
    last = left;
    for(int i=left+1; i<= right; i++)
        if( lessCX_Pos_OP(a[i], a[left]) ){
             swap(a[++last],a[i]);
        }
    swap(a[left],a[last]);
    sortByCXAndPositionAndOperation(a, left,   last-1);
    sortByCXAndPositionAndOperation(a, last+1, right);
}
//////////////////////////////////////////
inline bool lessPos_Op_Q( const action_rec& A, const action_rec& B)
{
    if( A.position() != B.position() ) {
        return (A.position() < B.position());
    } else if (A.operation() != B.operation()) {
        return (int(A.operation()) < int(B.operation()));
    } else if (A.quality() <  B.quality()){
        return true;
    }
    return false;
}
/* aus Kerninghan/Ritchie "Programmieren in C" P.106  (Kap. 5.6)*/
void sortByPosOperationQ(Array<action_rec>& a, int left, int right)
{
    int last;
    if(left >= right)
        return;

    swap(a[left],a[(left+right)/2]);
    last = left;
    for(int i=left+1; i<= right; i++)
        if( lessPos_Op_Q(a[i], a[left]) ){
            swap(a[++last],a[i]);
        }
    swap(a[left],a[last]);
    sortByPosOperationQ(a, left,   last-1);
    sortByPosOperationQ(a, last+1, right);
}

//////////////////////////////////////////
ostream& operator<<(ostream& os, const action_rec& a)
{
    os << action_rec::asString(a.operation())
       << " ";

    if(a.operation() == action_rec::opl_add
       || a.operation() == action_rec::opl_sub )
    {
        os << " [ "<<a.position()+1 <<","<< a.cxID()<<"]";
    } else if(a.operation() == action_rec::op_swap)
    {
        os << " items [" << a.cxID()    <<", "<< a.cxID2()
           << "] at ["   << a.position() +1 <<", "<< a.position2()+1
           << "]";
    } else {
        os << " item " << a.cxID()
           << " at "   << a.position() +1;
    }
    os  << " with "<<a.quality() ;

    return os;
}
bool isInverseLinkOpPair(  const action_rec& A, const action_rec& B)
{
    if( !isLinkOp(A) || !isLinkOp(B))
        return false;
    const double epsilon = 1e-6;
    if( A.position() == B.position() &&
        // a.q == -b.q  within epsilon
         abs(B.quality()+A.quality()) < epsilon &&
        A.operation() == B.inv_operation() )
        return true;

    return false;
}
bool isInverseLinkOpPair_wrongQ(const action_rec& A, const action_rec& B)
{
    if( !isLinkOp(A) || !isLinkOp(B))
        return false;
    const double epsilon = 1e-3;
    if( A.position() == B.position() &&
```

```
         // a.q == -b.q  within epsilon
        (abs(B.quality())-abs(A.quality())) < epsilon &&
        A.operation() == B.inv_operation() )
        return true;

     return false;
}

////////////////////////////////////////
////////////////////////////////////////
////////////////////////////////////////
StateDiff::StateDiff(
        State::Handle fromH,
        State::Handle toH)
: itsTop(0),itsActionPath(0),
  itsStart(fromH), itsEnd(toH)
{
    pathTo(itsStart, itsEnd);
};
////////////////////////////////////////
// bilde den pfad der operationen die aus
// dem state from den state to machen.
void  StateDiff::pathTo(
        State::Handle fromH, State::Handle toH
        )
{
    ASSERT( fromH->asCount() == toH->asCount() );

    action_rec noop_action;
    int lTop = 0;
    Array<action_rec> L(fromH->asCount()*4);
    L.fill(noop_action);
    itsTop = 0;

    itsActionPath.resize(fromH->asCount()*6); // AS(sub+add)+link(add+sub)*2
    itsActionPath.fill(noop_action);
    for(int i=0;i<fromH->asCount();i++){
        long from = -1;
        long to   = -1;

        if(!isNull(fromH->asAt(i))){
            from = fromH->asAt(i)->id();
        };
        if(!isNull(toH->asAt(i))){
            to   = toH->asAt(i)->id();
        };
        if(from == to){
            // noop
        } else if( from != -1 && to != -1){
            push(action_rec ( action_rec::op_sub, i, from,
                              -fromH->asQAt(i) ));
            push(action_rec ( action_rec::op_add, i, to,
                              toH->asQAt(i)  ));

            ////////// LINKS
            if( i>0 && !isNull(fromH->linkAt(i-1)) ){
                L[lTop++] =action_rec ( action_rec::opl_sub, i-1, i,
                                        -fromH->linkQAt(i-1) );
            }
            if( i<fromH->linkCount() && !isNull(fromH->linkAt(i)) ){
                L[lTop++] =action_rec ( action_rec::opl_sub, i, i+1,
                                        -fromH->linkQAt(i) );
            }

            if( i>0 && !isNull(toH->linkAt(i-1)) ){
                L[lTop++] =action_rec ( action_rec::opl_add, i-1, i,
                                        toH->linkQAt(i-1) );
            }
            if( i<toH->linkCount() && !isNull(toH->linkAt(i)) ){
                L[lTop++] =action_rec ( action_rec::opl_add, i, i+1,
                                        toH->linkQAt(i) );
            }
        } else if( from != -1){
            push(action_rec ( action_rec::op_add, i, from,
                              fromH->asQAt(i) ));
            if( i>0 && !isNull(fromH->linkAt(i-1)) ){
                L[lTop++] =action_rec ( action_rec::opl_add, i-1, i,
                                        fromH->linkQAt(i-1) );
            }
            if(i<fromH->linkCount() &&  !isNull(fromH->linkAt(i)) ){
                L[lTop++] =action_rec ( action_rec::opl_add, i, i+1,
                                        fromH->linkQAt(i) );
            }
        } else if( to != -1){
            push(action_rec ( action_rec::op_sub, i, to,
                              -toH->asQAt(i) ));
            if( i>0 && !isNull(toH->linkAt(i-1)) ){
                L[lTop++] =action_rec ( action_rec::opl_sub, i-1, i,
                                        -toH->linkQAt(i-1) );
            }
            if( i<toH->linkCount() && !isNull(toH->linkAt(i)) ){
                L[lTop++] =action_rec ( action_rec::opl_sub, i, i+1,
                                        -toH->linkQAt(i) );
            }
        }
    }
    // sortiere die Link Aktionen nach den positionen
    // und eliminiere doppelte.
    if( lTop> 0){
        sortByPosOperationQ(L,0,lTop-1);
        push( L[0] );
        for(int i=1;i<lTop;i++){
            if( L[i] != L[i-1]  && L[i].quality() != 0.0 )
                push( L[i] );
        }
        sortByPosOperationQ(itsActionPath,0,top()-1);
    }

};


StateDiff&  StateDiff::finalizePath(void)
{
    int orgTop = top();
    itsTop = 0;
    Array<action_rec> A(itsActionPath);
    // collect all link quality
    double q = 0.0;
    for(int i=0;i<orgTop;i++){
        if( isLinkOp(A[i])){
            q += A[i].quality();
        } else if(A[i].operation() == action_rec::op_noop) {
            i++; // skip it
        } else {
            push( A[i] );
        }
    }
    push( action_rec(action_rec::opl_sum,0,0,q) );

    sortByPosOperationQ(itsActionPath,0,top()-1);
    return *this;
}

StateDiff& StateDiff::reduceSwaps(void)
{
//    action_rec noop_action;
    int orgTop = top();
    itsTop = 0;
    Array<action_rec> A(itsActionPath);
    sortByCXAndPositionAndOperation(A,0,orgTop-1);
    // now all              [sub pos1] operations
    // stand in front of the [add pos1] operations
    // (in position i-1 relative to the add op)
    // that will simplify the search operations below.
    for(int i=0;i<orgTop-1;i++){
        if( A[i].operation()   == action_rec::op_sub &&
            A[i+1].operation() == action_rec::op_add &&
            A[i].position()    == A[i+1].position()
            )
        {
            // possibly one part of a swap pair
            // find the other pair (with the same cx's)
            int n= i+2;
            for(;n<orgTop-1;n++)
```

```
            {
                if( A[n].operation()   == action_rec::op_sub &&
                    A[n+1].operation() == action_rec::op_add &&
                    // pair of cx at n is equal to pair at i
                    A[n].cxID()   == A[i+1].cxID()&&
                    A[n+1].cxID() == A[i].cxID()
                    )
                {
                    // zwei aufeinander folgende nicht-link operationen
                    // alle mit dem gleichen CX
                    // wenn es ein zweites Paar von operationen
                    // mit den gleichen Positionen gibt, kann man sie
                    // durch eine swap operation ersetzen.
                    //
                    // operation pair at i
                    //  sub pos1 cx1
                    //  add pos1 cx2
                    //  ... and somewhere later at n
                    // operation pair at n
                    //  sub pos2 cx2
                    //  add pos2 cx1
                    // gives
                    //  swap pos1 pos2  cx1 cx2

                    double qDiff =  A[n].quality()
                        + A[i+1].quality()
                        + A[n+1].quality()
                        + A[i  ].quality();
                    action_rec sw( action_rec::op_swap,
                                   i,A[i].cxID(),qDiff,n, A[i+1].cxID());
                    push(sw);

                    A[i].operation(action_rec::op_noop);
                    A[i+1].operation(action_rec::op_noop);
                    A[n].operation(action_rec::op_noop);
                    A[n+1].operation(action_rec::op_noop);
                    i++;
                    break; // for( n ..)
                }
            }
            push( A[i] );
        } else {
            push( A[i] );
        }
    }
    sortByPosOperationQ(itsActionPath,0,top()-1);
    return *this;
}
//////////////////////////////////////////
/* aus Kerninghan/Ritchie "Programmieren in C" P.106  (Kap. 5.6)*/
void qsort(longVec& a, int left, int right)
{
    int last;
    if(left >= right)
        return;

    swap(a[left],a[(left+right)/2]);
    last = left;
    for(int i=left+1; i<= right; i++)
        if( a[i] < a[left] ){
            swap(a[++last],a[i]);
        }
    swap(a[left],a[last]);
    qsort(a, left,   last-1);
    qsort(a, last+1, right);
}

//////////////////////////////////////////
// Die Anzahl der CX, die von diesem Diff betroffen sind.
int  StateDiff::cxDistance(void)const
{
    longVec vec(top()*2); // SWAP actions need 2 CX ids per action!
    int  used = 0;

    for(int i = 0; i<top(); i++){
        if( !isLinkOp( at(i) ) ){
            if(at(i).operation() == action_rec::op_add)
            {
                if( at(i).cxID() != 0 )
                    vec[used++]= at(i).cxID();
            } else if( at(i).operation() == action_rec::op_sub)
            {
                if( at(i).cxID() != 0 )
                    vec[used++]= at(i).cxID();
            } else if( at(i).operation() == action_rec::op_swap){
                if( at(i).cxID() != 0 )
                    vec[used++]= at(i).cxID();
                if( at(i).cxID2() != 0 )
                    vec[used++]= at(i).cxID2();
            }
        }
    }

    // erzeuge einen Vector der jede ID nur einmal enthält.
    qsort(vec,0,used-1);
    int to = 0;
    for(int i = 1; i<used; i++){
        if(vec[to] == vec[i]){
            i++;
        } else {
            vec[to++] = vec[i];
        }
    }
    return to;
}

//////////////////////////////////////////
long extractPeakID(
        const Context::Handle &HC,
        const ResourceName& root,
        bool  acceptMissingResource /* = true */ )
{
    const Resource *r = NULL;
    long ret = -1;
    PeakNet::NodeIterator node;
    if((r = HC->Lookup(root))!=NULL){
        if(!r->isType(node)){
            ErrMessage err;
            err << fatal
                << "wrong type of resource for \""
                << stream << root
                << "\" expected NodeIterator, found "<<r->getClassName()
                << eom;
        } else {
            ret = node -> getPeak()->getID();
        }
    } else if(!acceptMissingResource){
        ErrMessage err;
        err << fatal
            << "no resource for \""
            << stream << root
            << "\" found."
            << eom;
    }
    return ret;
}
//////////////////////////////////////////
void extractResource(
        // return value
        PeakNet::NodeIterator& node,

        const Context::Handle &HC,
        const ResourceName& root,
        bool  acceptMissingResource /* = true */ )
{
    const Resource *r = NULL;
    node = PeakNet::NodeIterator();
    if((r = HC->Lookup(root))!=NULL){
        if(!r->isType(node)){
            ErrMessage err;
            err << fatal
                << "wrong type of resource for \""
                << stream << root
```

```
                << "\" expected NodeIterator, found "<<r->getClassName()
                << eom;
        }
    } else if(!acceptMissingResource){
        ErrMessage err;
        err << fatal
            << "no resource for \""
            << stream << root
            << "\" found."
            << eom;
    }
}
////////////////////////////////////////
void  StateDiff::convert(
        longVec&       vec,
        int&           top,

        State::Handle  stateH,
        const bool     ignoreNullCX /* = true */
        )
{
    top = 0;
    vec.resize(stateH->asCount());
    for(int i = 0; i<vec.count(); i++){
        if(!isNull(stateH->asAt(i))){
            vec[top++] = stateH->asAt(i)->id();
        } else if(!ignoreNullCX){
            vec[top++] = -1;
        };
    }
}
int  StateDiff::baseIDs(
        longVec& vec,

        const ResourceName& root,
        const bool containFrom,  /* = true, */
        const bool containTo,    /* = true, */
        const bool ignoreNullCX  /* = true */
        )
{
    int t = 0;
    int maxCount = top() * (containFrom + containTo);
    vec.resize(maxCount);
    for(int i = 0; i<top(); i++){
        if( !isLinkOp( at(i) ) ){
            if( containFrom ) {
                Context::Handle cx = itsStart->asAt(at(i).position());
                if(!isNull(cx)){
                    vec[t++] = extractPeakID(cx,root,false);
                } else if(!ignoreNullCX){
                    vec[t++] = -1;
                }
            }
            if( containTo ) {
                Context::Handle cx = itsEnd->asAt(at(i).position2());
                if(!isNull(cx)){
                    vec[t++] = extractPeakID(cx,root,false);
                } else if(!ignoreNullCX){
                    vec[t++] = -1;
                };
            }
        }
    }
    return t;
}
int  StateDiff::peakIDs(
        longVec& vec,

        const ResourceNameVec& root,
        const bool containFrom,  /* = true, */
        const bool containTo,    /* = true, */
        const bool ignoreNullCX  /* = true */
        )
{
    int t = 0;
    int maxCount = top() * root.count()* ( containFrom + containTo);
    vec.resize(maxCount);
    for(int i = 0; i<top(); i++){
        if( !isLinkOp( at(i) ) ){
            if( containFrom ) {
                Context::Handle cx = itsStart->asAt(at(i).position());
                for(int n = 0; n<root.count(); n++){
                    if(!isNull(cx)){
                        vec[t++] = extractPeakID(cx,root[n],false);
                    } else if(!ignoreNullCX){
                        vec[t++] = -1;
                    };
                }
            }
            if( containTo ) {
                Context::Handle cx = itsEnd->asAt(at(i).position2());
                for(int n = 0; n<root.count(); n++){
                    if(!isNull(cx)){
                        vec[t++] = extractPeakID(cx,root[n],false);
                    } else if(!ignoreNullCX){
                        vec[t++] = -1;
                    };
                }
            }
        }
    }
    return t;
}

//////////////////////////////////////////////////////////////
//
// zähle die Veränderungen je Operation
//
void countChanges(
        StateDiff & diff,
        longVec&  perAS,longVec& perCX, longVec& perPeak,

        const ResourceName& root,
        // Sollen nur die hinzugekommenen CX
        // gezählt werden oder auch die gelöschten?
        const bool countDropped = false
        )
{
    for(int k = 0; k<diff.top(); ++k){
        // nur die add/sub/swap operationen
        // tragen zu den Veränderungen bei, Links
        // entstehen implizit durch die anderen
        // Operationen.
        if(isLinkOp( diff.at(k))) continue;

        if(diff.at(k).operation() == action_rec::op_add)
        {
            // add: seite 1 enthält den eingefügten CX
            perAS  [diff.at(k).position()]++;
            if( diff.at(k).cxID() > 0 ){
                long baseID1 = extractPeakID(
                        diff.start()->asAt(diff.at(k).position())
                        ,root, false
                        );
                perCX [diff.at(k).cxID()]++;
                if(baseID1>-1)perPeak[baseID1]++;
            }
        } else if( diff.at(k).operation() == action_rec::op_sub)
        {
            // sub: seite 1 ist die gelöschte Position
            perAS  [diff.at(k).position()]++;
            if( countDropped && diff.at(k).cxID() > 0 ){
                long baseID1 = extractPeakID(
                        diff.start()->asAt(diff.at(k).position())
                        ,root, false
                        );
                perCX[diff.at(k).cxID()]++;
                if(baseID1>-1)perPeak[baseID1]++;
            }
        } else if(diff.at(k).operation() == action_rec::op_swap){
            // swap: beide seiten zählen als verbesserung
            perAS  [diff.at(k).position() ]++;
            perAS  [diff.at(k).position2()]++;

            if( diff.at(k).cxID() > 0 ){
```

```
                long baseID1 = extractPeakID(
                        diff.start()->asAt(diff.at(k).position())
                        ,root, false
                        );
                perCX  [diff.at(k).cxID()    ]++;
                if(baseID1>-1)perPeak[baseID1]++;
            }
            if( diff.at(k).cxID2() > 0 ){
                long baseID2 = extractPeakID(
                        diff.end()->asAt(diff.at(k).position2())
                        ,root, false
                        );
                perCX  [diff.at(k).cxID2()   ]++;
                if(baseID2>-1)perPeak[baseID2]++;
            }

        }
    }
}
////////////////////////////////////////
void apply(
        State::Handle&  S,
        StateDiff&      diff,
        // SWAP's are different from ADD and SUB operations
        // not all cx combinations are swapable.
        // So, SWAP's are not always applicable.
        bool apply_swaps /* = true */
        )
{
    // first run, apply op_sub and op_sub part of swaps
    for(int k = 0; k<diff.top(); ++k){
        if(diff.at(k).operation() == action_rec::op_sub){
            S->clearAS( diff.at(k).position() );
        } else if(apply_swaps && diff.at(k).operation() == action_rec::op_swap)
        {
            S->clearAS( diff.at(k).position() );
            S->clearAS( diff.at(k).position2() );
        }
    }
    // second run, apply all additions/swaps
    for(int k = 0; k<diff.top(); ++k){
        // nur die add/sub/swap operationen
        // tragen zu den Veränderungen bei, Links
        // entstehen implizit durch die anderen
        // Operationen.
        if(isLinkOp( diff.at(k))) continue;

        if(diff.at(k).operation() == action_rec::op_add)
        {
            Context::Handle cx;
            if( diff.at(k).position() > -1 ) {
                cx=diff.end()->asAt(diff.at(k).position());
            }
            S->putASAt(diff.at(k).position(),cx);
            /**** cant call evaluateLocalAt() without a random opti object ***/
        } else if(apply_swaps && diff.at(k).operation() == action_rec::op_swap)
        {
            Context::Handle cx;
            if( diff.at(k).position2() > -1 ) {
                cx=diff.end()->asAt(diff.at(k).position2());
            }
            S->putASAt(diff.at(k).position2(),cx);

            if( diff.at(k).position() > -1 ) {
                cx=diff.end()->asAt(diff.at(k).position());
            } else
                cx = Context::Handle();
            S->putASAt(diff.at(k).position(),cx);
        }
    }
    /**** cant call evaluateAt() without a random opti object ***/
    // => quality() is wrong / incomplete
    // you have to call ->rebuildQAt( )
}

////////////////////////////////////////
```

```
#include "Array.cc"
template class Array<action_rec>;
```

N.15 Quelltext: GeneticRandomOptimizer.h

```
#ifndef __GeneticRandomOptimizer_H__
#define __GeneticRandomOptimizer_H__  1
//
//
// $Id: GeneticRandomOptimizer.h,v 1.11 2000/05/29 16:23:25 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "Protein.h"
#include "PeakPool.h"
#include "PeakNet.h"
#include "LinkBuilder.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizerBASE.h"
#include "RandomOptimizer.h"

//////////////////////////////////////////////////////////////////

class StateIndependent;
class State;
class LinkBuilder;
class RandomOptimizerBASE;
class RandomOptimizer;
class CmdLine;
class CmdLineArgIter;

//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
// a wrapper around RandomOptimizer
//   disables  dumpState
//
class SubRandomOptimizer;
class SubRandomOptimizer : public RandomOptimizer
{
public:
    typedef SubRandomOptimizer Self;
    typedef IMngObject< RandomOptimizerBASE > Handle;

    virtual ~SubRandomOptimizer(void);
    SubRandomOptimizer(void);

    virtual void restartRun(void);
    virtual void dumpState(void);

    //: die Konfigurationsmethoden.
    static  void registerOptions(CmdLine& cmd);
    virtual bool configure(CmdLineArgIter& cmdIter);

protected:
    virtual Self& debugParams(void);
private:
};
//////////////////////////////////////////////////////////////////
class GeneticOptimizer;
class GeneticOptimizer : public RandomOptimizerBASE
```

```
{
public:
    typedef GeneticOptimizer            Self;
    typedef IMngObject< RandomOptimizerBASE > Handle;
    typedef SubRandomOptimizer SUBOptimizer;

    virtual ~GeneticOptimizer(void);
    GeneticOptimizer(void);

    void restartRun(void);

    //: Ausgabe der State Objekte auf dem Dumpfile.
    // INHERIT, virtual void dumpState(void);
    //: Lesen der State Objekte aus dem File "name".
    // INHERIT, virtual bool loadState(
    //                const MyString& name,
    //                PeakPool::Handle pool, PeakNet::Handle net);
    virtual bool loadConfig(istream & is);
    virtual void dumpConfig(ostream & os);

    //: Lesen der State Objekte aus den RO-Files in name.
    virtual bool collectStates(
            const MyStringVec& names);

    //: die konfigurationsmethoden.
    static  void registerOptions(CmdLine& cmd);
    virtual bool configure(CmdLineArgIter& cmdIter);

    //: führe die eigentliche Optimierung durch.
    virtual void optimize(void);
    //: soll optimize beendet werden?
    virtual bool abbruchbedingung(void);
    //: bewerte die Arena und verwaltet den besten State.
    virtual void evaluateArena(void);

    ////////////////////// Data Access Funtions
    inline Self& survivingBestSeedCount(int d);
    inline int   survivingBestSeedCount(void)const;

    inline Self& lengthOfSeedPool(int d);
    inline int   lengthOfSeedPool(void)const;

    inline Self& subRO(RandomOptimizerBASE::Handle H);
    inline RandomOptimizerBASE::Handle subRO(void)const;

    inline Self& finalRO(RandomOptimizerBASE::Handle H);
    inline RandomOptimizerBASE::Handle finalRO(void)const;

    inline Self& subMaxTime(long subMaxTime);
    inline long  subMaxTime(void)const;

    inline Self& subMaxLoops(long subMaxLoops);
    inline long  subMaxLoops(void)const;

    inline Self& genMaxLoopsWithEquivalentQ(long subMaxLoops);
    inline long  genMaxLoopsWithEquivalentQ(void)const;

    inline Self& isNotSorted(void);
    inline bool  isSortedByQ(void)const;
    inline Self& isSortedByQ(bool b);

    inline long  longestCountThread(void)const;
    inline long  seedLoopThread(void)const;

    //: Zerfallsrate für qThreshold
    GeneticOptimizer& thresholdDecayRate(double d);
    double thresholdDecayRate(void)const;

    //: start value for qThreshold
    GeneticOptimizer& initialQThreshold(double d);
    double initialQThreshold(void)const;

    //: current value for qThreshold
    inline Self&  genQThreshold(double d);
    inline double genQThreshold(void)const;

    //: schwelle des sliding-faktor für Crossover-Operationen
    GeneticOptimizer& crossoverThreshold(double d);
    double crossoverThreshold(void)const;
    virtual const char* crossoverOpName(void)const;
protected:
    //: verbessere einen State mit dem subRO
    virtual void singleOptimizeAt(int seqIndex);
    //: ruft singleOptimizeAt fuer alle States auf
    void  singleOptimizeAll(void);

    //: schliesst die Optimierung ab, indem sie den besten Zustand
    //  einer Einzeloptimierung mit eingeschraenkten Parametern
    //  unterwirft. Der State wird quasi in das naechste
    //  lokale Minimum gedrückt.
    void  finalOptimizeBest(void);

    //: Rebuild the arena from (bestSeedCount) seed states
    void crossOverAll(void);

    //:
    void sortArenaByQ(void);
    void qsortByQ(int left, int right);

    //:
    bool updateBest(void);

    //////////////
    Self& debugTraceEvolution(void);
    Self& debugTraceSingle(int seqIndex);
    Self& debugTraceCrossOver(const intVec &N,const intVec &I);
    Self& debugTracePopulationDiff(void);

    Self& debugFinal(void);

    //////////////
    int itsSurvivingBestSeedCount;
    int itsLengthOfSeedPool      ;

    RandomOptimizerBASE::Handle    itsSubRO;
    RandomOptimizerBASE::Handle    itsFinalRO;

    long                           itsSubMaxTime;
    long                           itsSubMaxLoops;

    // : no of loop with all qualities == bestQ
    int                            itsMaxLoopsWithEquivalentQ;
    //: initial threshold factor for quality
    double                         itsInitialQThreshold;
    //: decay factor for itsQThreshold
    double                         itsThresholdDecayRate;
    // :
    double                         itsCrossoverThreshold;
    /////////////// IMPLEMENTATION
    int                            itsLoopsWithEquivalentQ;
    bool                           itsIsSortedByQ;
    long                           itsSeedLoopThread;
    long                           itsLongestCountThread;
    // : schwelle, ab der einzelne States mittels crossover ersetzt werden
    double                         itsQThreshold;
    // :
    double                         itsSlidingCross;
    // :
    double                         itsCrossSumme;
};
//////////////////////////////////////////////////////////////
inline long   GeneticOptimizer::longestCountThread(void)const
{
    return itsLongestCountThread;
};
//////////////////////////////////////////////////////////////
inline long   GeneticOptimizer::seedLoopThread(void)const
{
    return itsSeedLoopThread;
};
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::survivingBestSeedCount(
        int d)
{
    itsSurvivingBestSeedCount = d;
    return *this;
```

```
};
inline int   GeneticOptimizer::survivingBestSeedCount(void)const
{
    return itsSurvivingBestSeedCount;
};
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::lengthOfSeedPool(
        int d)
{
    itsLengthOfSeedPool = d;
    return *this;
};
inline int   GeneticOptimizer::lengthOfSeedPool(void)const
{
    return itsLengthOfSeedPool;
};
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::subRO(
        RandomOptimizerBASE::Handle H)
{
    itsSubRO = H;
    return *this;
};
inline RandomOptimizerBASE::Handle GeneticOptimizer::subRO(void)const
{
    return itsSubRO;
};
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::finalRO(
        RandomOptimizerBASE::Handle H)
{
    itsFinalRO = H;
    return *this;
};
inline RandomOptimizerBASE::Handle GeneticOptimizer::finalRO(void)const
{
    return itsFinalRO;
};
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::subMaxTime(long subMaxTime)
{
    itsSubMaxTime = subMaxTime;
    return *this;
};
inline long  GeneticOptimizer::subMaxTime(void)const
{
    return itsSubMaxTime;
}
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::subMaxLoops(long subMaxLoops)
{
    itsSubMaxLoops = subMaxLoops;
    return *this;
};
inline long  GeneticOptimizer::subMaxLoops(void)const
{
    return itsSubMaxLoops;
}
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::genMaxLoopsWithEquivalentQ(
        long subMaxLoops)
{
    itsMaxLoopsWithEquivalentQ = subMaxLoops;
    return *this;
};
inline long  GeneticOptimizer::genMaxLoopsWithEquivalentQ(void)const
{
    return itsMaxLoopsWithEquivalentQ;
}
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::isNotSorted(void)
{
    itsIsSortedByQ = false;
    return *this;
};
inline bool  GeneticOptimizer::isSortedByQ(void)const
{
    return itsIsSortedByQ;
}
inline GeneticOptimizer& GeneticOptimizer::isSortedByQ(bool b)
{
    itsIsSortedByQ = b;
    return *this;
};
//////////////////////////////////////////////////////////////
inline GeneticOptimizer& GeneticOptimizer::genQThreshold(
        double d)
{
    itsQThreshold = d;
    return *this;
};
inline double   GeneticOptimizer::genQThreshold(void)const
{
    return itsQThreshold;
};
//////////////////////////////////////////////////////
//
inline GeneticOptimizer& GeneticOptimizer::initialQThreshold(double d){
    itsInitialQThreshold     = d;
    return *this;
};
inline double GeneticOptimizer::initialQThreshold(void)const{
    return itsInitialQThreshold;
};
//////////////////////////////////////////////////////
//
inline double GeneticOptimizer::thresholdDecayRate(void)const{
    return itsThresholdDecayRate;
};

inline GeneticOptimizer& GeneticOptimizer::thresholdDecayRate(double d){
    itsThresholdDecayRate = d;
    return *this;
};
//////////////////////////////////////////////////////
//
inline double GeneticOptimizer::crossoverThreshold(void)const{
    return itsCrossoverThreshold;
};

inline GeneticOptimizer& GeneticOptimizer::crossoverThreshold(double d){
    itsCrossoverThreshold = d;
    return *this;
};
#endif // __GeneticRandomOptimizer_H__
```

N.16 Quelltext: GeneticRandomOptimizer.cc

```
//
// $Id: GeneticRandomOptimizer.cc,v 1.15 2000/04/20 11:44:21 va Exp $
//
#define DONT_TRACE 1

#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include "NoDump.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "Cache.h"
#include "Protein.h"
```

```
#include "PeakPool.h"
#include "PeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "ParseCondition.h"
#include "LinkBuilder.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizerBASE.h"
#include "GeneticRandomOptimizer.h"
#include "StateMetrics.h"

//////////////////////////////////////////////////////////////
#include "RandomOptimizer.debug.h"
//////////////////////////////////////////////////////////////////
// GeneticOptimizer Paramters
static CmdArgInt       arg_genArenaSize(
        's',"genArenaSize","count",
        "number of states in the arena" ,
        CmdArg::isOPTVALREQ);
static CmdArgInt       arg_seedPoolSize(
        's',"seedPoolSize","count",
        "number of seeds in evolutionary process" ,
        CmdArg::isOPTVALREQ);
static CmdArgInt       arg_survivingSeeds(
        's',"survivingSeeds","count",
        "how many seeds should stay unchanged [2 .. 6]< seedPoolSize" ,
       CmdArg::isOPTVALREQ);
static CmdArgStrList   arg_seedFiles(
        'p',"seedFiles","filenames",
        "name of sub opti file names (seed for genetic opti)" ,
        CmdArg::isOPTVALREQ);

static CmdArgInt       arg_genMaxIdleLoops(
        's',"genMaxIdleLoops","count",
        "max number of idle loops" ,
        CmdArg::isOPTVALREQ);
static CmdArgInt       arg_genMaxEqualLoops(
        's',"genMaxEqualLoops","count",
        "max number of loops with all states equal" ,
        CmdArg::isOPTVALREQ);
static CmdArgFloat     arg_genQThreshold(
        'l',"genQThreshold","float",
        "Quality Threshold (% 0.0-1.0) of geneticOpt for crossover" ,
        CmdArg::isOPTVALREQ);
static CmdArgFloat     arg_genQThresholdDecayRate(
        's',"genQThresholdDecayRate","float",
        "decay rate of threshold in geneticOpt" ,
        CmdArg::isOPTVALREQ);
static CmdArgFloat     arg_genCrossOverThreshold(
        's',"genCrossOverThreshold","float",
        "threshold of sliding crossover rate in geneticOpt" ,
        CmdArg::isOPTVALREQ);
static CmdArgInt       arg_subMaxLoops(
        's',"subMaxLoops","count",
        "number loops in subprocess" ,
        CmdArg::isOPTVALREQ);
static CmdArgFloat     arg_subMaxTime(
        's',"subMaxTime","min",
        "time in subprocess" , CmdArg::isOPTVALREQ);
#ifdef HAS_STATE_TRACE_DUMP
static CmdArgStr     arg_traceStates(
        's',"traceStates","[all|digits]",
        "interval for state trace dumps " , CmdArg::isOPTVALREQ);
#endif // HAS_STATE_TRACE_DUMP
//////////////////////////////////////////////////////////////////
//
GeneticOptimizer::~GeneticOptimizer(void)
{
    /* EMPTY */
}

GeneticOptimizer::GeneticOptimizer(void)
: RandomOptimizerBASE(),
  itsSurvivingBestSeedCount( 2 ),
  itsLengthOfSeedPool( 4 ),
  itsSubRO(),
  itsSubMaxTime(60*4),
  itsSubMaxLoops(10000),
  itsMaxLoopsWithEquivalentQ(3),
  itsInitialQThreshold(0.9),
  itsThresholdDecayRate(0.005),
  itsLoopsWithEquivalentQ(0),
  itsIsSortedByQ(false),
  itsQThreshold(0.9),
  itsSlidingCross(0.0),
  itsCrossSumme(0.0)
{
    DEBUGTracedFrame;
    sequenceCount(8);
    // large TC's are fine for Optimizers with short cycle times
    // GeneticOptimizer' cycles are very large (~ some minutes )
    // if TC is larger than maxLoops, it can inhibit any dumps!
    itsInitialTC = 0;
    itsLongestCountThread = 0;
    itsSeedLoopThread = 0;
    // genetic optimizer zeigt in jedem Zyklus die Population an.
    itsPopulationDumpInterval = 0;
}

//////////////////////////////////////////////////////////////////
//
void GeneticOptimizer::registerOptions(CmdLine& cmd)
{
    DEBUGTracedFrame;
    static bool done = false;
    RandomOptimizerBASE::registerOptions(cmd);
    SUBOptimizer::registerOptions(cmd); // the internal optimizer

    if(!done){
        done = true;
        cmd << arg_survivingSeeds
            << arg_seedPoolSize
            << arg_seedFiles
            << arg_genArenaSize
            << arg_genMaxIdleLoops
            << arg_genMaxEqualLoops
            << arg_subMaxLoops
            << arg_subMaxTime
            << arg_genQThreshold
            << arg_genQThresholdDecayRate
            << arg_genCrossOverThreshold
#ifdef HAS_STATE_TRACE_DUMP
            << arg_traceStates
#endif // HAS_STATE_TRACE_DUMP
            ;
    }
}
//////////////////////////////////////////////////////////////////
//
bool  GeneticOptimizer::configure(CmdLineArgIter& cmdIter)
{
    DEBUGTracedFrame;
    long count;
    double   d;
    MyString s;

    if(!RandomOptimizerBASE::configure(cmdIter))
        return false;
    if(givenOption(arg_survivingSeeds)){
        evalOption(count,2,"",arg_survivingSeeds);
        survivingBestSeedCount( count );
    }
    if(givenOption(arg_seedPoolSize)){
        evalOption(count,4,"",arg_seedPoolSize);
        lengthOfSeedPool( count );
    }
    if(givenOption(arg_genArenaSize)){
        evalOption(count,8,"",arg_genArenaSize);
        sequenceCount( count );
```

```
    }
    if(givenOption(arg_seedFiles)){
        MyStringVec   vec;
        for (int i = 0 ; i < arg_seedFiles.count() ; i++) {
            vec[i] = arg_seedFiles[i];
        }
        if(collectStates( vec ))
            return false;
    }
    if(givenOption(arg_subMaxTime)){
        evalOption(d,5,"",arg_subMaxTime);
        d *= 60; // convert min -> sec
        subMaxTime( d );
    }
    if(givenOption(arg_subMaxLoops)){
        evalOption(count,10000,"",arg_subMaxLoops);
        subMaxLoops( count );
    }
    if(givenOption(arg_genMaxEqualLoops)){
        evalOption(count,5,"",arg_genMaxEqualLoops);
        genMaxLoopsWithEquivalentQ( count );
    }
    if(givenOption(arg_genQThreshold)){
        evalOption(d,0.96,"",arg_genQThreshold);
        initialQThreshold( d );
        genQThreshold( d );
    }
    if(givenOption(arg_genQThresholdDecayRate)){
        evalOption(d,0.005,"",arg_genQThresholdDecayRate);
        thresholdDecayRate( d );
    }
    if(givenOption(arg_genCrossOverThreshold)){
        d = 1.0/12.0;
        evalOption(d,d,"",arg_genCrossOverThreshold);
        crossoverThreshold( d );
    }
    if(givenOption(arg_genMaxIdleLoops)){
        evalOption(count,5,"",arg_genMaxIdleLoops);
        maxIdleLoops( count );
    }
#ifdef HAS_STATE_TRACE_DUMP
    if(givenOption(arg_traceStates)){
        MyString t = arg_traceStates;
        t.upcase();
        if(t.contains("ALL")){
            stateTraceInterval(0);
        } else if(t.contains("NONE")){
            stateTraceInterval(-1);
        } else if(t.matches(MyRXint)){
            stateTraceInterval(atoi(t.chars()));
        } else {
            ErrMessage err;
            err << error
                << "parameter for option -traceStates is wrong"
                << eom;
            return false;
        };

    }
#endif //  HAS_STATE_TRACE_DUMP
    return true;
};
////////////////////////////////////////////////////////////////
void GeneticOptimizer::restartRun(void)
{
    RandomOptimizerBASE::restartRun();
    itsLoopsWithEquivalentQ = 0;
    itsLongestCountThread   = 0;
    itsSeedLoopThread       = 0;
    itsSlidingCross         = 0.0;
    itsCrossSumme           = 0;

    genQThreshold(initialQThreshold());
    isNotSorted();
}
const char* GeneticOptimizer::crossoverOpName(void)const{
    return "C";
};
////////////////////////////////////////////////////////////

void GeneticOptimizer::sortArenaByQ(void)
{
    DEBUGTracedFrame;
    qsortByQ( 0, itsArena.count()-1);
    isSortedByQ(true);
}
/* aus Kerninghan/Ritchie "Programmieren in C" P.106  (Kap. 5.6)*/
void GeneticOptimizer::qsortByQ(int left, int right)
{
    int last;
    if(left >= right)
        return;

    swap(itsArena[left],itsArena[(left+right)/2]);
    last = left;
    for(int i=left+1; i<= right; i++)
        if( itsArena[i]->quality() > itsArena[left]-> quality()){
            swap(itsArena[++last],itsArena[i]);
        }
    swap(itsArena[left],itsArena[last]);
    qsortByQ( left,   last-1);
    qsortByQ( last+1, right);
}



void GeneticOptimizer::crossOverAll(void)
{
    DEBUGTracedFrame;
    // Rebuild the arena from #(bestSeedCount) seed states,
    // The best #(survivingBestSeedCount) seed states remain
    // unchanged after this operation.
    // The rest is a crossover-product of each of the (bestSeedCount)
    // seed states. The number of seed states may be larger
    // than the number of remaining seeds! (replacing some low q seeds)
    // requires: arena is sorted (decreasing)
    //           bestSeedCount (~4) << itsArena.count() (~10)
    //           survivingBestSeedCount (~2) <<= bestSeedCount
    // eg: crossOver(x1,x2  i,n)  => i:n;  n:i;
    //     (relative position of n:i and i:n not guaranteed!)
    // before [   1;   2;   3;   4;   5;   6;   7;   8;   9;  10]
    // after  [   1;   2; 1:2; 2:1; 1:3; 3:1; 1:4; 4:1; 2:3; 3:2]

    sortArenaByQ();
    debugTraceEvolution();

    double    bestQ=itsArena[0]->quality();

    //: src indices for position x (x>survivingBestSeedCount)
    intVec N(itsArena.count()); N.fill(-1);
    intVec I(itsArena.count()); I.fill(-1);
    int crossCount  = 0; // Anzahl der tatsächlich erfolgen Crossovers
    int wantedCross = 0; // erwünschte Anzahl der Crossovers

    // the threshold is already set if if we retry an crossover.
    double threshold = bestQ * genQThreshold();

    //: index of first target State (no of crossovers)
    //  used as index for positions in arena.
    int y;
    for(y    = itsArena.count();
        y-1 >= survivingBestSeedCount() && stateAt(y-1)->quality() < threshold;
        y--);
    ASSERT( y >= survivingBestSeedCount());
    ASSERT( y <= itsArena.count());

    // cases:
    // y > count:
    //       keine zu ersetzen
    // y < count && y >=   survivingBestSeedCount()
    //       ersetze [ y .. count-1 ]
    //
```

```
    wantedCross = itsArena.count() - y;

    ASSERT( y >= survivingBestSeedCount());
    ASSERT( y==itsArena.count() ||
            (y<itsArena.count() && stateAt(y)->quality()<threshold));
    ASSERT( y >= survivingBestSeedCount());

    if(  itsSlidingCross < crossoverThreshold() ){
#define FORCE_CROSSOVER 1

#ifdef  FORCE_CROSSOVER
        wantedCross = max(1,wantedCross);
#endif
        genQThreshold(genQThreshold()
                      +(thresholdDecayRate()*(1-genQThreshold())));
        const double maxGenQThreshold = 0.9999;
        if( genQThreshold() > maxGenQThreshold )
            genQThreshold(maxGenQThreshold);
    }
    // no assert: stateAt(wantedCross)->quality()
    if(wantedCross==0){
        debugTraceCrossOver(I,N);
    } else {
        // replace at least one state.
        // recalc y: is changed iff forced crossover !
        y = sequenceCount() - wantedCross;

        ASSERT( y >= survivingBestSeedCount());
        ASSERT( y < sequenceCount());
        // gilt nicht mehr, da ein erzwungenes CrossOver vorliegen kann!
        // ASSERT(stateAt( y )->quality() < threshold);

        // generate crossover partner schedule table.
        // put pair with higher indexes at the end to prevent
        // them from beeing overwritten before they were used.
        // (crossover target positions can be source positions
        //  in the same schedule!)
        int x = y;
        for(int i=0; i<lengthOfSeedPool(); i++){
            for(int n=i+1; x<itsArena.count() && n<lengthOfSeedPool(); n++){
                int x1 = x++;
                int x2 = x++; // q[x2] schlechter als q[x1] !
                // gilt nicht mehr, da ein erzwungenes CrossOver vorliegen kann!
                // ASSERT(stateAt( x2 )->quality() < threshold);
                if(x2<itsArena.count())
                {
                    I[x2] = n; N[x2] = i;
                }
                // gilt nicht mehr, da ein erzwungenes CrossOver vorliegen kann!
                // ASSERT(stateAt( x1 )->quality() < threshold);
                if(x1<itsArena.count())
                {
                    I[x1] = i; N[x1] = n;
                }
            }
        }

        debugTraceCrossOver(I,N);
        // perform the planned crossovers.
        // start with the target positions at count-1 to prevent overwritting
        // the sources!
        isNotSorted();
        for(int x=itsArena.count()-1; x>=y; x--){
            int x1 =   x;
            int x2 = --x; // q[x2] besser als q[x1] !
            // ASSERT(stateAt( x2 )->quality() > stateAt( x1 )->quality());
            //
            // wenn q[x1] besser ist als der threshold,
            // dann gibt es keine weiteren ersetzbaren Zustände,
            // da die Arena sortiert ist!
            // überflüssig: if( stateAt( x1 )->quality() > threshold) break;
            if( x2<y || I[x1]==N[x1] || I[x2]==N[x2] ) {
                int z = x1;
                if( I[x1] == -1 && N[x1]==-1 ){
                    z = x2;
                }
                if( I[z] != -1 && N[z]!=-1 ){
                    singleCrossOverAt(
                            z, I[z],N[z],
                            true              // generate [ I[z],N[z] ]
                            );
                    evaluateAt(x1);
                    crossCount ++;
                }
            } else {
                if( I[x1] != -1 && N[x1] !=-1 &&
                    I[x2] != -1 && N[x2] !=-1 )
                {
                    crossOverAt(x1,x2, I[x1],N[x1]);
                    evaluateAt(x1);
                    evaluateAt(x2);
                    crossCount ++;
                    crossCount ++;
                }
            }
        }
    } // if ( at least one State is below threshold )

#define SLIDING_CROSS_RATE_1

#ifdef  SLIDING_CROSS_RATE_1
    const double aufbauRate   = 0.3 ;
    const double zerfallsRate = 0.27;
    itsSlidingCross += crossCount * aufbauRate - itsSlidingCross * zerfallsRate;

#elif SLIDING_CROSS_RATE_2
    // calculate sliding crossover rate
    itsSlidingCross +=
        double(itsCrossSumme)/double(loopCount()) + crossCount;
    itsSlidingCross /= double(loopCount());
#endif
    itsCrossSumme   += crossCount;

    updateBest();
    debugTraceEvolution();
    debugTracePopulationDiff();
}

void GeneticOptimizer::singleOptimizeAt(int seqIndex)
{
    DEBUGTracedFrame;
    State::Handle sH=stateAt(seqIndex);
    State &S        = *stateAt(seqIndex);
    ActiveAccount active(S.account());

    if( isNull( subRO() ) ){
        ErrMessage err;
        err << error
            << "sub optimizer of GeneticOptimizer not set (is NULL)! "
            << "can not optimize !";
        err << eom;
        return ;
    }

    ////////////////
    // kopiere den State in alle Positionen in subRO
    // 1- setze alle Zustandsvariablen auf den Anfang zurueck
    subRO()->restartRun();// initStates ???
    // setze den best State
    subRO()->stateAt(-1,sH);
    // .. und alle anderen States
    for(int i=0; i<subRO()->sequenceCount(); i++){
        subRO()->stateAt(i,sH);
    }

    ////////////////
    //: setze alle Parameter in subRO() auf start
    subRO()->runTime(subMaxTime()); // ~ 5 min
    subRO()->maxLoop(subMaxLoops());
    // effectively disables dumpTimeInterval of subRO()
    subRO()->dumpTimeInterval(subRO()->runTime()+1);
    // effectively disables dumpLoopInterval of subRO()
    subRO()->dumpLoopInterval(subRO()->maxLoop()+1);

    ////////////////
```

```
    // optimiere
    subRO()->optimize();
    isNotSorted();

    stateAt(seqIndex,subRO()->stateAt(-1));
    debugTraceSingle(seqIndex);
}
void GeneticOptimizer::finalOptimizeBest(void)
{
    DEBUGTracedFrame;
    const int seqIndex = -1;
    State::Handle sH=  stateAt(seqIndex);
    State &S        = *stateAt(seqIndex);
    ActiveAccount active(S.account());

    if( isNull( finalRO() ) ){
        ErrMessage err;
        err << error
            << "optimizer for finalization of GeneticOptimizer not set (is NULL)! "
            << "can not optimize !";
        err << eom;
        return ;
    }

    ////////////////
    // kopiere den State in alle Positionen in finalRO
    // 1- setze alle Zustandsvariablen auf den Anfang zurueck
    finalRO()->restartRun();// initStates ???
    // setze den best State
    finalRO()->stateAt(-1,sH);
    // .. und alle anderen States
    for(int i=0; i<finalRO()->sequenceCount(); i++){
        finalRO()->stateAt(i,sH);
    }

    // effectively disables dumpTimeInterval of finalRO()
    finalRO()->dumpTimeInterval(finalRO()->runTime()+1);
    // effectively disables dumpLoopInterval of finalRO()
    finalRO()->dumpLoopInterval(finalRO()->maxLoop()+1);

    ////////////////
    // optimiere
    finalRO()->optimize();

    stateAt(seqIndex,finalRO()->stateAt(-1));
}

void GeneticOptimizer::singleOptimizeAll(void)
{
    DEBUGTracedFrame;
    long      longestLoopCount = 0;
    longVec   seedLoopCount(lengthOfSeedPool());seedLoopCount.fill(0);
    double    seedQ[lengthOfSeedPool()];
    int top = 0;
    // optimiere jeden Einzelzustand.
    // suche den laengsten Pfad ( = summe der counts der laengsten SubRO )
    // und den laengsten Pfad entlang der seeds
    for(int i=0; i<sequenceCount(); i++){
        singleOptimizeAt(i);
        longestLoopCount = max(longestLoopCount, subRO()->loopCount());
        if(top<lengthOfSeedPool() || seedQ[top-1] < stateAt(i)->quality()){
            if(top<lengthOfSeedPool()){
                seedQ[top]           = stateAt(i)->quality();
                seedLoopCount[top++] = subRO()->loopCount();
            } else {
                seedQ[top-1]         = stateAt(i)->quality();
                seedLoopCount[top-1] = subRO()->loopCount();
            }
            int x = top-1;
            while( x>0 && seedQ[x-1] < seedQ[x]){
                swap(seedQ[x-1] ,seedQ[x]);
                swap(seedLoopCount[x-1] ,seedLoopCount[x]);
            }
        }
    }
    itsLongestCountThread += longestLoopCount;
    long count = 0;
    for(int i=0; i<lengthOfSeedPool(); i++){
        count = max(count,seedLoopCount[i]);
    }
    // der laengste count eines Seeds
    itsSeedLoopThread +=  count;
    // werte info aus
    debugTraceEvolution();
    debugTracePopulationDiff();
}
bool GeneticOptimizer::updateBest(void)
{
    DEBUGTracedFrame;
    const double epsilon = 1E-8;
    int currBest=0;
    if(!isSortedByQ()){
        for(int i=0; i<sequenceCount(); i++){
            if(stateAt( i )->quality()-stateAt(currBest)->quality() > epsilon)
            {
                currBest = i;
            }
        }
    }
    itsIsImproved   = false;
    double delta = stateAt( currBest )->quality() - stateAt( -1 )->quality();

    if(delta >= 0.0)
    {
        if( delta  > epsilon ){
            // der neue Beste in der Arena ist besser als der global Beste.
            debugTracePopulationStat();

            stateAt(-1,stateAt(currBest));
            itsLastQ = stateAt(currBest) ->quality();
            itsIsImproved   = true;
            itsLastQChanged = loopCount();
        }
        return true;
    }

    return false;
}
void GeneticOptimizer::evaluateArena(void)
{
    DEBUGTracedFrame;
    sortArenaByQ();
    RandomOptimizerBASE::evaluateArena();
    // find and record best. (first pos, is sorted!)
    if( !updateBest() && !itsIsImproved ){
        // der neue Beste ist schlecher als der alte Beste.
        // (Wie kann sowas passieren? Alle SubOptis sollten
        //  doch mindestens den Besten der letzten Runde beibehalten?!)
        //  Geschieht wenn ein alter Zustand geladen wurde und die
        //  Subzustände leer sind!
        //  mache den algo robust, fixe den Fehler.
        //
        // verschiebe state[0 .. survivingBestSeedCount-1] um eins
        // nach hinten, dann kopiere den Besten in die Arena auf
        // Position 0
        for(int x = lengthOfSeedPool()-2; x>=0; x--){
            swap(itsArena[x], itsArena[x+1]) ;
        }
        // zwischenzustand:
        // [ 1, 0, 2, 3, 4...] survivingBestSeedCount=2
        // [ 3, 0, 1, 2, 4...] survivingBestSeedCount=4
        stateAt(0,stateAt(-1));
        // [ -1, 0, 2, 3, 4...]
        // die arena bleibt nach Q sortiert!
    }

    if(lastQChanged() < loopCount() - maxIdleLoops()){
        genQThreshold(genQThreshold()
                      + (thresholdDecayRate() * (1.0 - genQThreshold())));
    }
}


////////////////////////////////////////////////////////////////////
```

```
bool GeneticOptimizer::abbruchbedingung(void)
{
    DEBUGTracedFrame;
    const double epsilon = 1E-8;

    // zeit, loops, q besser als maxQ if maxQ > 0
    if(RandomOptimizerBASE::abbruchbedingung())
        return true;
    // beende das Program nach X aufeinander folgenden Loops in denen
    // die Qualitaet *aller* States immer gleich dem besten State ist.
    if( abs(itsArena[0]->quality() - itsBest->quality()) <epsilon ){
        int countEqualQ = 0;
        double firstQ =itsArena[0]->quality();
        for(int i=0; i<sequenceCount(); i++){
            double currQ =itsArena[i]->quality();
            if(abs(currQ-firstQ)<epsilon){
                countEqualQ ++ ;
            }
        }
        if( countEqualQ == sequenceCount() ){
            itsLoopsWithEquivalentQ ++;
        } else {
            itsLoopsWithEquivalentQ = 0;
        }
        if(itsLoopsWithEquivalentQ>itsMaxLoopsWithEquivalentQ){
            return true;
        }
    }

    return false;
}

void GeneticOptimizer::optimize(void)
{
    DEBUGTracedFrame;
    itsEndTime = time(NULL) + runTime();
    while(true){
        singleOptimizeAll();
        evaluateArena();
        if(abbruchbedingung())              break;
        crossOverAll();
    }
    debugTraceEvolution();
    debugTraceFinalPopulationStat();

    finalOptimizeBest();
    debugTraceEvolution();
    debugFinal();
}
////////////////////////////////////////////////////////////////////
// TRACING
// vor dem sortieren
// EVOLUTION 8 [ (323.5:A140:L120),(323.5:A140:L120),(323.5:A140:L120) ] bestSeed( 2,5,6,7)
// DEVELOP   8 [ ((323.5:A140:L120) -> 37.5:A140:L121)), ...
////////////////////////////////////////////////////////////////////
GeneticOptimizer& GeneticOptimizer::debugTraceSingle(int seqIndex)
{
    DEBUGTracedFrame;
    const char sep = ';';
    (debugStream()) << "TRACE_GEN_SINGLE"<<sep<< (seqIndex+1) << sep;
    (debugStream()) << '('
                    <<stateAt(seqIndex)->quality()<<':'
                    <<stateAt(seqIndex)->usedCount()<<':'
                    <<stateAt(seqIndex)->usedLinks()<<')';
    (debugStream())
        << " loops=" << subRO()->loopCount();
    (debugStream()) << endl;
    return *this;
}

////////////////////////////////////////////////////////////////////
/* aus Kerninghan/Ritchie "Programmieren in C" P.106  (Kap. 5.6)*/
void sortIndexed(intVec &pos ,Array<State::Handle>& s, int left, int right)
{
    ASSERT(left>=0);
    // ASSERT(left<=right); wrong, teil des algo !
    ASSERT(right<pos.count());
    ASSERT(pos.count()<=s.count());

    int last;
    if(left >= right)
        return;

    swap(pos[left],pos[(left+right)/2]);
    last = left;
    for(int i=left+1; i<= right; i++)
        if( s[pos[i]]->quality() > s[pos[left]]->quality() ){
            swap(pos[++last],pos[i]);
        }
    swap(pos[left],pos[last]);
    sortIndexed(pos,s, left,   last-1);
    sortIndexed(pos,s, last+1, right);
}
GeneticOptimizer& GeneticOptimizer::debugTraceEvolution(void)
{
    DEBUGTracedFrame;

    double best=0.0, worst=itsArena[0]->quality(), mid=0.0;
    // finde die vier besten, best first

    intVec bestVec(sequenceCount());
    for(int i=0; i<sequenceCount(); i++){
        bestVec[i] = i;
    }
    sortIndexed( bestVec,itsArena, 0, bestVec.count()-1  );
    for(int i=0; i<sequenceCount(); i++){
        mid += stateAt(i)->quality();
    }
    best   = stateAt( bestVec[0] )->quality() ;
    worst  = stateAt( bestVec[sequenceCount()-1] )->quality()  ;
    mid /=sequenceCount();

    {
        itsLastTraceCount = itsLoopCount;
        const char sep = ';';
        (debugStream()) << "TRACE_GEN_EVOL"<<sep<< itsLoopCount << sep;
        (debugStream()) <<'[';
        for(int i=0; i<sequenceCount(); i++){
            if(i>0)(debugStream()) <<',';
            (debugStream()) << '('
                            <<stateAt(i)->quality()<<':'
                            <<stateAt(i)->usedCount()<<':'
                            <<stateAt(i)->usedLinks()<<')';
        }
        (debugStream()) <<"] best(";
        for(int n = 0; n<lengthOfSeedPool(); n++){
            if(n>0)(debugStream()) <<',';
            (debugStream()) << bestVec[n]+1;
        }
        (debugStream()) <<')';
        (debugStream()) << sep
                        << itsBest->quality() << sep;
        (debugStream()) << best << sep << (mid) << sep << worst << sep;
        (debugStream()) << endl;
    }
    return *this;
};
////////////////////////////////////////////////////////////////////
GeneticOptimizer& GeneticOptimizer::debugTraceCrossOver(
        const intVec &N,const intVec &I)
{
    DEBUGTracedFrame;
    const char sep = ';';
    (debugStream()) << "TRACE_GEN_CROSSOVER"<<sep<< itsLoopCount << sep;
    (debugStream()) <<'[';
    for(int i=0; i<sequenceCount(); i++){
        if(i>0)(debugStream()) <<',';
        if (I[i]==-1 && N[i] == -1)
            (debugStream()) << '('<<(i+1)<<':'<<(i+1)<<") ";
        else
            (debugStream()) << '('<<(I[i]+1)<<':'<<(N[i]+1)<<") ";
    }
    (debugStream()) <<']';
    (debugStream()) << endl;
```

```
    return *this;
};
////////////////////////////////////////////////////////////////
// berechne den Abstand der Zustaende zum Besten.
GeneticOptimizer& GeneticOptimizer::debugTracePopulationDiff(void)
{
    DEBUGTracedFrame;
    longVec cxDiff(sequenceCount());

    const char sep = ';';
    (debugStream()) << "TRACE_GEN_POPULATION_DIFF"<<sep<< itsLoopCount << sep;
    (debugStream()) <<'[';
    for(int i=0; i<sequenceCount(); i++){
        if(i>0)(debugStream()) <<',';
        StateDiff diff(stateAt(-1),stateAt(i));
        diff.reduceSwaps().finalizePath();
        (debugStream())<< diff.atomDistance();
        cxDiff[i] = diff.cxDistance();
    }
    (debugStream()) <<']';
    (debugStream()) <<'[';
    for(int i=0; i<sequenceCount(); i++){
        if(i>0)(debugStream()) <<',';
        (debugStream())<< cxDiff[i];
    }
    (debugStream()) <<']';
    (debugStream()) << endl;

    return *this;
};
////////////////////////////////////////////////////////////////
GeneticOptimizer& GeneticOptimizer::debugFinal(void)
{
//////////
    DEBUGTracedFrame;
    const char sep = ';';
    debugTraceFinalPopulationStat();

    (debugStream()) << "TRACE_GEN_FINAL"<<sep<< itsLoopCount << sep;
    (debugStream()) <<'[';
    (debugStream())
        <<" longestCountThread="<< longestCountThread()
        <<" seedLoopThread="    << seedLoopThread();
    (debugStream()) <<']';
    (debugStream()) << endl;

    return *this;
};
#ifdef HAS_STATE_TRACE_DUMP
////////////////////////////////////////////////////////////////
GeneticOptimizer& GeneticOptimizer::debugStateTrace(void)
{
//////////
    DEBUGTracedFrame;
    const char sep = ';';

    (debugStream()) << "TRACE_STATE_DUMP"<<sep<< itsLoopCount << sep;
    (debugStream()) <<
    (debugStream()) << endl;

    return *this;
};
#endif //  HAS_STATE_TRACE_DUMP


////////////////////////////////////////////////////////////////
bool GeneticOptimizer::collectStates( const MyStringVec& vec ){
    DEBUGTracedFrame;
    enum{ ok = 0 , fail = 1 };

    RandomOptimizerBASE lOpti;
    lOpti.useIndependent(usedIndependent());

    for(int i=0; i< min(vec.count(),sequenceCount()); i++){
        if(ok != lOpti.loadState(vec[i],peakPool(),peakNet())){
            return fail;
        };

        stateAt(i,lOpti.stateAt(-1));
    }
    // wenn mehr Files angegeben wurden, als Plaetze vorhanden sind
    // verfallen die restlichen Namen.
    return ok;
};
////////////////////////////////////////////////////////////////
bool GeneticOptimizer::loadConfig(istream & is)
{
    DEBUGTracedFrame;
    enum{ ok = 0 , fail = 1 };
    int errPos=0;
    AssertHook noStackDump(NULL);

    ReadBlock  block2(is,"CONFIG");
    ReadBlock  block3(is,"ACTIONCONFIG");
    if(!block2.open())       ERROR_RECOVER;
    if(!block2.skipToEnd())  ERROR_RECOVER;
    if(!block3.skipBlock())  ERROR_RECOVER;

    return ok;
  error_recover:
    ErrMessage err;
    err << error
        << " parsing optimizer seed failed. (code =  " << errPos
        << " at line " << streampos2lineNr(is,is.tellg())
        << ")"  << eom;
    return fail;
}
void GeneticOptimizer::dumpConfig(ostream & os)
{
    DEBUGTracedFrame;
    enum{ ok = 0 , fail = 1 };
    {WriteBlock  block2(os,"CONFIG");
    if(block2){
        os << indent << "VERSION            2" <<endl;
        os << indent << "DUMPTIMEINTERVAL   "<<itsDumpTimeInterval<<endl;
        os << indent << "LOOPCOUNT          "<<  itsLoopCount<< endl;
        os << indent << "MAXLOOPCOUNT       "<< itsMaxLoopCount<< endl;
        os << indent << "LASTQ              " <<  itsLastQ<< endl;
        os << indent << "LASTQCHANGED       " <<  itsLastQChanged<< endl;
    }
    }
    {WriteBlock  block3(os,"ACTIONCONFIG");
    if(block3){
    }
    }
}

////////////////////////////////////////////////////////////////
SubRandomOptimizer::~SubRandomOptimizer(void)
{
    /* EMPTY */
};
SubRandomOptimizer::SubRandomOptimizer(void)
:  RandomOptimizer()
{
    /* EMPTY */
};

void SubRandomOptimizer::dumpState(void)
{
    // SubOptimizers don't dump!
};
////////////////////////////////////////////////////////////////
static CmdArgInt       arg_subTraceInterval('s',"subTraceInterval","loops", "number loops in
   subprocess" , CmdArg::isOPTVALREQ);
static CmdArgInt       arg_subMaxIdleLoops('s',"subMaxIdleLoops","min", "time in subprocess" ,
   CmdArg::isOPTVALREQ);
////////////////////////////////////////////////////////////////
//
void SubRandomOptimizer::registerOptions(CmdLine& cmd){
    DEBUGTracedFrame;
    static bool done = false;
    RandomOptimizerBASE::registerOptions(cmd);
```

```
    if(!done){
        done = true;
        cmd << arg_subTraceInterval
            << arg_subMaxIdleLoops ;
    }
}
////////////////////////////////////////////////////////////////////
//
void SubRandomOptimizer::restartRun(void)
{
    RandomOptimizer::restartRun();
    // disable population traces
    itsPopulationDumpInterval = -1;
};

////////////////////////////////////////////////////////////////////
//
bool  SubRandomOptimizer::configure(CmdLineArgIter& cmdIter){
    DEBUGTracedFrame;
    long count;

    if(!RandomOptimizer::configure(cmdIter))
        return false;
    if(givenOption(arg_subTraceInterval)){
        evalOption(count,100,"",arg_subTraceInterval);
        maxTraceInterval( count );
    }
    if(givenOption(arg_subMaxIdleLoops)){
        evalOption(count,1000,"",arg_subMaxIdleLoops);
        maxIdleLoops( count );
    }
    return true;
}
SubRandomOptimizer& SubRandomOptimizer::debugParams(void)
{
    // do nothing, inhibits OPTI_PARAM und OPTI_DEFINE dumps!
    return *this;
}
```

N.17 Quelltext: GeneticOptimizerFactory.h

```
#ifndef __GENETIC_OPTIMIZER_FACTORY__H__
#define __GENETIC_OPTIMIZER_FACTORY__H__

class MyString;
class GeneticOptimizer;

class GeneticOptimizerFactory {
public:
    inline ~GeneticOptimizerFactory(){};
    inline GeneticOptimizerFactory(){};
    inline GeneticOptimizerFactory(const GeneticOptimizerFactory &){};
    inline GeneticOptimizerFactory& operator=(const GeneticOptimizerFactory&){ return *this;};

    static GeneticOptimizer* make(const MyString& name);
private:
};

#endif // __GENETIC_OPTIMIZER_FACTORY__H__
```

N.18 Quelltext: GeneticOptimizerFactory.cc

```
#include "MyString.h"

#include "GeneticOptimizerFactory.h"

#include "GeneticRandomOptimizer.h"

#include "GeneticRandomOptimizer_C.h"

#include "GeneticRandomOptimizer_Gs1p1.h"
#include "GeneticRandomOptimizer_Gs1p2.h"
#include "GeneticRandomOptimizer_Gs1p3.h"

#include "GeneticRandomOptimizer_Gs2p1.h"
#include "GeneticRandomOptimizer_Gs2p2.h"
#include "GeneticRandomOptimizer_Gs2p3.h"

#include "GeneticRandomOptimizer_Gs3p1.h"
#include "GeneticRandomOptimizer_Gs3p2.h"
#include "GeneticRandomOptimizer_Gs3p3.h"

#include "GeneticRandomOptimizer_elitaer.h"

GeneticOptimizer*  GeneticOptimizerFactory::make(const MyString& name)
{
    GeneticOptimizer* opt=NULL;
    if( name == "" && name == "default" ){
        opt = new GeneticOptimizer();

    } else if( name == "C" ){
        opt = new GeneticOptimizer_C();

    } else if( name == "Gs1p1" ){
        opt = new GeneticOptimizer_Gs1p1();
    } else if( name == "Gs1p2" ){
        opt = new GeneticOptimizer_Gs1p2();
    } else if( name == "Gs1p3" ){
        opt = new GeneticOptimizer_Gs1p3();

    } else if( name == "Gs2p1" ){
        opt = new GeneticOptimizer_Gs2p1();
    } else if( name == "Gs2p2" ){
        opt = new GeneticOptimizer_Gs2p2();
    } else if( name == "Gs2p3" ){
        opt = new GeneticOptimizer_Gs2p3();

    } else if( name == "Gs3p1" ){
        opt = new GeneticOptimizer_Gs3p1();
    } else if( name == "Gs32p2" ){
        opt = new GeneticOptimizer_Gs3p2();
    } else if( name == "Gs3p3" ){
        opt = new GeneticOptimizer_Gs3p3();

    } else if( name == "elitaer" ){
        opt = new GeneticOptimizer_elitaer();

    } else {
        opt = new GeneticOptimizer();
    }
    return opt;
};
```

N.19 Quelltext: GeneticRandomOptimizer_C.h

```
#ifndef __GeneticRandomOptimizer_C_H__
#define __GeneticRandomOptimizer_C_H__  1
//
//
// $Id: GeneticRandomOptimizer_C.h,v 1.3 2000/05/29 16:23:04 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
```

```
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "Cache.h"
#include "Protein.h"
#include "PeakPool.h"
#include "PeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizerBASE.h"
#include "RandomOptimizer.h"
#include "GeneticRandomOptimizer.h"

//////////////////////////////////////////////////////////////////

class StateIndependent;
class State;
class LinkBuilder;
class RandomOptimizerBASE;
class RandomOptimizer;

//////////////////////////////////////////////////////////////
class GeneticOptimizer;
class GeneticOptimizer_C;
class GeneticOptimizer_C : public GeneticOptimizer
{
public:
    typedef GeneticOptimizer_C            Self;
    typedef IMngObject< RandomOptimizerBASE > Handle;
    typedef SubRandomOptimizer SUBOptimizer;

    virtual ~GeneticOptimizer_C(void);
    GeneticOptimizer_C(void);

     //: generates two new States by merging two other States
    void     crossOverAt(
            int destIndex1,int destIndex2, // dest
            int srcIndex1 ,int srcIndex2   // src
            );
    //:  generates *one* of two new States by merging two other States
    //   kopiere Stuecke aus srcS1 und srcS2 nach destS1
    //   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
    void     singleCrossOverAt(
            int destIndex1,
            int srcIndex1 ,int srcIndex2,
            bool generateFirst   // true generate dest1 , false generates dest2
            );
    const char* crossoverOpName(void)const;
protected:
};

#endif // __GeneticRandomOptimizer_C_H__
```

N.20 Quelltext: GeneticRandomOptimizer_C.cc

```
#include "GeneticRandomOptimizer_C.h"
#include "StateMetrics.h"

//////////////////////////////////////////////////////////////
#include "RandomOptimizer.debug.h"

//////////////////////////////////////////////////////////////////////
//
GeneticOptimizer_C::~GeneticOptimizer_C(void)
{
    DEBUGTracedFrame;
    /* EMPTY */
}

GeneticOptimizer_C::GeneticOptimizer_C(void)
:  GeneticOptimizer()
{
    DEBUGTracedFrame;
}
//////////////////////////////////////////////////////
//
const char* GeneticOptimizer_C::crossoverOpName(void)const{
    return "C";
};


//////////////////////////////////////////////////////////////////////////
//:  generates two new States by merging two other States
//   kopiere Stuecke aus srcS1 und srcS2 nach destS1 und destS2
//   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
void     GeneticOptimizer_C::crossOverAt(
        int destIndex1,int destIndex2, // dest  [ 0 .. sequenceCount()-1]
        int srcIndex1  ,int srcIndex2   // src   [-1 .. sequenceCount()-1]
        )
{
    DEBUGTracedFrame;
    RandomOptimizerBASE::crossOverAt( destIndex1, destIndex2, srcIndex1, srcIndex2);
}


//////////////////////////////////////////////////////////////////////////
//:  generates one of two new States by merging two other States
//   kopiere Stuecke aus srcS1 und srcS2 nach destS1 und destS2
//   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
/////
// generateFirst:
//    true generates dest1 , false generates dest2
void     GeneticOptimizer_C::singleCrossOverAt(
        int destIndex1,
        int srcIndex1 ,int srcIndex2,
        bool generateFirst  // true generate dest1 , false generates dest2
        )
{
    DEBUGTracedFrame;
    RandomOptimizerBASE::singleCrossOverAt(
            destIndex1,
            srcIndex1 , srcIndex2,
            generateFirst
            );
}
```

N.21 Quelltext: GeneticRandomOptimizer_OPE−RATOR.cc

```
// OPTIMIZER is one of GeneticOptimizer_Gs2p2 ... _C
#ifndef OPTIMIZER
#warn OPTIMIZER not defined !
#endif

//////////////////////////////////////////////////////////////////////////
//:  generates two new States by merging two other States
//   wendet Teile der Differenz von (schlecht->gut) auf die
//   Zielzustände an.
//
void     OPTIMIZER::crossOverAt(
        int destIndex1,int destIndex2, // dest  [ 0 .. sequenceCount()-1]
        int srcIndex1 ,int srcIndex2   // src   [-1 .. sequenceCount()-1]
        )
```

```
{
    DEBUGTracedFrame;
    ASSERT( destIndex1 != srcIndex1 );
    ASSERT( destIndex1 != srcIndex2 );
    ASSERT( destIndex2 != srcIndex1 );
    ASSERT( destIndex2 != srcIndex2 );
    State &destS1 = *stateAt(destIndex1);
    State &destS2 = *stateAt(destIndex2);
    State &srcS1  = *stateAt(srcIndex1);
    State &srcS2  = *stateAt(srcIndex2);
#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "SEQUENCES["
         << destIndex1+1 <<", "<< destIndex2+1 <<", "
         << srcIndex1+1 <<", "<< srcIndex2+1
         << "] singleCrossOverAt             Q(src1)="
         << srcS1.quality()<<"  Q(src2)="<< srcS2.quality()<< endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q
    CheckLockState checkState1s( &srcS1 );
    CheckLockState checkState2s( &srcS2 );
    CheckLockState checkState1d( &destS1 );
    CheckLockState checkState2d( &destS2 );

#if defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1) ||
   defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1a)
    StateDiff diff1(stateAt(destIndex1),stateAt(srcIndex1));
    StateDiff diff2(stateAt(destIndex2),stateAt(srcIndex2));
    diff1.finalizePath();
    diff2.finalizePath();
#if defined(GUIDED_CROSSOVER_SELECTION_VARIANTE_3)
    State& diffStart1 = destS1;
    State& diffStart2 = destS2;
#endif
#elif defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_2)
    StateDiff diff1(stateAt(srcIndex1),stateAt(srcIndex2));
    StateDiff diff2(stateAt(srcIndex2),stateAt(srcIndex1));
    diff1.finalizePath();
    diff2.finalizePath();
#if defined(GUIDED_CROSSOVER_SELECTION_VARIANTE_3)
    State& diffStart1 = srcS1;
    State& diffStart2 = srcS2;
#endif
#endif

    // wähle eine Teilmenge der States aus.
    //   * wenn die Operation aus einem Paar (SUB+ADD) besteht, wird die ADD
    //     Operation kopiert, sonst wird die SUB Operation kopiert.
    //   ANNAHME: ADD+SUB treten in Paaren auf, (SUB(i),ADD(i+1))
    //   ANNAHME: diff ist nach AS-Positionen sortiert.
#ifdef GUIDED_CROSSOVER_SELECTION_VARIANTE_1
    //algo: wählt itsGuidedCrossOverSelectionPct% Aktionen zufällig aus.
    //   - kopiert maximal itsGuidedCrossOverSelectionPct% Aktionen.
    //   - die Aktionen hängen nicht zusammen.
    {StateDiff res1; res1.start(diff1.start()).end(diff1.end());
    for(int t = 0; t<diff1.top(); ++t){
        if( itsGuidedCrossOverSelectionPct >= getRandomDouble() ){
            if(diff1.at(t).isLinkOp()){
                // skip links
            } else if(diff1.at(t).operation() == action_rec::op_add){
                res1.push(diff1.at(t));
            } else if(diff1.at(t).operation() == action_rec::op_sub ){
                if(t<diff1.top()
                   && diff1.at(t+1).operation()==action_rec::op_add
                   && diff1.at(t+1).cxID() == diff1.at(t).cxID())
                {
                    res1.push(diff1.at(t+1));
                    t++;
                } else {
                    res1.push(diff1.at(t));
                }
            }
        }
    }
    swap(diff1,res1);
    StateDiff res2; res2.start(diff2.start()).end(diff2.end());
    for(int t = 0; t<diff2.top(); ++t){
        if( itsGuidedCrossOverSelectionPct >= getRandomDouble() ){
            if(diff2.at(t).isLinkOp()){
                // skip links
            } else if(diff2.at(t).operation() == action_rec::op_add){
                res2.push(diff2.at(t));
            } else if(diff2.at(t).operation() == action_rec::op_sub ){
                if(t<diff2.top()
                   && diff2.at(t+1).operation()==action_rec::op_add
                   && diff2.at(t+1).cxID() == diff2.at(t).cxID())
                {
                    res2.push(diff2.at(t+1));
                    t++;
                } else {
                    res2.push(diff2.at(t));
                }
            }
        }
    }
    swap(diff2,res2);
    }

#elif  defined(GUIDED_CROSSOVER_SELECTION_VARIANTE_2)
    // algo:
    //   - kopiert mindestens eine Aktion.
    //   - kopiert maximal itsGuidedCrossOverSelectionPct% Aktionen.
    //   - kopiert jeweils #itsGuidedCrossOverChunkSize zusammenhängende
    //     Aktionen. (so soll die Nachbarschaftsbeziehung erhalten bleiben.)
    //
    const int itsGuidedCrossOverChunkSize = 4;
    intVec used;
    int tCount;
    {
    if(diff1.top()>0){
        StateDiff res1; res1.start(diff1.start()).end(diff1.end());
        used.resize(diff1.top()); used.fill(0);
        tCount = int(diff1.top() * itsGuidedCrossOverSelectionPct);
        tCount = max(1,tCount);
        int eBreak = used.count() * 2;
        while(tCount>0 && eBreak > 0){
            eBreak --;
            int t = longInRange0(diff1.top()-1);
            int sz = itsGuidedCrossOverChunkSize-1;
            while(t<diff1.top() && tCount > 0 && used[t] != 1 && sz>0){
                // suche die erste unverbrauchte Aktion
                while(t<diff1.top() &&
                      !diff1.at(t).isASOp() && used[t] != 1){
                    // ignoriere Links und bearbeite die nächsten Aktion
                    // wenn sie noch nicht benutzt wurde
                    used[t++] = 1;
                }
                if(t == diff1.top() || used[t] == 1) continue;

                if(diff1.at(t).operation() == action_rec::op_sub ){
                    // wenn dies eine einzelne SUB Aktion ist, verwende diese.
                    // wenn es der erste Teil eines SUB,ADD Paares ist,
                    // gehe zum ADD Teil.
                    if(diff1.at(t+1).operation()==action_rec::op_add
                       && diff1.at(t+1).cxID() == diff1.at(t).cxID())
                    {
                        // teil eines Paares, markieren und zum ADD Teil
                        // weiter gehen.
                        used[t++] = 1;
                    } else {
                        // eine einzelne SUB Operation.
                        // wird unten in if(){} kopiert.
                    }
                }
                //
                if(diff1.at(t).operation() == action_rec::op_sub ||
                   diff1.at(t).operation() == action_rec::op_add){

                    res1.push(diff1.at(t));
                    used[t] = 1;
                    tCount--; sz--;
                }
            }
        }
        swap(diff1,res1);
    }
    if(diff2.top()>0){
```

```
        StateDiff res2; res2.start(diff2.start()).end(diff2.end());
        used.resize(diff2.top()); used.fill(0);
        tCount = int(diff2.top() * itsGuidedCrossOverSelectionPct);
        tCount = max(1,tCount);
        int eBreak = used.count() * 2;
        while(tCount>0 && eBreak > 0){
            eBreak --;
            int t = longInRange0(diff2.top()-1);
            int sz = itsGuidedCrossOverChunkSize-1;
            while(t<diff2.top() && tCount > 0 && used[t] != 1 && sz>0){
                // suche die erste unverbrauchte Aktion
                while(t<diff2.top() &&
                      !diff2.at(t).isASOp() && used[t] != 1){
                    // ignoriere links und bearbeite die nächsten Aktion
                    // wenn sie noch nicht benutzt wurde
                    used[t++] = 1;
                }
                if(t == diff2.top() || used[t] == 1) continue;

                if(diff2.at(t).operation() == action_rec::op_sub ){
                    // wenn dies eine einzelne SUB Aktion ist, verwende diese.
                    // wenn es der erste Teil eines SUB,ADD Paares ist,
                    // gehe zum ADD Teil.
                    if(diff2.at(t+1).operation()==action_rec::op_add
                       && diff2.at(t+1).cxID() == diff2.at(t).cxID())
                    {
                        // teil eines Paares, markieren und zum ADD Teil
                        // weiter gehen.
                        used[t++] = 1;
                    } else {
                        // eine einzelne SUB Operation.
                        // wird unten in if(){} kopiert.
                    }
                }
                //
                if(diff2.at(t).operation() == action_rec::op_sub ||
                   diff2.at(t).operation() == action_rec::op_add){

                    res2.push(diff2.at(t));
                    used[t] = 1;
                    tCount--; sz--;
                }
            }
        }
        swap(diff2,res2);
    }
    }
#elif  defined(GUIDED_CROSSOVER_SELECTION_VARIANTE_3)
    // algo:
    //   - kopiert mindestens eine Aktion.
    //   - kopiert maximal itsGuidedCrossOverSelectionPct% Aktionen.
    //   - kopiert jeweils #itsGuidedCrossOverChunkSize zusammenhängende
    //     Aktionen. (so soll die Nachbarschaftsbeziehung erhalten bleiben.)
    //   - kopiere nur, wenn as[i] und as[i+1] verbunden sind.
    // Annahme: diff ist nach AS-Positionen sortiert.
    //
    const int itsGuidedCrossOverChunkSize = 4;
    intVec used;
    int tCount;
    {
    if(diff1.top()>0){
        StateDiff res1; res1.start(diff1.start()).end(diff1.end());
        used.resize(diff1.top()); used.fill(0);
        tCount = int(diff1.top() * itsGuidedCrossOverSelectionPct);
        tCount = max(1,tCount);
        int eBreak = used.count() * 2;
        while(tCount>0 && eBreak > 0){
            eBreak --;
            int t = longInRange0(diff1.top()-1);
            int sz = itsGuidedCrossOverChunkSize-1;
            while(t<diff1.top() && tCount > 0 && used[t] != 1  && sz>0){
                ////////////////////////
                // suche die erste unverbrauchte Aktion
                while(t<diff1.top() &&
                      !diff1.at(t).isASOp() && used[t] != 1){
                    // ignoriere Links und bearbeite die nächsten Aktion
                    // wenn sie noch nicht benutzt wurde
                    used[t++] = 1;
                }
                // at(t) ist jetzt entweder
                //     hinter der letzten Aktion     -> neuen Start wählen
                //     auf einem benutzen Element    -> neuen Start wählen
                // oder eine AS operation (SUB,ADD,...)
                // (neuen Start wählen == continue da die abbruchkriterien
                //   teil der schleife sind.)
                if(t == diff1.top() || used[t] == 1) continue;

                // neuen Start wählen wenn
                //   cx an pos (at(t).position()) mit dem Nachfolger
                //   verbunden ist.
                // oder kein Nachfolger vorhanden sein kann.
                if(diff1.at(t).position()<diffStart1.linkCount()
                    && !isNull(diffStart1.linkAt(diff1.at(t).position()))
                   )
                {
                    // erzeugt ein künstliches abbruch kriterium, das in der
                    // While() schleife sowieso getestet werden muss.
                    t = diff1.top();
                    continue;
                }

                if(diff1.at(t).operation() == action_rec::op_sub ){
                    // wenn dies eine einzelne SUB Aktion ist, verwende diese.
                    // wenn es der erste Teil eines SUB,ADD Paares ist,
                    // gehe zum ADD Teil.
                    if(diff1.at(t+1).operation()==action_rec::op_add
                       && diff1.at(t+1).cxID() == diff1.at(t).cxID())
                    {
                        // teil eines Paares, markieren und zum ADD Teil
                        // weiter gehen.
                        used[t++] = 1;
                    } else {
                        // eine einzelne SUB Operation.
                        // wird unten in if(){} kopiert.
                    }
                }
                //
                if(diff1.at(t).operation() == action_rec::op_sub ||
                   diff1.at(t).operation() == action_rec::op_add){

                    res1.push(diff1.at(t));
                    used[t] = 1;
                    tCount--; sz--;
                }
                ////////////////////////
            }
        }
        swap(diff1,res1);
    }
    if(diff2.top()>0){
        StateDiff res2; res2.start(diff2.start()).end(diff2.end());
        used.resize(diff2.top()); used.fill(0);
        tCount = int(diff2.top() * itsGuidedCrossOverSelectionPct);
        tCount = max(1,tCount);
        int eBreak = used.count() * 2;
        while(tCount>0 && eBreak > 0){
            eBreak --;
            int t = longInRange0(diff2.top()-1);
            int sz = itsGuidedCrossOverChunkSize-1;

            while(t<diff2.top() && tCount > 0 && used[t] != 1  && sz>0){
                ////////////////////////
                // suche die erste unverbrauchte Aktion
                while(t<diff2.top() &&
                      !diff2.at(t).isASOp() && used[t] != 1){
                    // ignoriere Links und bearbeite die nächsten Aktion
                    // wenn sie noch nicht benutzt wurde
                    used[t++] = 1;
                }
                // at(t) ist jetzt entweder
                //     hinter der letzten Aktion     -> neuen Start wählen
                //     auf einem benutzen Element    -> neuen Start wählen
                // oder eine AS operation (SUB,ADD,...)
                // (neuen Start wählen == continue da die abbruchkriterien
                //   teil der schleife sind.)
```

```
                if(t == diff2.top() || used[t] == 1) continue;

                // neuen Start wählen wenn
                //   cx an pos (at(t).position()) mit dem Nachfolger
                //   verbunden ist.
                // oder kein Nachfolger vorhanden sein kann.
                if(diff2.at(t).position()<diffStart2.linkCount()
                    && !isNull(diffStart2.linkAt(diff2.at(t).position()))
                  )
                {
                    // erzeugt ein künstliches abbruch kriterium, das in der
                    // While() schleife sowieso getestet werden muss.
                    t = diff2.top();
                    continue;
                }

                if(diff2.at(t).operation() == action_rec::op_sub ){
                    // wenn dies eine einzelne SUB Aktion ist, verwende diese.
                    // wenn es der erste Teil eines SUB,ADD Paares ist,
                    // gehe zum ADD Teil.
                    if(diff2.at(t+1).operation()==action_rec::op_add
                       && diff2.at(t+1).cxID() == diff2.at(t).cxID())
                    {
                        // teil eines Paares, markieren und zum ADD Teil
                        // weiter gehen.
                        used[t++] = 1;
                    } else {
                        // eine einzelne SUB Operation.
                        // wird unten in if(){} kopiert.
                    }
                }
                //
                if(diff2.at(t).operation() == action_rec::op_sub ||
                   diff2.at(t).operation() == action_rec::op_add){

                    res2.push(diff2.at(t));
                    used[t] = 1;
                    tCount--;  sz--;
                }
               ////////////////////////
              }
            }
            swap(diff2,res2);
        }
        }
#endif // SELECTION

#if  defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1)
    int max1  = diff1.top();
    int max2  = diff2.top();
    int half1 = max(0,max1/2-1);
    int half2 = max(0,max2/2-1);
#endif

#if defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1)
    // diff1 == (d1->s1)
    // diff2 == (d2->s2)
    // wende die untere Hälfte von diff1 auf  dest1 an
    // wende die obere  Hälfte von diff2 auf  dest1 an
    {ActiveAccount active(destS1.account());
    applyDiffRangeAt( destIndex1, diff1, 0 , max(0,half1-1));
    applyDiffRangeAt( destIndex1, diff2, half2,
                      diff2.top()-half2);
    }
    // wende die untere Hälfte von diff2 auf  dest2 an
    // wende die obere  Hälfte von diff1 auf  dest2 an
    {ActiveAccount active(destS2.account());
    applyDiffRangeAt( destIndex2, diff2, 0 , max(0,half2-1));
    applyDiffRangeAt( destIndex2, diff1, half1,
                      diff1.top()-half1);
    }
#elif  defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1a)
    // wende diff1 (d1->s1) auf dest1 an
    {ActiveAccount active(destS1.account());
    applyDiffRangeAt( destIndex1, diff1);
    }
    // wende diff2 (d2->s2) auf dest2 an
    {ActiveAccount active(destS2.account());
    applyDiffRangeAt( destIndex2, diff2);
    }
#elif  defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_2)
    // kopiere src1 nach dest1
    // wende diff1 (s1->s2) auf  dest1 an
    {ActiveAccount active(destS1.account());
    destS1.copyFrom(srcS1);
    applyDiffRangeAt( destIndex1, diff1 );
    }
    // kopiere src2 nach dest2
    // wende diff2 (s2->s1) auf  dest2 an
    {ActiveAccount active(destS2.account());
    destS2.copyFrom(srcS2);
    applyDiffRangeAt( destIndex2, diff2);
    }
#endif // GUIDED_CROSSOVER_PRODUCTION_VARIANTE_x

#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "singleCrossOver ("
         << destIndex1+1 <<", "<< destIndex2+1 <<" <<--- "
         << srcIndex1+1 <<", "<< srcIndex2+1
         << ")  range("<<firstPos+1<<" ... "<<firstPos+len
         <<")  Q(dest1)="<< destS1.quality()<<"  Q(dest2)="<< destS2.quality()
         << endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q

};

////////////////////////////////////////////////////////////////////////////
//:  generates one of two new States by merging two other States
//   kopiere Stuecke aus srcS1 und srcS2 nach destS1 und destS2
//   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
/////
// generateFirst:
//    true generates dest1 , false generates dest2

void  OPTIMIZER::singleCrossOverAt(
        int destIndex1,
        int srcIndex1 ,int srcIndex2,
        bool generateFirst  // true generate dest1 , false generates dest2
        )
{
    DEBUGTracedFrame;
    ASSERT( destIndex1 != srcIndex1 );
    ASSERT( destIndex1 != srcIndex2 );
    State &destS1 = *stateAt(destIndex1);
    State &srcS1  = *stateAt(srcIndex1);
    State &srcS2  = *stateAt(srcIndex2);

#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "SEQUENCES["
         << destIndex1+1 <<", "
         << srcIndex1+1 <<", "<< srcIndex2+1
         << "] singleCrossOverAt              Q(src1)="
         << srcS1.quality()<<"  Q(src2)="<< srcS2.quality()<< endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q
    CheckLockState checkState1s( &srcS1 );
    CheckLockState checkState2s( &srcS2 );
    CheckLockState checkState1d( &destS1 );

#if defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1) ||
   defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1a)
    StateDiff diff1(stateAt(destIndex1),stateAt(srcIndex1));
    StateDiff diff2(stateAt(destIndex1),stateAt(srcIndex2)); // different index !!
    diff1.finalizePath();
    diff2.finalizePath();
#if defined(GUIDED_CROSSOVER_SELECTION_VARIANTE_3)
    State& diffStart1 = destS1;
    State& diffStart2 = destS1;
#endif
#elif defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_2)
    StateDiff diff1(stateAt(srcIndex1),stateAt(srcIndex2));
    StateDiff diff2(stateAt(srcIndex2),stateAt(srcIndex1));
    diff1.finalizePath();
    diff2.finalizePath();
#if defined(GUIDED_CROSSOVER_SELECTION_VARIANTE_3)
    State& diffStart1 = srcS1;
```

```
    State& diffStart2 = srcS2;
#endif
#endif

    // wähle eine Teilmenge der States aus.
#ifdef GUIDED_CROSSOVER_SELECTION_VARIANTE_1
    //algo: wählt itsGuidedCrossOverSelectionPct% Aktionen zufällig aus.
    //   - kopiert maximal itsGuidedCrossOverSelectionPct% Aktionen.
    //   - die Aktionen häangen nicht zusammen.
    {StateDiff res1; res1.start(diff1.start()).end(diff1.end());
    for(int t = 0; t<diff1.top(); ++t){
        if( itsGuidedCrossOverSelectionPct >= getRandomDouble() ){
            if( itsGuidedCrossOverSelectionPct >= getRandomDouble() ){
                if(diff1.at(t).isLinkOp()){
                    // skip links
                } else if(diff1.at(t).operation() == action_rec::op_add){
                    res1.push(diff1.at(t));
                } else if(diff1.at(t).operation() == action_rec::op_sub ){
                    if(t<diff1.top()
                       && diff1.at(t+1).operation()==action_rec::op_add
                       && diff1.at(t+1).cxID() == diff1.at(t).cxID())
                    {
                        res1.push(diff1.at(t+1));
                        t++;
                    } else {
                        res1.push(diff1.at(t));
                    }
                }
            }
        }
    }
    swap(diff1,res1);
    StateDiff res2; res2.start(diff2.start()).end(diff2.end());
    for(int t = 0; t<diff2.top(); ++t){
        if( itsGuidedCrossOverSelectionPct >= getRandomDouble() ){
            if(diff2.at(t).isLinkOp()){
                // skip links
            } else if(diff2.at(t).operation() == action_rec::op_add){
                res2.push(diff2.at(t));
            } else if(diff2.at(t).operation() == action_rec::op_sub ){
                if(t<diff2.top()
                   && diff2.at(t+1).operation()==action_rec::op_add
                   && diff2.at(t+1).cxID() == diff2.at(t).cxID())
                {
                    res2.push(diff2.at(t+1));
                    t++;
                } else {
                    res2.push(diff2.at(t));
                }
            }
        }
    }
    swap(diff2,res2);
    }

#elif  defined(GUIDED_CROSSOVER_SELECTION_VARIANTE_2)
    // algo:
    //   - kopiert mindestens eine Aktion.
    //   - kopiert maximal itsGuidedCrossOverSelectionPct% Aktionen.
    //   - kopiert jeweils #itsGuidedCrossOverChunkSize zusammenhängende
    //     Aktionen. (so soll die Nachbarschaftsbeziehung erhalten bleiben.)
    // Annahme: diff ist nach AS-Positionen sortiert.
    //
    const int itsGuidedCrossOverChunkSize = 4;
    intVec used;
    int    tCount;
    {
    if(diff1.top()>0){
        StateDiff res1; res1.start(diff1.start()).end(diff1.end());
        used.resize(diff1.top()); used.fill(0);
        tCount = int(diff1.top() * itsGuidedCrossOverSelectionPct);
        tCount = max(1,tCount);
        int eBreak = used.count() * 2;
        while(tCount>0 && eBreak > 0){
            eBreak --;
            int t = longInRange0(diff1.top()-1);
            int sz = itsGuidedCrossOverChunkSize-1;
            while(t<diff1.top() && tCount > 0 && used[t] != 1 && sz-->0){
                // suche die erste unverbrauchte Aktion
                while(t<diff1.top() &&
                      !diff1.at(t).isASOp() && used[t] != 1){
                    // ignoriere Links und bearbeite die nächsten Aktion
                    // wenn sie noch nicht benutzt wurde
                    used[t++] = 1;
                }
                if(t == diff1.top() || used[t] == 1) continue;

                if(diff1.at(t).operation() == action_rec::op_sub ){
                    // wenn dies eine einzelne SUB Aktion ist, verwende diese.
                    // wenn es der erste Teil eines SUB,ADD Paares ist,
                    // gehe zum ADD Teil.
                    if(diff1.at(t+1).operation()==action_rec::op_add
                       && diff1.at(t+1).cxID() == diff1.at(t).cxID())
                    {
                        // teil eines Paares, markieren und zum ADD Teil
                        // weiter gehen.
                        used[t++] = 1;
                    } else {
                        // eine einzelne SUB Operation.
                        // wird unten in if(){} kopiert.
                    }
                }
                //
                if(diff1.at(t).operation() == action_rec::op_sub ||
                   diff1.at(t).operation() == action_rec::op_add){

                    res1.push(diff1.at(t));
                    used[t] = 1;
                    tCount--; sz--;
                }
            }
        }
        swap(diff1,res1);
    }
    if(diff2.top()>0){
        StateDiff res2; res2.start(diff2.start()).end(diff2.end());
        used.resize(diff2.top()); used.fill(0);
        tCount = int(diff2.top() * itsGuidedCrossOverSelectionPct);
        tCount = max(1,tCount);
        int eBreak = used.count() * 2;
        while(tCount>0  && eBreak > 0){
            eBreak --;
            int t = longInRange0(diff2.top()-1);
            int sz = itsGuidedCrossOverChunkSize-1;
            while(t<diff2.top() && tCount > 0 && used[t] != 1 && sz-->0){
                                // suche die erste unverbrauchte Aktion
                while(t<diff2.top() &&
                      !diff2.at(t).isASOp() && used[t] != 1){
                    // ignoriere links und bearbeite die nächsten Aktion
                    // wenn sie noch nicht benutzt wurde
                    used[t++] = 1;
                }
                if(t == diff2.top() || used[t] == 1) continue;

                if(diff2.at(t).operation() == action_rec::op_sub ){
                    // wenn dies eine einzelne SUB Aktion ist, verwende diese.
                    // wenn es der erste Teil eines SUB,ADD Paares ist,
                    // gehe zum ADD Teil.
                    if(diff2.at(t+1).operation()==action_rec::op_add
                       && diff2.at(t+1).cxID() == diff2.at(t).cxID())
                    {
                        // teil eines Paares, markieren und zum ADD Teil
                        // weiter gehen.
                        used[t++] = 1;
                    } else {
                        // eine einzelne SUB Operation.
                        // wird unten in if(){} kopiert.
                    }
                }
                //
                if(diff2.at(t).operation() == action_rec::op_sub ||
                   diff2.at(t).operation() == action_rec::op_add){

                    res2.push(diff2.at(t));
```

```
                    used[t] = 1;
                    tCount--; sz--;
                }
            }
        }
        swap(diff2,res2);
    }
    }
#elif  defined(GUIDED_CROSSOVER_SELECTION_VARIANTE_3)
    // algo:
    //   - kopiert mindestens eine Aktion.
    //   - kopiert maximal itsGuidedCrossOverSelectionPct% Aktionen.
    //   - kopiert jeweils #itsGuidedCrossOverChunkSize zusammenhängende
    //     Aktionen. (so soll die Nachbarschaftsbeziehung erhalten bleiben.)
    //   - kopiere nur, wenn as[i] und as[i+1] verbunden sind.
    // Annahme: diff ist nach AS-Positionen sortiert.
    //
    const int itsGuidedCrossOverChunkSize = 4;
    intVec used;
    int tCount;
    {
    if(diff1.top()>0){
        StateDiff res1; res1.start(diff1.start()).end(diff1.end());
        used.resize(diff1.top()); used.fill(0);
        tCount = int(diff1.top() * itsGuidedCrossOverSelectionPct);
        tCount = max(1,tCount);
        int eBreak = used.count()  * 2;
        while(tCount>0 && eBreak > 0){
            eBreak --;
            int t = longInRange0(diff1.top()-1);
            int sz = itsGuidedCrossOverChunkSize-1;
            while(t<diff1.top() && tCount > 0 && used[t] != 1  && sz>0){
               //////////////////////////
                // suche die erste unverbrauchte Aktion
                while(t<diff1.top() &&
                      !diff1.at(t).isASOp() && used[t] != 1){
                    // ignoriere Links und bearbeite die nächsten Aktion
                    // wenn sie noch nicht benutzt wurde
                    used[t++] = 1;
                }
                // at(t) ist jetzt entweder
                //     hinter der letzten Aktion     -> neuen Start wählen
                //     auf einem benutzen Element    -> neuen Start wählen
                // oder eine AS operation (SUB,ADD,...)
                // (neuen Start wählen == continue da die abbruchkriterien
                //   teil der schleife sind.)
                if(t == diff1.top() || used[t] == 1) continue;

                // neuen Start wählen wenn
                //   cx an pos (at(t).position()) mit dem Nachfolger
                //   verbunden ist.
                // oder kein Nachfolger vorhanden sein kann.
                if(diff1.at(t).position()<diffStart1.linkCount()
                   && !isNull(diffStart1.linkAt(diff1.at(t).position()))
                  )
                {
                    // erzeugt ein künstliches abbruch kriterium, das in der
                    // While() schleife sowieso getestet werden muss.
                    t = diff1.top();
                    continue;
                }

                if(diff1.at(t).operation() == action_rec::op_sub ){
                    // wenn dies eine einzelne SUB Aktion ist, verwende diese.
                    // wenn es der erste Teil eines SUB,ADD Paares ist,
                    // gehe zum ADD Teil.
                    if(diff1.at(t+1).operation()==action_rec::op_add
                       && diff1.at(t+1).cxID() == diff1.at(t).cxID())
                    {
                        // teil eines Paares, markieren und zum ADD Teil
                        // weiter gehen.
                        used[t++] = 1;
                    } else {
                        // eine einzelne SUB Operation.
                        // wird unten in if(){} kopiert.
                    }
                }
                //
                if(diff1.at(t).operation() == action_rec::op_sub ||
                   diff1.at(t).operation() == action_rec::op_add){

                    res1.push(diff1.at(t));
                    used[t] = 1;
                    tCount--; sz--;
                }
               //////////////////////////
            }
        }
        swap(diff1,res1);
    }
    if(diff2.top()>0){
        StateDiff res2; res2.start(diff2.start()).end(diff2.end());
        used.resize(diff2.top()); used.fill(0);
        tCount = int(diff2.top() * itsGuidedCrossOverSelectionPct);
        tCount = max(1,tCount);
        int eBreak = used.count() * 2;
        while(tCount>0 && eBreak > 0){
            eBreak --;
            int t = longInRange0(diff2.top()-1);
            int sz = itsGuidedCrossOverChunkSize-1;
            while(t<diff2.top() && tCount > 0 && used[t] != 1  && sz>0){
               // suche die erste unverbrauchte Aktion
                while(t<diff2.top() &&
                      !diff2.at(t).isASOp() && used[t] != 1){
                    // ignoriere Links und bearbeite die nächsten Aktion
                    // wenn sie noch nicht benutzt wurde
                    used[t++] = 1;
                }
                // at(t) ist jetzt entweder
                //     hinter der letzten Aktion     -> neuen Start wählen
                //     auf einem benutzen Element    -> neuen Start wählen
                // oder eine AS operation (SUB,ADD,...)
                // (neuen Start wählen == continue da die abbruchkriterien
                //   teil der schleife sind.)
                if(t == diff2.top() || used[t] == 1) continue;

                // neuen Start wählen wenn
                //   cx an pos (at(t).position()) mit dem Nachfolger
                //   verbunden ist.
                // oder kein Nachfolger vorhanden sein kann.
                if(diff2.at(t).position()<diffStart2.linkCount()
                   && !isNull(diffStart2.linkAt(diff2.at(t).position()))
                  )
                {
                    // erzeugt ein künstliches abbruch kriterium, das in der
                    // While() schleife sowieso getestet werden muss.
                    t = diff2.top();
                    continue;
                }

                if(diff2.at(t).operation() == action_rec::op_sub ){
                    // wenn dies eine einzelne SUB Aktion ist, verwende diese.
                    // wenn es der erste Teil eines SUB,ADD Paares ist,
                    // gehe zum ADD Teil.
                    if(diff2.at(t+1).operation()==action_rec::op_add
                       && diff2.at(t+1).cxID() == diff2.at(t).cxID())
                    {
                        // teil eines Paares, markieren und zum ADD Teil
                        // weiter gehen.
                        used[t++] = 1;
                    } else {
                        // eine einzelne SUB Operation.
                        // wird unten in if(){} kopiert.
                    }
                }
                //
                if(diff2.at(t).operation() == action_rec::op_sub ||
                   diff2.at(t).operation() == action_rec::op_add){

                    res2.push(diff2.at(t));
                    used[t] = 1;
                    tCount--;  sz--;
                }
               //////////////////////////
            }
        }
    }
```

```
        swap(diff2,res2);
    }
    }
#endif // SELECTION

#if  defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1)
    int max1  = diff1.top();
    int max2  = diff2.top();
    int half1 = max(0,max1/2-1);
    int half2 = max(0,max2/2-1);
#endif

#if defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1)
    // diff1 == (d1->s1)
    // diff2 == (d2->s2)
    // wende die untere Hälfte von diff1 auf  dest1 an
    // wende die obere  Hälfte von diff2 auf  dest1 an
    if(generateFirst)
    {ActiveAccount active(destS1.account());
    applyDiffRangeAt( destIndex1, diff1, 0 , max(0,half1-1));
    applyDiffRangeAt( destIndex1, diff2, half2,
                      diff2.top()-half2);
    }
    // wende die untere Hälfte von diff2 auf  dest2 an
    // wende die obere  Hälfte von diff1 auf  dest2 an
    if(!generateFirst)
    {ActiveAccount active(destS1.account());
    applyDiffRangeAt( destIndex1, diff2, 0 , max(0,half2-1));
    applyDiffRangeAt( destIndex1, diff1, half1,
                      diff1.top()-half1);
    }
#elif  defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1a)
    // wende diff1 (d1->s1) auf dest1 an
    if(generateFirst)
    {ActiveAccount active(destS1.account());
    applyDiffRangeAt( destIndex1, diff1);
    }
    // wende diff2 (d2->s2) auf dest2 an
    if(!generateFirst)
    {ActiveAccount active(destS1.account());
    applyDiffRangeAt( destIndex1, diff2);
    }
#elif  defined(GUIDED_CROSSOVER_PRODUCTION_VARIANTE_2)
    // kopiere src1 nach dest1
    // wende diff1 (s1->s2) auf  dest1 an
    if(generateFirst)
    {ActiveAccount active(destS1.account());
    destS1.copyFrom(srcS1);
    applyDiffRangeAt( destIndex1, diff1 );
    }
    // kopiere src2 nach dest2
    // wende diff2 (s2->s1) auf  dest2 an
    if(!generateFirst)
    {ActiveAccount active(destS1.account());
    destS1.copyFrom(srcS2);
    applyDiffRangeAt( destIndex1, diff2);
    }
#endif // GUIDED_CROSSOVER_PRODUCTION_VARIANTE_x


#ifdef  crossOverAt_DEBUG_CALCULATION_Q
    cout <<  "singleGuidedCrossOver ("
         << destIndex1+1 <<" <<--- "
         << srcIndex1+1 <<", "<< srcIndex2+1
         << ")  range("<<firstPos+1<<" ... "<<firstPos+len
         <<")  Q(dest1)="<< destS1.quality()<<"  Q(dest2)="<< destS2.quality()
         << endl;
#endif  crossOverAt_DEBUG_CALCULATION_Q
}
```

N.22 Quelltext: GeneticRandomOptimizer_Gs2p2.h

```
#ifndef __GeneticRandomOptimizer_Gs2p2_H__
#define __GeneticRandomOptimizer_Gs2p2_H__  1

//
// $Id: GeneticRandomOptimizer_Gs2p2.h,v 1.3 2000/05/29 16:23:04 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "StreamRedirect.h"
#include "gr_prob.h"
#include "MyACG.h"
#include "MySmplHist.h"
#include "Cache.h"
#include "Protein.h"
#include "PeakPool.h"
#include "ParsePoolFile.h"
#include "PeakNet.h"
#include "ParsePeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "AsciiContextReader.h"
#include "Expression.h"
#include "Condition.h"
#include "AllConditions.h"
#include "ExpressionBuilder.h"
#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "ParseCondition.h"

#include "LinkBuilder.h"

#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizerBASE.h"
#include "RandomOptimizer.h"
#include "GeneticRandomOptimizer.h"



class StateIndependent;
class State;
class LinkBuilder;
class RandomOptimizerBASE;
class RandomOptimizer;

//////////////////////////////////////////////////////////////
class GeneticOptimizer;
class GeneticOptimizer_Gs2p2;
class GeneticOptimizer_Gs2p2 : public GeneticOptimizer
{
public:
    typedef GeneticOptimizer_Gs2p2            Self;
    typedef IMngObject< RandomOptimizerBASE > Handle;
    typedef SubRandomOptimizer SUBOptimizer;

    virtual ~GeneticOptimizer_Gs2p2(void);
    GeneticOptimizer_Gs2p2(void);

     //: generates two new States by merging two other States
    void     crossOverAt(
```

```
            int destIndex1,int destIndex2, // dest
            int srcIndex1 ,int srcIndex2   // src
            );
    //:  generates *one* of two new States by merging two other States
    //   kopiere Stuecke aus srcS1 und srcS2 nach destS1
    //   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
    void     singleCrossOverAt(
            int destIndex1,
            int srcIndex1 ,int srcIndex2,
            bool generateFirst   // true generate dest1 , false generates dest2
            );
    const char* crossoverOpName(void)const;
protected:
};

#endif // __GeneticRandomOptimizer_Gs2p2_H__
```

N.23 Quelltext: GeneticRandomOptimizer_Gs2p2.cc

```
#include "GeneticRandomOptimizer_Gs2p2.h"
#include "StateMetrics.h"

//////////////////////////////////////////////////////////////
#include "RandomOptimizer.debug.h"

//////////////////////////////////////////////////////////////////////
//
GeneticOptimizer_Gs2p2::~GeneticOptimizer_Gs2p2(void)
{
    /* EMPTY */
}

GeneticOptimizer_Gs2p2::GeneticOptimizer_Gs2p2(void)
:  GeneticOptimizer()
{
    DEBUGTracedFrame;
}

// SELECT DIFF PARTS:
// variante1:   select random diff pieces !
// variante2:   while reduceing diff, keep longer continuous pieces!
// variante3:   while reduceing diff, keep longer continuous pieces!
//              stop collecting if no link for as[i] to as[i+1] exists.

//#define GUIDED_CROSSOVER_SELECTION_VARIANTE_1
#define GUIDED_CROSSOVER_SELECTION_VARIANTE_2
//#define GUIDED_CROSSOVER_SELECTION_VARIANTE_3


// PRODUCTION VARIANTE:
// variante 1:  keep old  + part diff(old->src1) + part diff(old->src2) -> neu
// variante 1a: keep old  + all of diff(old->src) -> neu
// variante 2:  copy src1 + part diff(src1->src2) -> neu
//
// variante 1a ist die Nr.3 in der externen Benennung!
//

//#define GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1
#define GUIDED_CROSSOVER_PRODUCTION_VARIANTE_2
//#define GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1a

//////////////////////////////////////////////////////
//
const char* GeneticOptimizer_Gs2p2::crossoverOpName(void)const{
    return "Gs2p2";
};

#define OPTIMIZER GeneticOptimizer_Gs2p2

#include "GeneticRandomOptimizer_OPERATOR.cc"
```

N.24 Quelltext: GeneticRandomOptimizer_elitaer.h

```
#ifndef __GeneticRandomOptimizer_elitaer_H__
#define __GeneticRandomOptimizer_elitaer_H__  1
//
//
// $Id: GeneticRandomOptimizer_elitaer.h,v 1.3 2000/05/29 16:23:04 va Exp $
//

#define DONT_TRACE 1

#include "StatArray.h"
#include "Array.h"
#include "macros.h"
#include "Debug.h"
#include <time.h>
#include "Math.h"
#include <math.h>
#include <limits.h>
#include <values.h>
#include "ErrMessage.h"
#include "iomanip.h"
#include "iostream.h"
#include "fstream.h"
#include "cmdline.h"
#include "cmdargs.h"
#include "GetOptions.h"
#include "StreamRedirect.h"
#include "gr_prob.h"
#include "MyACG.h"
#include "MySmplHist.h"
#include "Cache.h"
#include "Protein.h"
#include "PeakPool.h"
#include "ParsePoolFile.h"
#include "PeakNet.h"
#include "ParsePeakNet.h"
#include "Context.h"
#include "ContextList.h"
#include "ContextLock.h"
#include "ContextOp.h"
#include "AsciiContextReader.h"
#include "Expression.h"
#include "Condition.h"
#include "AllConditions.h"
#include "ExpressionBuilder.h"
#include "myiomanip.h"
#include "ParseBlock.h"
#include "ParseContextList.h"
#include "ParseCondition.h"
#include "LinkBuilder.h"
#include "RandomOpti_StateIndependent.h"
#include "RandomOptimizerBASE.h"
#include "RandomOptimizer.h"
#include "GeneticRandomOptimizer.h"

//////////////////////////////////////////////////////////////////

class StateIndependent;
class State;
class LinkBuilder;
class RandomOptimizerBASE;
class RandomOptimizer;

//////////////////////////////////////////////////////////////////
class GeneticOptimizer;
class GeneticOptimizer_elitaer;
class GeneticOptimizer_elitaer : public GeneticOptimizer
{
public:
    typedef GeneticOptimizer_elitaer           Self;
    typedef IMngObject< RandomOptimizerBASE > Handle;
    typedef SubRandomOptimizer SUBOptimizer;

    virtual ~GeneticOptimizer_elitaer(void);
```

```
    GeneticOptimizer_elitaer(void);

    //: generates two new States by merging two other States
    void     crossOverAt(
            int destIndex1,int destIndex2, // dest
            int srcIndex1 ,int srcIndex2   // src
            );
    //:  generates *one* of two new States by merging two other States
    //   kopiere Stuecke aus srcS1 und srcS2 nach destS1
    //   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
    void     singleCrossOverAt(
            int destIndex1,
            int srcIndex1 ,int srcIndex2,
            bool generateFirst   // true generate dest1 , false generates dest2
            );
    const char* crossoverOpName(void)const;
protected:
};

#endif // __GeneticRandomOptimizer_elitaer_H__
```

N.25 Quelltext: GeneticRandomOptimizer_elitaer.cc

```
#include "GeneticRandomOptimizer_elitaer.h"
#include "StateMetrics.h"

////////////////////////////////////////////////////////////
#include "RandomOptimizer.debug.h"

////////////////////////////////////////////////////////////////////
//
GeneticOptimizer_elitaer::~GeneticOptimizer_elitaer(void)
{
    /* EMPTY */
}

GeneticOptimizer_elitaer::GeneticOptimizer_elitaer(void)
:  GeneticOptimizer()
{
    DEBUGTracedFrame;
}

// SELECT DIFF PARTS:
// variante1:   select random diff pieces !
// variante2:   while reduceing diff, keep longer continuous pieces!
// variante3:   while reduceing diff, keep longer continuous pieces!
//              stop collecting if no link for as[i] to as[i+1] exists.

#define GUIDED_CROSSOVER_SELECTION_VARIANTE_1
//#define GUIDED_CROSSOVER_SELECTION_VARIANTE_2
//#define GUIDED_CROSSOVER_SELECTION_VARIANTE_3


// PRODUCTION VARIANTE:
// variante 1:  keep old  + part diff(old->src1) + part diff(old->src2) -> neu
// variante 1a: keep old  + all of diff(old->src) -> neu
// variante 2:  copy src1 + part diff(src1->src2) -> neu
//
// variante 1a ist die Nr.3 in der externen Benennung!
//

#define GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1
//#define GUIDED_CROSSOVER_PRODUCTION_VARIANTE_2
//#define GUIDED_CROSSOVER_PRODUCTION_VARIANTE_1a

////////////////////////////////////////////////////
//
const char* GeneticOptimizer_elitaer::crossoverOpName(void)const{
    return "elitaer";
};
```

```
#define OPTIMIZER GeneticOptimizer_elitaer


////////////////////////////////////////////////////////////////////////
//:  generates two new States by merging two other States
//   wendet Teile der Differenz von (schlecht->gut) auf die
//   Zielzustände an.
//
void     OPTIMIZER::crossOverAt(
           // dest  [ 0 .. sequenceCount()-1]
         int destIndex1,int destIndex2,
           // src   [-1 .. sequenceCount()-1]
         int PRODUCTION_UNUSED(srcIndex1) ,int PRODUCTION_UNUSED(srcIndex2)
         )
{
    DEBUGTracedFrame;
    ASSERT( destIndex1 != srcIndex1 );
    ASSERT( destIndex1 != srcIndex2 );
    ASSERT( destIndex2 != srcIndex1 );
    ASSERT( destIndex2 != srcIndex2 );
    State &destS1 = *stateAt(destIndex1);
    State &destS2 = *stateAt(destIndex2);

    // elitaer means: always copy state #1, no matter what configuration.

    State &srcS1  = *stateAt(0);
    //State &srcS2  = *stateAt(1);

    {ActiveAccount active(destS1.account());
    destS1.copyFrom(srcS1);
    }

    {ActiveAccount active(destS1.account());
    destS2.copyFrom(srcS1);
    }
}


////////////////////////////////////////////////////////////////////////////
//:  generates one of two new States by merging two other States
//   kopiere Stuecke aus srcS1 und srcS2 nach destS1 und destS2
//   die Stuecke sind ( [8 .. asCount()/3]  AS lang)
/////
// generateFirst:
//    true generates dest1 , false generates dest2

void  OPTIMIZER::singleCrossOverAt(
        int destIndex1,
        int PRODUCTION_UNUSED(srcIndex1) ,int PRODUCTION_UNUSED(srcIndex2),
        bool UNUSED(generateFirst)  // true generate dest1 , false generates dest2
        )
{
    DEBUGTracedFrame;
    ASSERT( destIndex1 != srcIndex1 );
    ASSERT( destIndex1 != srcIndex2 );
    State &destS1 = *stateAt(destIndex1);
    State &srcS1  = *stateAt(0);

    {ActiveAccount active(destS1.account());
    destS1.copyFrom(srcS1);
    }
}
```

Literaturstellen

[Kessler95] Kessler, (1995), *Angew. Chem.*, **100**, 300−320.

Aminosäureerkennung

[Grzesiek93] Stephan Grzesiek and Ad Bax,(1993), "*Amino acid type determination in the sequential assignment procedure of uniformly $^{13}C/^{15}N$−enriched proteins"*, J. Biomol. NMR,**3**,185−204

[Groß88] Karl−Heinz Groß, Hans Robert Kalbitzer, (1988), "*Distribution of Chemical Shifts in ^{1}H Nuclear Magnetic Resonance Spectra of Proteins*", J. Magn. Res,. **76**, 87−99

[Wütherich86] K. Wütherich, (1986), "*NMR of proteins and nucleic acids*", Wiley & Sons, Inc., S. 14 ff.

[Olejniczak92] Edward T. Olejniczak, Robert X. Xu, Stephen W. Fesik, (1992), "*A 4D HCCH−TOCSY experiment for assigning the side chain ^{1}H and ^{13}C resonance of proteins*", J. Biomol. NMR, **2**, 655−659

[Wishart91] D. J. Wishart, B. D. Sykes, and F. M. Richards, (1991), "*Relationship between Nuclear Magnetic Resonance Chemical Shift and Protein Structure*", J. Mol. Biol., **222**, 311−333

Kombinatorische Optimierung

[Lawler85] E. L. Lawler et. al. (Hrsg.)"*The traveling salesman problem. A guided tour of combinatorial optimization*.", Wiley, Chichester. 1985.

[Reinelt94] G.Reinelt, (1994), "*Contributions To Practical Traveling Salesman Problem Solving*", Lecture Notes Comp. Sience 840, Spinger Berlin.

[Michalewicz92] Zbigniew Michalewiz, (1996, 1999), "*Genetic Algorithms + Data Structures = Evolution Programs*", **3rd ed**., P. 100.

[Althöfer89] J.Althöfer,K.−U. Koschnick, (1989), "*On the Convergence of Threshold Accepting"*, Reseach Report, Universität Bochum

[Dueck90] G.Dueck, T. Scheuer, (1990), "*Threshold Accepting: A general Purpose Optimization Algorithm Superior to Simulated Annealing"*, Journal of Computational Physics **90**, 161−175

[Hajek85] B.Hajek, (1985), "*A Tutorial Survey of Theory and Applications of Simulated Annealing*" Proc. 24th. IEEE Conf. on Decision and Control, 755−760.

[Kirkpatrick83] S.Kirkpatrick, C.D.Gelatt Jr., M.P. Vecchi (1983), *"Optimization by Simulated Annealing"*, Sience **222**, 671−680

[Lin73] S.Lin, B.W. Kernighan, (1973), "*An effecive Heuristic Algorithm for the Traveling Sales−man Problem"* Operations Research **21**, 498−516

[Laarhoven88] P.J.M. Laarhoven (1988), "*Theoretical and Computational Aspects of Simulated Annea−ling",* Dorctoral Thesis, Erasmus Universiteit Rotterdam.

[Goldberg89] D.E. Goldberg (1989), *"Genetic Algorithms in Search, Optimization and Machine Learning"*, Addison−Wesley

Protein CheY

[Franklin94] F.J.Moy, D.F.Lowry, P.Matsumura, F.W. Dahlquist, J.E. Krywko, P.J.Domaille, (1994), *"Assignments, Secondary Structure, Global Fold, and Dynamics of Chemotaxis Y Protein Using Three− and Four−Dimensional Heteronuclear (^{13}C,^{15}N) NMR Spectroscopy"*, Bio−chemistry 33, 10731−10742.

[Volz93] K. Volz, (1993), Biochemistry **32**, 11741−11753.

[Stock93] A. M. Stock, E. Martinez−Hackert, B.F. Rasmussen, A.H. West, J. B. Stock, D. Ringe, G. A. Petsko, (1993), Biochemistry 32, 13375−13380.

Protein Trigger

[Crooke87] Eliot Crooke, William Wickner, (1987), "*Trigger factor: A soluble protein that folds pro−OmpA into a membrane−assembly−competent form*", Proc. Natl. Acad. Sci. USA, **84**, 5216−5220.

[Hesterkamp96] Thomas Hesterkamp, Stefanie HAuser, Heinrich Lütcke, Bernd Bukau, (1996), "*Escheri−chia coli trigger factor is a prolyl isomerase that associates with nascent polypeptide chains*", Proc. Natl. Acad. Sci USA **93**. 4437−4441.

[Göthel97] S. F. Göthel, R. Schmidt, A. Wipat, N. M. Carter, P. T. Emmerson, C. R. Harwood, M. A. Marahiel, (1997), "*An internal FK506−binding domain is the catalytic core of the prolyl isomerase activity associated with the Bacillus subtilis trigger factor*", Eur. J. Biochem. **244**, 59−65.

[Zarut97] Toralf Zarut, Thomas Tradler, Gerlind Stoller, Christian Scholz, Franz X. Schmid, Gunter Fischer, "*Modular Structure of the Trigger Factor Required for High Activity in Protein Folding*", J. Mol. Biol. **271**, 827−837.

[Reimer98] Ulf Reimer, Gerd Scherer, Mario Drewello, Susanne Kruber, Mike Schrutkowski, Gunter Fischer, (1998), "*Side−chain Effects on Peptidyl−prolyl cis/trans Isomerisation*", J. Mol. Biol. **279**, 449−460.

[Scholz98] Christian Scholz, Matthias Mücke, Michael Rape, Anja Pecht, Andreas Pahl, Holger Bang, Franz X. Schmid, (1998), "*Recoginition of Protein Substrates by the Prolyl Isome−rase Trigger Factor is Independent of Prolin Residues*", J. Mol. Biol. **277**, 723−732.

[Deuerling99] Elke Deuerling, Agnes Schulz−Specking, Toshifumi Tomoyasu, Axel Mogk, Bernd Bukau, (1999), "*Trigger factor and DnaK cooperate in folding of newly synthesized protein*", Nature, Vol. **400**, 693−698.

[Göthel99] S.F.Göthel, M. A. Marahiel, (1999), "*Peptidyl−prolyl cis−trans isomerases, a superfamily of ubiquitous folding catalysts*", Cell. Mol. Life Sci. **55**, 432−436.

Assignment Programme

[Fesik94] Robert P. Meadows, Edward T. Olejniczak, Stephen W. Fesik, (1994) , "*A Computer−Based Protocol for Semi−Automated Assignments and 3D Structure Determination of Proteins*", J. Biomol. NMR **4**, 79−96.

[Markley94] J.B.Olson Jr., J.L. Markley, (1994), "*Evaluation of an algorithm for the automated sequential assignment of protein backbond resonances: A demonstration of the connec−tivity tracing assignment tools (CONTRAST) sorfware package*", J. Biomol. NMR **4**, 385−410.

[Leutner98] Michael Leutner, Ruth M. Gschwind, Jens Liermann, Christian Schwarz, Gerd Gemme−cker, Horst Kessler, (1998), "*Automated backbone assignment of labeld proteins using the threshold accepting algorithm*", J. Biomol. NMR **11**, 31−43.

[Ikura90a] Mitsuhiko Ikura, Lewis E. Kay, Ad Bax, (1990), "*A Novel Approach for Sequential Assignment of 1H, 13C and 15N Spectra of Larger Proteins: Heteronuckear Tipple−Resonance Three−Dimensional NMR Spectroscopy. Application to Calmodulin*", Bioche−mistry **29**, 4659−4667

[Lukin97] Jonathn A. Lukin, Andrew P. Grove, Sarosh N. Talukdar, Chien Ho (1997), "*Automated probabilistic method for assigning backbone resonances of (^{13}C, ^{15}N)−labeled proteins*", J. Biomol NMR **9**. 151−166.

[Zimmerman95] D. Zimmerman, G.T.Montelione (1995), Curr. Opin. Struct. Biol. **5**, 664−673.

[Neidig95] P. Neidig, M. Geyer, A. Görler, C. Antz, R. Saffrich, W. Beneicke, H.R. Kalbitzer (1995), "", J. Biomol. NMR, **6**, 255−270

NMR−Experimente

[Maurer2000] Markus Maurer, (2000), "*Untersuchungen von Proteinen zur Bestimmung von Struktur und Wechselwirkung mit biologisch aktiven Molekülen mittels multidimensionaler NMR−Spektroskopie*", Dissertation, Johann Wolfgang Goethe Universität, Frankfurt am Main.

[Wittekind93] Michael Wittekind, Luciano Mueller, (1993), "*HNCACB, a High−Sensitvity 3DNMR Expe−riment to Correlate Amide−Proton and Nitrogen Resonances with the Alpha− and Beta−Carbon Resonances in Proteins*", J. Magn. Res. Serie **B 101**, 201−205.

[Ikura90] M. Ikura, D.Marion, L.E.Kay, (1990), J. Magn. Res. **86**, 204−209.

[Marion89] D. Marion, M. Ikura, A. Bax, (1989), J. Magn. Res. **85**, 393−399.

[Archer91] J.S. Archer, M. Ikura, D.A. Torchia, A. Bax, (1991), J. Magn. Reson. **95**, 636.

[Grzesiek92a] S. Grzesiek, A. Bax, (1992), J. Magn. Reson., **96**, 432.

[Grzesiek92b] S. Grzesiek, A. Bax, (1992), J. Magn. Reson., **99**, 201.

[Kay90] L. E. Kay, G. M. Clore, A. Bax, A. M. Groneborn (1990), Sience **249**, 411.

[Kay91a] L. E. Kay, M. Ikura, A. Bax, (1991a), J. Magn. Reson. **91**, 84.

[Kay91b] L. E. Kay, M. Ikura, A. Bax, (1991b), J. Magn. Reson. **91**, 422.

[Muhandiram94] D. R. Muhandiram, L. E. Kay (1994) J.Magn. Reson. **B 103**, 203.

[Croasmun94] William Croasmun, Robert M. K. Carlson, (1994), "*Two−dimensional NMR Spektroscopy: Application for Chemists and Biochemists*", 2.te Ausgabe, VCH, New York, Weinheim, Cambidge.

Algorithmen, Informatik und Mathematik

[Aho74] A.V. Aho, J.E. Hopcroft, J.D. Ullman, (1974), *The Design and Analysis of Computer Algorithms*, Addison−Wessley, Reading, MA.

[Bronstein91] I. N. Bronstein, K. A. Semendjajew. (1991), "*Taschenbuch der Mathematik, Ergänzende Kapitel*", Thun, Frankfurt/Main, 6. Aufl.

[Hopcroft79] J.E.Hopcroft, J.D. Ullman, (1979), "*Einführung in die Automatentheorie, Formale Spra−chen und Komplexitätstheorie*", Oldenbourg ,München, Wien, 4.Aufl. 2000.

[Knuth97] Donald Ervin Knuth, (1997), "*The Art of Computer Programming, Seminumerical Algo−rithms, Volume 2*", 3rd Edition, 1997, Addison−Wesley Longman

[Zachmann84] Hans G. Zachman, (1984), "*Mathematik für Chemiker*" ,1.ter Nachdruck 4.te Auflage, Verlag Chemie, Weinheim

Lebenslauf

von

Volker Hans Philiph Apelt

27 August 1965	Geboren in Bad Homburg v. d. H.
August 1971 − Juni 1975	Grundschule in Bad Homburg v. d. H.
August 1975 − Juni 1985	Kaiserin Friedrich Gymnasium in Bad Homburg v. d. H.
13 Juni 1985	Abitur (Note 2,5)
1 Oktober 1985 − 31 Dezember 1986	Bundeswehr
November 1986 − Juli 1992	Studium der Chemie an der Johann Wolfgang Goethe Universität, Frankfurt am Main
11 Mai 1989	Vordiplom (Note "sehr gut")
	Diplomarbeit bei Prof. Dr. Kohlmaier mit dem Thema "Physikalisch−Chemisches Modell zur trockenen Deposition von umweltre−levanten Spurengasen in Böden am Bespiel von SO_2"
13 Juli 1992	Diplom (Note "gut")
Januar 1993 − 2001	Anfertigung der Dissertation im Arbeitskreis von Prof. Dr. C. Griesinger; Assistentenstelle am Institut für Organische Chemie der Johann Wolfgang Goethe Universität, Frankfurt am Main. Thema: "Kombinatorische Optimierungsverfahren zum backbone Assignment von NMR Daten an Proteinen"