Analyse von Heuristiken

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt am Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe – Universität
in Frankfurt am Main

von
Maik Weinard
aus Frankfurt am Main

Frankfurt 2006
(D 30)

vom Fachbereich Informatik und Mathematik der Johann Wolfgang Goethe – Universität als Dissertation angenommen.

Dekan : Prof. Dr. D. Krömker
(Graphische Datenverarbeitung)
Institut für Informatik
Johann Wolfgang Goethe-Universität
60054 Frankfurt am Main

Gutachter :

Prof. Dr. T. Hagerup
(Theoretische Informatik)
Institut für Informatik
Universität Augsburg
86159 Augsburg

Prof. Dr. G. Schnitger
(Theoretische Informatik)
Institut für Informatik
Johann Wolfgang Goethe-Universität
60054 Frankfurt am Main

Datum der Disputation : 16. Jan. 2006

Inhaltsverzeichnis

Kapitel 1

Einleitung

Heuristiken treten insbesondere im Zusammenhang mit Optimierungsproblemen in Erscheinung, bei solchen Problemen also, bei denen nicht nur eine Lösung zu finden ist, sondern unter mehreren möglichen Lösungen eine in einem objektiven Sinne beste Lösung ausfindig gemacht werden soll.

Die Themenkreise, in denen Heuristiken zum Einsatz kommen, reichen vom VLSI-Entwurf bis zum Operation Research und von der Bioinformatik bis zum Netzwerk-Management. Vielen Heuristiken gemein ist dabei, dass ein heuristischer Algorithmus Entscheidungen trifft, die auf Grund von vernünftigen Beurteilungen zum gegenwärtigen Zeitpunkt *gut aussehen*. Die Hoffnung ist stets, dass man durch das Hintereinanderausführen dieser erfolgversprechenden Schritte letztlich auch zu einer Lösung akzeptabler Güte kommt.

Das klassische Einsatzgebiet von Heuristiken sind Probleme, bei denen alle bekannten exakten Algorithmen einen inakzeptablen Ressourcenverbrauch aufweisen. Die knappen Ressourcen sind in der Regel die Laufzeit des Algorithmus oder sein Speicherbedarf. Aus Sicht der Theoretischen Informatik hatten Heuristiken zunächst den unliebsamen Charme einer weißen Fahne zur Kapitulation. So heißt es in W.J. Pauls Buch "Komplexitätstheorie":

> Insofern bedeutet der Nachweis, daß eine Sprache nicht in $\mathcal{P}$ liegt, daß ihre charakteristische Funktion für praktische Zwecke als nicht berechenbar angesehen werden muss.
>
> *W.J. Paul, Komplexitätstheorie (Pa78), S.182*

Spätestens seit der Etablierung des Begriffs der $\mathcal{NP}$-Vollständigkeit war aber klar geworden, dass zu viele relevante Probleme höchstwahrscheinlich keine exakten Algorithmen mit polynomieller Laufzeit haben. Sich der wissenschaftlichen Behandlung der entsprechenden Probleme zu verweigern, kann hier jedoch keine Lösung sein. Einem Vertreter einer Anwendungsdis-

ziplin ist mit dem theoretischen Bescheid, sein Problem sei $\mathcal{NP}$-vollständig und folglich sei es besser, er hätte es erst gar nicht, nicht gedient.

So endet die Arbeit "On Finding Minimal Length Superstrings", in der John Gallant, David Maier und James A. Storer die Härte des Problems kürzester Superstrings nachweisen, passenderweise mit der Aufforderung:

> Because the superstring problem has many practical applications, the $\mathcal{NP}$-completeness results presented in this paper should not discourage future research regarding the superstring problem. Rather, they should provide the impetus for studying approximation algorithms and heuristics for finding a minimal length superstring.
>
> *J. Gallant, D. Maier und J.A. Storer, On Finding Minimal Length Superstrings (GMS80)*

Bemerkenswert ist, dass hier zwischen Approximationsalgorithmen und Heuristiken unterschieden wird. Man sollte Heuristiken, die für Optimierungsprobleme eingesetzt werden, wohl eher als eine Teilmenge der Approximationsalgorithmen verstehen. Auf informaler Ebene könnte man definieren: Eine Heuristik ist ein Approximationsalgorithmus, der nicht mit der Zielvorgabe der Analysierbarkeit entwickelt wurde. Heuristiken sammeln algorithmische Ideen, von denen angenommen wird, dass sie zu einer Verbesserung der gefundenen Lösung führen. Ob der erwartete Gewinn dabei auch zu einem verbesserten Approximationsfaktor führt, und ob diese Verbesserung formal beweisbar ist, ist zunächst von nachrangigem Interesse. In "Complexity and Approximation" wird eine ähnliche Begriffsbildung vorgenommen:

> [...] it is well known, for many problems, there exist algorithms that do not provide any guarantee about their worst case behaviour in terms of both the quality of the returned solution and the efficiency of their execution, even if in many cases they return good solutions in a short time. In the following we refer to such algorithms as heuristic methods (or heuristics).
>
> *G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation (ACG+99), S.321*

Da auf Analysierbarkeit beim Entwurf von Heuristiken nicht geachtet wird, sind Heuristiken oft schwierig zu analysieren.

> Heuristic algorithms [...] have often proved quite successful in practice, although a considerable amount of fine tuning is usually required in order to achieve satisfactory performance.
>
> As a consequence, it is rarely possible to predict how well such algorithms will perform by formally analyzing them beforehand. Recently however a number of results have been obtained that show,

> that heuristic algorithms may not always be so immune to formal analysis.
>
> *M.R. Garey, D.S. Johnson, Computers and intractability (GJ79), S.122*

Die Schwierigkeit, Heuristiken formal zu analysieren, steht häufig im krassen Kontrast zur Einfachheit der Heuristik selbst. Die Greedy-Heuristik für das Problem kürzester Superstrings, der Kapitel 2 und damit der Löwenanteil dieser Dissertation gewidmet ist, ist ein Paradebeispiel hierfür. Jedem Schüler, dem ein paar Buchstaben und die natürlichen Zahlen bekannt sind, kann das Problem kürzester Superstrings, wie auch die Greedy-Heuristik zu dessen Lösung verstehen. Die Frage, wie gut der Approximationsfaktor ist, gibt Forschern dagegen bis heute Rätsel auf.

Hier ist zudem anzumerken, dass die theoretische Analyse eines Approximationsfaktors sich fast immer auf den Worst-Case bezieht. Ist der Worst-Case selten, so ist der unmittelbare Erkenntnisgewinn für praktische Zwecke scheinbar gering. Auch stellt sich die Frage, ob der Aufwand gerechtfertigt ist, der betrieben wird, um eine Worst-Case Garantie für einen Algorithmus aussprechen zu können, der gar nicht mit dem Ziel der Analysierbarkeit konstruiert worden ist, und dessen Leistung für *real-world-input* deutlich besser als im Worst-Case ist. Vijay V. Vazirani kommt hier zu einer klaren Antwort

> With practitioners looking for high performance algorithms having error within 2% or 5% of the optimal, what good are algorithms that come within a factor of 2, or even worse, $O(\log(n))$, of the optimal? Further, by this token, what is the usefulness of improving the approximation guarantee from, say factor 2 to $3/2$?
>
> Let us address both issues and point out some fallacies in these assertions. The approximation guarantee only reflects the performance of the algorithm on the most pathological instances. Perhaps it is more appropriate to view the approximation guarantee as a measure that forces us to explore deeper into the combinatorial structure of the problem and discover more powerful tools for exploiting this structure.
>
> *Vijay V. Vazirani, Approximation Algorithms, (Va01), S IX*

Die Analyse des Approximationsfaktors als Wegweiser zu kritischen Probleminstanzen und damit zu strukturellen Einsichten in das Problem, diesen Ansatz verfolgt auch die vorliegende Arbeit. Die hier vorzustellenden Fortschritte in der Analyse der Greedy-Heuristik für das Problem kürzester Superstrings basieren im Wesentlichen auf einem genaueren Verständnis der Schwierigkeit von Instanzen. Es wird die Frage aufgeworfen, wann – für welche Instanzen – die Heuristik Gefahr läuft, eine schlechte Lösung zu liefern.

Sind die strukturellen Eigenschaften solcher Instanzen verstanden, so kann man zum Einen genauer beschreiben, wann die Heuristik bedenkenlos eingesetzt werden kann. Zum anderen scheint es ein natürliches Vorgehen zu sein, Verbesserungen an einem Algorithmus dadurch vorzunehmen, dass man ihn gegen den Worst-Case robuster macht. Die Hoffnung ist, dass so gefundene Verbesserungen sich nicht allein für die Worst-Case Instanz auswirken, sondern insgesamt zu einer Steigerung der Leistung der Heuristik führen.

Zu dem klassischen Einsatzszenario von Heuristiken – Probleme, die keine akzeptablen exakten Algorithmen haben – sind weitere hinzugekommen. Eines davon ist das Gebiet der Online-Algorithmen. Hierbei geht es um Probleme, bei denen bestimmte Bestandteile der Eingabe erst nach und nach bekannt werden, der Algorithmus aber vorher schon Entscheidungen treffen muss, die er später nicht mehr zurücknehmen kann. Hier können Probleme, die in der klassischen Offline-Sicht längst verstanden sind, zu neuem Leben erwachen. Ein einfaches Beispiel besteht in der Routenplanung. Mit den Entfernungsangaben einer Landkarte ist es ein Leichtes, etwa mit Dijkstras Algorithmus, den kürzesten Weg von A nach B zu finden. Sobald man aber auch aktuelle Verkehrsentwicklungen wie Staus oder Straßensperren ins Modell aufnimmt, kann man realistischerweise nicht mehr hoffen, den schnellsten Weg ins Ziel zu finden, und gute Entscheidungshilfen – Heuristiken – sind gefragt. Bei einigen $\mathcal{NP}$-harten Problemen bringt die Online-Sicht wettbewerbsfähige Heuristiken hervor.

> Although many fundamental combinatorial optimization problems (e.g. bin packing, machine scheduling, and load balancing) do not always have to be solved online (i.e., they are traditionally considered offline problems), for such problems online algorithms constitute an interesting restricted class of algorithms that may sometimes yield surprisingly good approximate solutions.
>
> *A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, (BEY98), S.XIII*

Im zweiten großen Kapitel dieser Arbeit liegt dieser zweite Einsatzfall für Heuristiken vor – die Notwendigkeit von Entscheidungen bei unvollständiger Information. Pakete sollen innerhalb eines Netzwerks versandt werden, und realistischerweise darf weder angenommen werden, dass ein Router weiß, welche Pakete in der Zukunft eingefügt werden, noch, dass er über die aktuelle Auslastung anderer Knoten und Kanten informiert ist. Der Online-Restriktion, der Zukunftsblindheit, wird hier also noch eine Lokalitätsrestriktion zur Seite gestellt. Beurteilt wird die Güte von Heuristiken im Adversarial Queueing Modell, das eine Worst-Case Betrachtung erlauben soll.

Hier kann gezeigt werden, dass für eine große Klasse von Heuristiken eine akzeptable Güte, gemessen in Zustellzeiten oder Gesamtpaketaufkom-

men, ausgeschlossen werden kann. Auch hier ist der Schlüssel zum Erfolg ein Verständnis der Effekte, die eine konkrete Heuristik scheitern lassen. Insofern erlaubt dieses negative Ergebnis Rückschlüsse auf mögliche erfolgreiche Heuristiken, insbesondere in der Frage, wovon diese Heuristiken ihre lokalen Entscheidungen abhängig machen sollten.

1.1 Grundlagen

In diesem Abschnitt sammeln wir einige Definitionen und Grundlagen, die in den folgenden Kapiteln ohne besondere Erwähnung zur Anwendung kommen. Darüberhinaus werden elementare Rechengesetze, logische Schlusstechniken und fundamentale Konzepte der Informatik natürlich vorausgesetzt. In Standardlehrbüchern (CLR90; Ho97; Ka91; Va01) finden sich bei Bedarf Motivationen der Begriffe und Beweise zu den hier aufgelisteten Lemmata.

Definition 1.1 (Asymptotische Notation) *Seien f und g Funktionen mit nichtnegativen reellen Werten. Wir führen ein:*

- $f(n) = O(g(n)) \iff \exists c > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
- $f(n) = \Omega(g(n)) \iff g(n) = O(f(n))$
- $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$
- $f(n) = o(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- $f(n) = \omega(g(n)) \iff g(n) = o(f(n))$

Lemma 1.1 (Summenformeln)

1. $\sum_{i=1}^{n} i = \frac{n\cdot(n+1)}{2}$
2. $\sum_{i=1}^{n} i^2 = \frac{1}{6} \cdot (2n+1) \cdot (n+1) \cdot n$
3. $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ *für* $a \neq 1$
4. $\sum_{i=0}^{n} \binom{n}{i} = 2^n$
5. $\sum_{i=1}^{n} f(i-1) = \sum_{i=0}^{n-1} f(i)$
6. $\sum_{i=1}^{n} f(i) - f(i-1) = f(n) - f(0)$

Die Menge der Permutationen von n Objekten bezeichnen wir mit $\mathcal{S}_n$.

Lemma 1.2 (Kombinatorik)

Die Menge $\mathcal{S}_n$ besteht aus $n!$ Permutationen.

Es gibt $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$ *Teilmengen der Größe* k *in einer Menge der Größe* n*.*

Lemma 1.3 (Schubfachprinzip) *Werden* n *Objekte auf* k *Fächer verteilt, so existiert ein Fach, auf das mindestens* $\lceil \frac{n}{k} \rceil$ *Objekte entfallen.*

Heuristiken fallen in das Gebiet der Approximationsalgorithmen. Daher listen wir die für uns relevanten grundlegenden Begriffe auf.

Definition 1.2 (NP-Optimierungsproblem) *Wir beschreiben ein* NP*-Optimierungsproblem durch das Tripel* $P = (opt, f, L)$*. Dabei ist* $opt \in \{min, max\}$*.* L *ist ein Prädikat, so dass* x *genau dann eine Lösung für Instanz* x_0 *ist, wenn* $L(x_0, x)$ *wahr ist. Darüberhinaus darf eine Lösung* x *für eine Instanz* x_0 *nur polynomielle Größe in* $|x_0|$ *haben.* L *muss von einem deterministischen Algorithmus in Zeit polynomiell in* $|x_0|$ *und* $|x|$ *entschieden werden können. Die Zielfunktion* $f(x_0, x)$ *muss ebenso deterministisch in polynomieller Zeit berechnet werden können. Für Instanz* x_0 *ist*

$$opt\ f(x_0, x),\ \ so\ dass\ L(x_0, x)$$

der optimale Wert des Optimierungsproblems für Instanz x_0*.*

Die Klasse aller NP*-Optimierungsprobleme nennen wir* NPO*.*

Die Forderungen aus Definition 1.2 verlangen im Wesentlichen, dass die Lösungen zu NPO-Problemen kompakt sind und, dass zu einer vorgeschlagenen Lösung schnell entschieden werden kann, ob sie in der Tat eine Lösung darstellt, und wie gut sie ist.

Definition 1.3 (Approximationsfaktor) *Sei* A *ein Algorithmus für ein* NPO*-Problem* $P = (opt, f, L)$*. Sei* X_n *die Menge aller Instanzen der Größe* n *und sei* $A(x_0)$ *für* $x_0 \in X_n$ *die von* A *für Eingabe* x_0 *gelieferte Lösung. Sei weiter* $opt(x_0)$ *eine optimale Lösung für* x_0*. Als Approximationsfaktor* V *von Algorithmus* A *bezeichnet man die Funktion* $\mathbb{N} \to \mathbb{R}$ *mit*

$$V(n) := \sup \left\{ \frac{f(A(x_0))}{f(opt(x_0))} \middle| x_0 \in X_n \right\},$$

falls $opt = min$ *und*

$$V(n) := \sup \left\{ \frac{f(opt(x_0))}{f(A(x_0))} \middle| x_0 \in X_n \right\}$$

sonst. Wir sagen, dass Algorithmus A $f(n)$*-approximativ ist, wenn der Verlustfaktor durch* $f(n)$ *nach oben beschränkt ist.*

Approximationsfaktoren sind umso besser, je kleiner sie sind, wobei ein Algorithmus, der das Problem exakt löst, 1-approximativ ist. Klassen bezüglich der möglichen Approximationsgüte entstehen nun natürlich.

Definition 1.4 (Die Klasse $\epsilon - APX$) *Sei $\epsilon : \mathbb{N} \to \mathbb{R}$ eine Funktion mit nichtnegativen werten. Die Klasse $\epsilon - APX$ umfasst alle Optimierungsprobleme für die es effiziente $O(\epsilon)$-approximative Algorithmen gibt.*

Insbesondere ist die Klasse $1 - APX$ die Klasse aller Probleme, die mit konstantem Approximationsfaktor approximiert werden können. Sie wird auch einfach APX genannt. Innerhalb von APX liegen zum einen Probleme, für die zu *jedem* vorgelegten $\delta > 1$ ein effizienter δ-approximativer Algorithmus angegeben werden kann (polynomielle Approximationsschemata), aber auch solche, für die Approximationsfaktoren unter einer gewissen Schranke ausgeschlossen sind.

Zum Nachweis von unteren Schranken dient das Konzept der approximationsbewahrenden (engl.: **a**pproximation **p**reserving) Reduktion.

Definition 1.5 (Approximationsbewahrende Reduktion) *Sind $P_1 = (opt_1, f_1, L_1)$ und $P_2 = (opt_2, f_2, L_2)$ zwei NPO-Optimierungsprobleme, so sagen wir, dass P_1 auf P_2 AP-reduzierbar ist ($P_1 \leq_{AP} P_2$), falls es in polynomieller Zeit berechenbare Funktionen INSTANZ und LÖSUNG und eine positive Konstante α gibt, so dass die folgenden Eigenschaften für jede Instanz x_1 und jede positive rationale Zahl ϵ erfüllt sind:*

- $x_2 = \text{INSTANZ}(x_1, \epsilon)$ *ist eine Instanz von P_2. INSTANZ kann in Zeit polynomiell in $|x_1|$ (bei fixiertem ϵ) berechnet werden. INSTANZ bildet also P_1-Instanzen auf P_2-Instanzen ab.*
- *für jede Lösung y_2 mit $L_2(x_2, y_2)$ ist $y_1 = \text{LÖSUNG}(x_1, y_2, \epsilon)$ eine Lösung zur Instanz x_1, d.h. das Prädikat $L_1(x_1, y_1)$ ist wahr. LÖSUNG kann in Zeit polynomiell in $|x_1| + |y_2|$ (bei fixiertem ϵ) berechnet werden. LÖSUNG bildet also Lösungen zur Instanz x_2 auf Lösungen zur Instanz x_1 ab.*
- *Wenn y_2 eine $(1 + \epsilon)$-approximative Lösung zur Instanz x_2 ist, dann ist y_1 eine $(1 + \alpha \cdot \epsilon)$-approximative Lösung zur Instanz x_1.*

Wir können nun einen Vollständigkeitsbegriff für die Klasse APX angeben.

Definition 1.6 (APX-Vollständigkeit) *Sei P ein NPO-Optimierungsproblem. P ist APX-hart, wenn für jedes Problem $Q \in APX$ die Reduktion $Q \leq_{AP} P$ gilt. Gilt zudem $P \in APX$, so sagen wir, dass P APX-vollständig ist.*

APX-vollständige Probleme sind also schwerstmöglich unter den APX-Problemen. Für sie existieren insbesondere keine polynomiellen Approximationsschemata. Das Problem kürzester Superstrings, dem das zweite Kapitel dieser Arbeit gewidmet ist, fällt in diese Klasse (BLTY94).

1.2 Publikationen

Die in dieser Arbeit vorgestellten Ergebnisse sind in der Substanz in den Publikationen (WS03; LW04; LW05; We05; WS06) bereits veröffentlicht worden. Die Darstellung der Ergebnisse ist in dieser Dissertation allerdings ausführlicher und im Falle von Kapitel 2 ist die Beweisführung gegenüber den Arbeiten (WS03; LW04; LW05; WS06) verbessert worden.

Kapitel 2

Kürzeste Superstrings

Das zentrale Thema dieses Kapitels ist das Problem der kürzesten Superstrings: Gegeben ist eine Menge $S := \{s_1, \ldots, s_n\}$ von Strings. Gesucht ist ein String s, so dass jeder String $s_i \in S$ Teilstring von s ist. Die Länge von s ist dabei zu minimieren. Wir definieren zunächst die Grundbegriffe im Kontext von Strings.

Definition 2.1 (Strings) *1. Ein String ist eine endliche Aneinanderreihung von Zeichen eines Alphabets Σ. Die Länge eines Strings $s = s_1, s_2, \ldots, s_k$ ist die Anzahl seiner Zeichen: $|s| = k$. Der leere String wird mit ϵ dargestellt und hat Länge 0.*

2. *Zwei Strings u und v sind gleich, wenn ihre Längen gleich sind und $u_i = v_i$ für alle $i \leq |u|$ gilt.*

3. *Die Konkatenation zweier Strings $u \circ v$ ist der String*

$$u_1, \ldots, u_{|u|}, v_1, \ldots, v_{|v|}.$$

4. *Sei u ein String, dann definieren wir u^k mit $k \in \mathbb{N}$ induktiv*

$$u^k := \begin{cases} \epsilon & \text{für } k = 0 \\ u^{k-1} \circ u & \text{für } k \geq 1 \end{cases}$$

5. *String u ist Teilstring von v, wenn es Strings w_1, w_2 gibt, so dass $v = w_1 \circ u \circ w_2$ gilt. String u ist Präfix von v, wenn ein String w existiert, so dass $v = u \circ w$ gilt. String u ist Suffix von v, wenn ein String w existiert, so dass $v = w \circ u$ gilt.*

6. *String u ist echter Teilstring [Präfix, Suffix] von v, wenn u Teilstring [Präfix, Suffix] von v ist und $u \neq v$ gilt.*

Es wird beim Problem kürzester Superstrings stets davon ausgegangen, dass kein String der Eingabe Teilstring eines anderen Strings der Eingabe ist. Diese Eigenschaft kann schnell überprüft werden und, falls ein String gefunden wird, der Teilstring eines anderen ist, so kann der kleinere String entfernt werden. Eine Lösung des in diesem Sinne bereinigten Problems wird stets auch eine Lösung des ursprünglichen Problems sein.

Das Problem kürzester Superstrings ist 1978 von D. Maier und J. A. Storer als $\mathcal{NP}$-vollständig erkannt worden (MS78; GMS80). 1991 haben Blum et al. (BLTY94) nachgewiesen, dass das Problem APX-vollständig ist, also nicht beliebig genau approximiert werden kann, falls P$\neq\mathcal{NP}$. Im selben Paper wurde der erste Approximationsalgorithmus mit konstantem Approximationsfaktor vorgestellt. Dabei handelt es sich um die Greedy-Heuristik, die zunächst dargestellt werden soll.

Das wesentliche Konzept, auf dem die Greedy-Heuristik basiert, ist die Überlappung zweier Strings:

Definition 2.2 *Seien a und b zwei nichtleere Strings.*

1. *$\langle a, b\rangle$ beschreibt den längsten echten Suffix von a, der auch ein Präfix von b ist. Die Überlappung (a, b) von a und b ist die Länge von $\langle a, b\rangle$.*

2. *Mit $prf\,(a, b)$ bezeichnen wir den Präfix von a mit Länge $|a| - (a, b)$, also genau die Zeichen, die nicht zu $\langle a, b\rangle$ gehören.*

3. *Wir bezeichnen $prf\,(a, a)$ als die Periode von a und schreiben dafür kurz p_a.*

Wir haben p_a die Periode von a genannt, weil a stets ein Teilstring von p_a^l für ein hinreichend großes l ist.

Bei einer Lösung des Superstringproblems ist man intuitiv bestrebt, möglichst viele und große Überlappungen auszunutzen. Die Greedy-Heuristik (Algorithmus 2.1) erscheint vor dem Hintergrund dieser Überlegung natürlich.

Von der Korrektheit überzeugt man sich rasch. Zu jedem String s der Eingabe existiert zu jedem Zeitpunkt ein String in der Menge S, der s enthält. Zu Beginn ist diese Invariante trivialerweise erfüllt. Es genügt zu beobachten, dass in jeder Iteration c sowohl a als auch b enthält. Folglich produziert der Algorithmus einen Superstring der Menge S. Die gefundene Lösung wird jedoch im Allgemeinen nicht optimal sein.

Beispiel 1: Man betrachte die drei Strings $s_1 = w(xy)^k$, $s_2 = (yx)^k$ und $s_3 = (xy)^k z$ über dem Alphabet $\Sigma := \{w, x, y, z\}$. Es ergeben sich die in der

Die Greedy-Heuristik

1. Eingabe: Eine Menge S von Strings
2. Solange $|S| > 1$ wiederhole:
   (a) Wähle $a, b \in S$, so dass $a \neq b$ gilt und (a, b) maximal ist.
   (b) Berechne $c = prf(a, b) \circ b$
   (c) Berechne $S := (S \setminus \{a, b\}) \cup \{c\}$
3. AUSGABE: Der in S verbliebene String.

Algorithmus 2.1: Die Greedy-Heuristik für das Problem kürzester Superstrings

nachfolgenden Tabelle aufgelisteten Überlappungen:

	$s_1 = w(xy)^k$	$s_2 = (yx)^k$	$s_3 = (xy)^k z$
$s_1 = w(xy)^k$		$2k-1$	$2k$
$s_2 = (yx)^k$	0		$2k-1$
$s_3 = (xy)^k z$	0	0	

Folglich wählt die Greedy-Heuristik im ersten Schleifendurchlauf $a = s_1$ und $b = s_3$ und generiert den String $c = w(xy)^k z$, der zum verbliebenen String s_2 keinen Überlappung hat. String s_2 und c werden im letzten Schritt einfach konkateniert und der resultierende String hat eine Länge von $4k + 2$. Die Lösung $w(xy)^{k+1}z$ hat Länge $2k + 4$ und, da k frei gewählt werden kann, nähert sich der Approximationsfaktor dem Wert 2 beliebig nahe an.

Dieses Beispiel wurde 1986 von J. Tarhio und E. Ukkonen (TU88) eingesetzt, um eine Vermutung aus dem Jahr 1982 (Ga82), die Greedy-Heuristik sei 1,5-approximativ, zu widerlegen. Desweiteren zeigen sie, dass die Greedy-Heuristik 2-approximativ ist, wenn man als Kostenmaß die Kompression, also die Zahl der gegenüber der Konkatenation eingesparten Buchstaben wählt: Enthält die Menge S also Strings mit einer Gesamtlänge von m und hat der optimale Superstring eine Länge von $m - x$, so liefert die Greedy-Heuristik einen String der Länge höchstens $m - \frac{x}{2}$. Aus diesem Sachverhalt lässt sich allerdings keine Schranke für die Längenbeziehung beider Strings ableiten. Die Vermutung, dass Faktor zwei auch für die Länge gilt – die Greedy-Conjecture – wird hier erstmals formuliert. Blum et al. haben 1990 (BLTY94) nachgewiesen, dass die Greedy-Heuristik einen Superstring liefert, der höchstens vier mal so lang ist wie das Optimum.

Danach wurden für das Problem kürzester Superstrings eine Reihe von Approximationsalgorithmen vorgestellt, deren zeitliche Abfolge nebst erreichtem Approximationsfaktor und Referenz in Tabelle 2.1 dargestellt ist.

Autoren	Ref.	Vf.
Blum et al. (1991)	(BLTY94)	3
S. Teng, F. Yao (1993)	(TY97)	$2\frac{8}{9}$
A. Czumaj et al. (1994)	(CGPR97)	$2\frac{5}{6}$
R. Kosaraju, J. Park, C.Stein (1994)	(KPS94)	$2\frac{50}{63}$
C. Armen, C. Stein (1995)	(AS95)	$2\frac{1}{4}$
C. Armen, C. Stein (1996)	(AS98)	$2\frac{2}{3}$
D. Breslauer, T. Jiang, Z. Jiang (1997)	(BJJ97)	2.596
Z. Sweedyk (1999)	(Sw99)	$2\frac{1}{2}$

Tabelle 2.1: Die zu ihrer Zeit besten Approximationsalgorithmen für das Problem kürzester Superstrings.

Bemerkenswert an dieser Entwicklung ist, dass es bis heute nicht gelungen ist, die simple Greedy-Heuristik zu einem Approximationsfaktor größer zwei zu zwingen. Erst 2005 ist es H. Kaplan und N. Shafrir gelungen die obere Schranke für die Greedy-Heuristik von 4 auf 3,5 zu verbessern (KS05). Es wird allgemein vermutet, dass der Approximationsfaktor der Greedy-Heuristik in der Tat zwei beträgt (vergl. (Gu97)). Das heißt, es wird gemeinhin angenommen, dass alle in Tabelle 2.1 aufgeführten, z.T. äußerst diffizilen Algorithmen von der vergleichsweise sehr simplen Greedy-Heuristik geschlagen werden. Konsequenterweise werden diese aufwendigen Algorithmen praktisch nicht eingesetzt.

Die Greedy-Conjecture ist das zentrale Thema dieses Kapitels. Die erzielten Ergebnisse sind im Wesentlichen:

- Durch die Betrachtung von *Greedyordnungen* können *bedingte lineare Ungleichungen* nutzbar gemacht werden. Dieser Ansatz ermöglicht den Einsatz linearer Programmierung zum Auffinden interessanter Instanzen und eine Vertiefung des Verständnisses solcher *schwerer* Instanzen. Dieser Ansatz wird in Abschnitt 2.2 eingeführt und in Abschnitt 2.3.1 wird eine Interpretation des dualen Problems dargestellt.

- Für die nichttriviale, große Teilklasse der *bilinearen* Greedyordnungen wird gezeigt, dass die Länge des von der Greedy-Heuristik gefundenen Superstrings und die des optimalen Superstrings sich höchstens um die Größe einer optimalen Kreisüberdeckung[1] der Strings unterscheiden. Da eine optimale Kreisüberdeckung einer Menge von Strings stets höchstens so groß ist wie ein optimaler Superstring (man schließe einen Superstring zu einem einzelnen Kreis), ist das erzielte Ergebnis für die

[1]Eine Kreisüberdeckung einer Stringmenge $\{s_1, \ldots, s_n\}$ ist eine Menge $\{r_1, \ldots, r_k\}$ von Strings, so dass jeder String s_i Teilstring eines r_j^v für hinreichend großes v ist. Die Größe einer Kreisüberdeckung ist die Summe der Längen der r_j.

betrachtete Teilklasse der Greedyordnungen stärker als die klassische Greedy-Conjecture. (Theorem 2.1, Abschnitt 2.4)

- Es wird eine neue bedingte lineare Ungleichung auf Strings – die Tripelungleichung – gezeigt, die für das eben genannte Hauptergebnis wesentlich ist.

- Schließlich wird gezeigt, dass die zum Nachweis der oberen Schranke von 3.5 für den Approximationsfaktor herangezogenen bedingten Ungleichungen (etwa die Monge-Ungleichung) inhärent zu schwach sind, um die Greedy-Conjecture selbst für lineare Greedyordnungen zu beweisen. Also ist die neue Tripelungleichung auch notwendig. Zuletzt wird gezeigt, dass das um die Tripelungleichung erweiterte System bedingter linearer Ungleichungen inhärent zu schwach ist, um die klassische Greedy-Conjecture für beliebige Greedyordnungen zu beweisen. (Abschnitt 2.6)

Zunächst wird in Abschnitt 2.1 die zentrale Bedeutung des Problems kürzester Superstrings in der Bioinformatik – der Grund für die z.Zt. hohe Aufmerksamkeit, die das Problem erfährt – kurz dargestellt.

2.1 Kürzeste Superstrings und DNA-Assembly

Das Problem kürzester Superstrings ist als ein zentrales Stringproblem für sich genommen interessant. Es erhält jedoch zusätzliche Bedeutung durch Anwendungen im Kontext der DNA-Sequenzierung.

Eine erschöpfende Diskussion dieses Themas würde den Rahmen dieser Arbeit sprengen. Das Buch (Gu97) bietet eine hervorragende Beschreibung dieses Themengebiets aus der Perspektive der Informatik.

Die DNA (Desoxyribonucleinsäure) enthält alle wesentlichen Informationen über die Proteinsynthese in den Zellen des betreffende Lebewesen. Grundbausteine sind die Nukleotide Adenin (A), Guanin (G), Thymin (T) und Cytosin (C). Die Struktur der DNA ist die Doppelhelix, in der jeweils komplementäre Nukleotide einander gegenüberstehen; dabei sind A und T sowie C und G komplementär zueinander. Die Anordnung der Nukleotide auf der DNA lässt eine natürliche Interpretation als zwei im besagten Sinne komplementäre Strings zu, wobei die Nukleotide die Rolle der Buchstaben übernehmen.

Die durch den String der Basen abgebildete Information nennt man die *Primärstruktur*, Faltungen (Bindungen zwischen nicht benachbarten Punkten der Kette) und räumliche Ausrichtung werden als *Sekundär-* bzw. *Tertiärstruktur* bezeichnet. Die Konzentration auf die Primärstruktur, also auf Strings, wird damit begründet, dass die in die Ebene projizierte Sekundärstruktur und die räumliche Tertiärstruktur im Wesentlichen als Funktionen der Primärstruktur verstanden werden können.

Die vererbbare DNA dient als *Bauanleitung* für Proteine. Jeweils ein Tripel von Nukleotiden kodiert eine Aminosäure, die die Grundbausteine aller Proteine sind. Bei genetisch bedingten Krankheiten ist ein Fehler in der DNA Ursache dafür, dass ein benötigtes Protein nicht synthetisiert wird, ein falsches synthetisiert wird oder eine falsche Quantität eines Proteins synthetisiert wird.

Durch eine Lokalisierung aller relevanten Passagen der DNA – der Gene – und einer sich anschließenden Erforschung der Bedeutung und des Zusammenspiels der einzelnen Gene erhofft man sich fundamentale Fortschritte im Kampf gegen genetisch bedingte Krankheiten.

Die Bestimmung der Primärstruktur einer DNA-Sequenz bezeichnet man als Sequenzierung. Der Prozess des Auslesens muss bei längeren DNA-Sequenzen parallelisiert werden. Mittels der so genannten Shotgun Sequenzierung kann eine DNA-Sequenz nahezu zufällig in kleine Fragmente zertrümmert werden, wobei allerdings sowohl die Reihenfolge der Fragmente wie auch deren Orientierung in der ursprünglichen Sequenz verloren geht. Das Problem, aus den sequenzierten Fragmenten die ursprüngliche große Sequenz zu rekonstruieren, das so genannte Fragment Assembly Problem, ist hier nicht lösbar, da jede mögliche Anordnung gleich gut die richtige sein könnte.

Dem kann begegnet werden, indem vorab genügend viele Kopien der zu sequenzierenden DNA-Sequenz geschaffen werden. Zertrümmert man diese Kopien unabhängig voneinander, so werden die verschiedenen Fragmente einander überlappen. So erhält man Anhaltspunkte für die ursprüngliche Sequenz. Kürzeste Superstrings sind nun gute Ausgangspunkte für das Fragment Assembly, denn in einem kurzen Superstring werden Überlappungen zwischen Fragmenten bestmöglich ausgenutzt.

2.2 Superstrings und Kreisüberdeckungen

Zu einer gegebenen Menge von Strings gibt es stets unendlich viele Superstrings. Das folgende Lemma benennt eine Menge von *ernstzunehmenden Kandidaten*, unter denen sich der optimale Superstring befinden muss.

Lemma 2.1 *Sei $S := \{s_1, \ldots, s_n\}$ eine Menge von Strings. Dann existiert eine Permutation $\pi \in \mathcal{S}_n$, so dass*

$$S_\pi := prf(s_{\pi(1)}, s_{\pi(2)}) \circ prf(s_{\pi(2)}, s_{\pi(3)}) \circ \ldots \circ prf(s_{\pi(n-1)}, s_{\pi(n)}) \circ s_{\pi(n)}$$

ein optimaler Superstring der Menge S ist.

Beweis: Zunächst verifizieren wir, dass S_π ein Superstring der Menge S ist: Offensichtlich endet S_π mit dem String $s_{\pi(n)}$. Wir halten weiter fest, dass $s_{\pi(i)}$ ein Präfix von

$$prf(s_{\pi(i)}, s_{\pi(i+1)}) \circ s_{\pi(i+1)}$$

ist. Damit ist der String $s_{\pi(n-1)}$ in S_π enthalten, und zwar beginnend an der Stelle $prf\,(s_{\pi(n-1)}, s_{\pi(n)})$. Wir argumentieren induktiv weiter: Taucht der String $s_{\pi(i+1)}$ beginnend an der Stelle $prf\,(s_{\pi(i+1)}, s_{\pi(i+2)})$ in S_π auf, so ist, da $s_{\pi(i)}$ ein Präfix von $prf(s_{\pi(i)}, s_{\pi(i+1)}) \circ s_{\pi(i+1)}$ ist, auch der String $s_{\pi(i)}$ in S_π enthalten, und zwar beginnend an der Stelle $prf\,(s_{\pi(i)}, s_{\pi(i+1)})$.

Weiter ist zu zeigen, dass einer der Strings S_π ein kürzester Superstring ist. Sei S' ein beliebiger kürzester Superstring der Menge S. Sei $\pi \in \mathcal{S}_n$ so gewählt, dass das erste Auftreten von $s_{\pi(i)}$ in S' vor dem ersten Auftreten von $s_{\pi(i+1)}$ liegt. Da sich Strings gegenseitig nicht enthalten, endet ein String, der vor einem zweiten in S' beginnt, auch vor diesem. Falls S' Zeichen enthält, die nicht zum ersten Auftreten eines Strings s_i gehören, so könnten diese entfernt werden, die Superstring Eigenschaft bliebe erhalten und S' wäre nicht optimal gewesen. Beginnt $s_{\pi(i+1)}$ in S' mehr als $|prf(s_{\pi(i)}, s_{\pi(i+1)})|$ Zeichen nach $s_{\pi(i)}$, so kann $s_{\pi(i+1)}$ mit allen nachfolgenden Strings weiter über $s_{\pi(i)}$ geschoben werden und S' kann verkleinert werden. Folglich ist $S' = S_\pi$. □

Jeder Algorithmus für das Problem kürzester Superstrings wird also – zumindest implizit – für eine gegebene Stringmenge eine Permutation π berechnen.

Ein Superstring S_π zu einer gegebenen Permutation π hat dann die Länge

$$\begin{aligned} |S_\pi| &= \sum_{i=1}^{n-1} |prf\,(s_{\pi(i)}, s_{\pi(i+1)})| + |s_{\pi(n)}| \\ &= \sum_{i=1}^{n-1} \big(|s_{\pi(i)}| - (s_{\pi(i)}, s_{\pi(i+1)})\big) + |s_{\pi(n)}| \\ &= \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n-1} (s_{\pi(i)}, s_{\pi(i+1)}). \end{aligned}$$

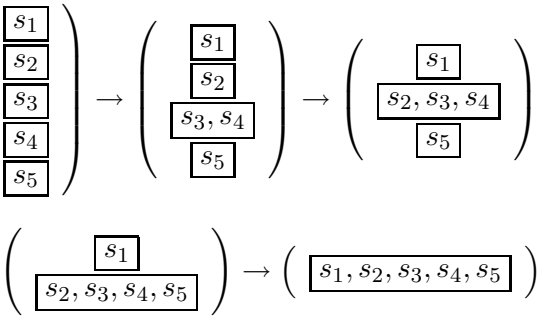
Die Greedy-Heuristik wählt die Permutation, indem in jedem Schritt zwei Strings als aufeinanderfolgend fixiert werden. Um die Notation möglichst einfach zu halten, nehmen wir stets ohne Beschränkung der Allgemeinheit an, dass die Greedy-Heuristik die Identität als Permutation wählt, die Strings werden also in die Reihenfolge $s_1, s_2, \ldots, s_n$ gebracht. Der Freiheitsgrad festzulegen, wann während des Ablaufs die Strings s_i und s_{i+1} verbunden werden, bleibt davon unberührt.

Jede der $(n-1)!$ verschiedenen Reihenfolgen, in denen diese Verbindungen erfolgen können, offenbart Teilinformationen über die Größenbeziehungen zwischen den uns unbekannten Überlappungen: Wählt die Greedy-Heuristik das Paar (s_i, s_{i+1}) zu einem Zeitpunkt, an dem die Wahl von (s_j, s_k) auch möglich war, so kann

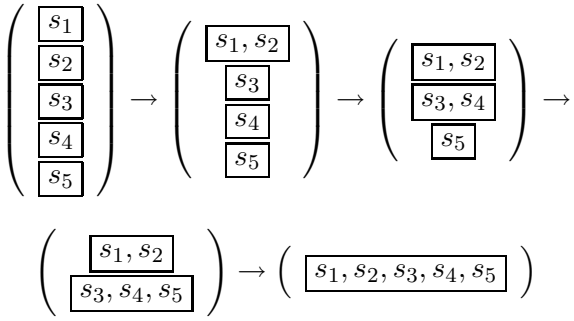
$$(s_i, s_{i+1}) \geq (s_j, s_k)$$

gefolgert werden. Die durch die Reihenfolge der Wahlen bekannt werdende partielle Ordnung auf den unbekannten Überlappungen nennen wir *Greedy-ordnung.*

Beispiel 2: Eine mögliche Vorgehensweise der Greedy-Heuristik:



Dasselbe Ergebnis mit einer anderen Greedyordnung:



Definition 2.3 (Greedyordnung) *Sei S eine Menge von n Strings, die von der Greedy-Heuristik in die Reihenfolge $s_1, \ldots, s_n$ gebracht werden.*

1. *Die Greedyordnung der Menge S ist die partielle Ordnung $\leq_G$ auf den geordneten Paaren $(i, j) \in \{1, \ldots, n\}^2$ mit*

$$(a, b) \leq_G (c, d) \iff \begin{array}{l}(c, d) \text{ wird von der Greedy Heuristik} \\ \text{gewählt, während } (a, b) \text{ für die Greedy} \\ \text{Heuristik eine legale Wahl ist.}\end{array}$$

Haben zu einem Zeitpunkt mehrere Paare maximale Überlappung, so häng die partielle Ordnung nicht nur von den Strings, sondern auch von der Art des Tie-Breakings ab. Die Greedyordnung der Stringmenge S repräsentieren wir durch die Reihenfolge der $n-1$ Paaren $(i, i+1)$ für $1 \leq i \leq n-1$. Steht das Paar $(i, i+1)$ an r-ter Stelle, so wird in der r-ten Schleifeniteration der Greedy-Heuristik s_{i+1} zum Nachfolger von s_i bestimmt, die Überlappung (s_i, s_{i+1}) ist also zu dem Zeitpunkt maximal unter allen Überlappungen, die noch wählbar waren.

2. *Eine Greedyordnung heißt linear, wenn die Greedy-Heuristik in ihrem Verlauf niemals zwei zusammengesetzte Strings der Menge S verknüpft. Ein String heißt zusammengesetzt, wenn er mindestens zwei der ursprünglichen Eingabestrings beinhaltet.*

3. *Eine Greedyordnung heißt bilinear, wenn die Greedy-Heuristik höchstens einmalig, und zwar in der letzten Schleifeniteration, zwei zusammengesetzte Strings verknüpft.*

Eine lineare Greedyordnung liegt also vor, wenn die Greedy-Heuristik mit einer beliebigen Verbindung (s_i, s_{i+1}) beginnt und zu jedem späteren Zeitpunkt, wenn $s_j, \ldots, s_k$ bereits zu einem partiellen Superstring verbunden sind, entweder (s_{j-1}, s_j) oder (s_k, s_{k+1}) wählt. Bei einer bilinearen Greedy-Ordnung können zwei zusammengesetzte Strings existieren, die unabhängig voneinander anwachsen und dann im letzten Schritt verschmolzen werden.

Die beiden Greedyordnungen des obigen Beispiels sind:

$$(3,4),(2,3),(4,5),(1,2)$$

und

$$(1,2),(3,4),(4,5),(2,3).$$

Also ist die erste Greedyordnung linear und die zweite bilinear.

Der Nachweis der 4-Approximativität der Greedy-Heuristik in (BLTY94) basiert auf der Tatsache, dass eine leicht modifizierte Greedy-Heuristik (Algorithmus 2.2) optimale Kreisüberdeckungen auf Strings findet.

Definition 2.4 (Kreisüberdeckung) *Sei $S = \{s_1, \ldots, s_n\}$ eine Menge von Strings. Eine Menge von Strings $C = \{c_1, \ldots, c_k\}$ ist eine Kreisüberdeckung von S, wenn jeder String s_i Teilstring eines Strings c_j^l für genügend großes l ist. Wir definieren $|C| = \sum_{j=1}^{k} |c_j|$ als Größe der Kreisüberdeckung.*

Eine Kreisüberdeckung kann durch eine Bijektion τ beschrieben werden, wobei τ jedem String einen eindeutigen Nachfolger in der Kreisüberdeckung zuordnet. Wir werden auch die Menge der von einer Kreisüberdeckung *ausgenutzten* Überlappungen $\{(i, \tau(i)\}$ als Kreisüberdeckung bezeichnen.

Die Größe von C_τ schreibt sich dann wie folgt:

$$|C_\tau| = \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n} (s_i, s_{\tau(i)}).$$

Jeder Superstring einer Menge S führt kanonisch zu einer trivialen Kreisüberdeckung: Zu einem Superstring S_π der Menge S sei S_π° der String, der entsteht, wenn die letzten $(s_{\pi(n)}, s_{\pi(1)})$ Buchstaben von S_π gelöscht werden. Dann ist $|S_\pi^\circ| = |S_\pi| - (s_{\pi(n)}, s_{\pi(1)})$ und jeder String s_i ist in $(S_\pi^\circ)^l$ für ein hinreichend großes l enthalten.

Die modifizierte Greedy-Heuristik

1. EINGABE: Eine Menge S von Strings
2. $C = \emptyset$
3. Solange $S \neq \emptyset$ wiederhole:
   (a) Wähle $a, b \in S$, so dass (a, b) maximal ist.
   (b) Falls $a \neq b$:
      i. Berechne $c = prf(a, b) \circ b$.
      ii. Setze $S := (S \setminus \{a, b\}) \cup \{c\}$.
   (c) Falls $a = b$:
      i. Setze $C := C \cup \{p_a\}$.
      ii. Setze $S := S \setminus \{a\}$.
4. AUSGABE: Die Menge C.

Algorithmus 2.2: Die Greedy-Heuristik für optimale Kreisüberdeckungen

Für den von der Greedy-Heuristik gefundenen Superstring S_G ergibt sich folglich $|S_G^\circ| = |S_G| - (s_n, s_1)$. Für einen optimalen Superstring S_{opt} und eine optimale Kreisüberdeckung C_{opt} gilt

$$|C_{opt}| \leq |S_{opt}^\circ| \leq |S_{opt}|.$$

Daher ist die in (WS03) formulierte Fassung der Greedy-Conjecture, nämlich

$$|S_G| \leq |S_{opt}| + |C_{opt}|, \tag{2.1}$$

eine schärfere Abschätzung als die klassische Greedy-Conjecture

$$|S_G| \leq 2 \cdot |S_{opt}|. \tag{2.2}$$

Für bilineare Greedyordnungen beweisen wir diese stärkere Variante (2.1). Gegenbeispiele sind uns aber auch für allgemeine Greedyordnungen nicht bekannt. Dieses Ergebnis folgt, wie wir später zeigen werden, aus dem stärkeren Theorem 2.1.

Theorem 2.1 *Sei S eine Menge von Strings mit einer bilinearen Greedyordnung. Seien weiter C_1 und C_2 zwei Kreisüberdeckungen der Menge S mit der folgenden Expansionseigenschaft: Es gibt keine Teilmenge $S' \subset S$, für die sowohl C_1 als auch C_2 eine Kreisüberdeckung enthält. Dann gilt*

$$|S_G^\circ| \leq |C_1| + |C_2|.$$

Sind τ_1 und τ_2 die zwei Bijektionen, die jedem String s_i seinen Nachfolger $s_{\tau_1(i)}$ bzw. $s_{\tau_2(i)}$ in C_1 bzw. C_2 zuweisen, dann ist die Expansionseigenschaft genau dann erfüllt, wenn für jede nichttriviale Teilmenge $\emptyset \subset X \subset \{1, \ldots, n\}$ ein $x_0 \in X$ existiert mit $\tau_1(x_0) \notin X$ oder $\tau_2(x_0) \notin X$. Der Beweis von Theorem 2.1 wird in den Abschnitten 2.3 und 2.4 entwickelt.

Aus Theorem 2.1 folgt sofort, dass für bilineare Greedyordnungen die Abschätzung

$$|S_G^\circ| \leq |S_{opt}^\circ| + |C_{opt}| \tag{2.3}$$

gilt: Hierfür genügt es festzuhalten, dass die verlangte Expansionseigenschaft immer gegeben ist, wenn eine der Kreisüberdeckungen ein einziger Kreis ist. In Abschnitt 2.4.4 wird gezeigt, wie aus Ungleichung 2.3 das folgende Korollar folgt.

Korollar 2.1 *Sei S eine Menge von Strings mit einer bilinearen Greedyordnung. Dann gilt*

$$|S_G| \leq |S_{opt}| + |C_{opt}|.$$

2.3 Greedyordnungen und bedingte lineare Ungleichungen

Für den Beweis von Theorem 2.1 bringen wir die zu zeigende Abschätzung $|S_G^\circ| \leq |C_1| + |C_2|$ zunächst in die ausführliche Form

$$\sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n-1} (s_i, s_{i+1}) - (s_n, s_1) \leq 2 \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n} (s_i, s_{\tau_1(i)}) - \sum_{i=1}^{n} (s_i, s_{\tau_2(i)}),$$

was wir durch Umordnung der Terme in die Form

$$\sum_{i=1}^{n} (s_i, s_{\tau_1(i)}) + \sum_{i=1}^{n} (s_i, s_{\tau_2(i)}) \leq \sum_{i=1}^{n} |s_i| + \sum_{i=1}^{n-1} (s_i, s_{i+1}) + (s_n, s_1) \tag{2.4}$$

überführen. Die rechte Seite der Ungleichung ist nun von den τ_i unabhängig.

Wesentlich ist nun, dass die auftretenden Terme nicht beliebige Werte annehmen können, sondern Längen bzw. Überlappungen von Strings repräsentieren. Daraus ergeben sich Beziehungen zwischen diesen Größen, die für die Abschätzung benutzt werden können. Zunächst gibt es einige wenige *unbedingte* Ungleichungen, die nicht von der Greedyordnung abhängen: Beispielsweise ist eine Überlappung stets kleiner als jeder der beiden beteiligten Strings. Es kann also stets $|s_i| > (s_i, s_j)$ zur Abschätzung verwendet werden.

Selbstredend muss auch das Vorgehen der Greedy-Heuristik ausgenutzt werden. Hat die Greedy-Heuristik die Überlappung (s_i, s_{i+1}) zu einem Zeitpunkt gewählt, als auch die Wahl (s_j, s_k) möglich war, so kann $(s_i, s_{i+1}) \geq (s_j, s_k)$ gefolgert werden. Solche Ungleichungen, die von der Greedyordnung

abhängen, nennen wir *bedingte Ungleichungen*. Im Verlauf dieses Kapitels werden wir bekannte Ungleichungen sammeln und neue Ungleichungen vorstellen und beweisen.

Entscheidend ist hierbei die geschickte Auswahl der heranzuziehenden Ungleichungen für eine gegebene Instanz. Als Instanz bezeichnen wir im Folgenden eine Greedyordnung und ein Paar von Kreisüberdeckungen. Die Aufgabe, für eine gegebene Instanz die passenden Ungleichungen zur Abschätzung heranzuziehen, kann als ein Brettspiel visualisiert werden, das als nächstes eingeführt wird.

Für den Rest dieses Kapitels wird uns das folgende Paar von Kreisüberdeckungen als durchgehendes Beispiel begleiten:

$$\begin{array}{lllll} \tau_1(1)=1 & \tau_1(2)=4 & \tau_1(3)=2 & \tau_1(4)=3 & \tau_1(5)=5 \\ \tau_2(1)=4 & \tau_2(2)=5 & \tau_2(3)=1 & \tau_2(4)=3 & \tau_2(5)=2 \end{array}$$

Beachte, dass die Expansionseigenschaft aus Theorem 2.1 erfüllt ist.

2.3.1 Das Mäusespiel

Das Mäusespiel für n Strings wird auf einem Brett bestehend aus $n \times n$ Zellen gespielt (vergl. Abb. 2.1). Die rechteckige Fläche innerhalb der Zelle (i, j) repräsentiert die Überlappung (s_i, s_j). Die Diagonalzelle (i, i) enthält zusätzlich eine ellipsenförmige Fläche, die die Stringlänge $|s_i|$ repräsentiert. Die Stringlänge $|s_i|$ ist von der Eigenüberlappungen (s_i, s_i), die durch das Rechteck in (i, i) repräsentiert wird, zu unterscheiden.

Wir platzieren $2n$ Mäuse auf dem Brett, wobei ihre Positionen den Termen der linken Seite von Ungleichung 2.4 entsprechen. Genauer: die Startpositionen der Mäuse werden durch die Bijektionen τ_1, τ_2 der beiden Kreisüberdeckungen geliefert. Für jedes i mit $1 \leq i \leq n$ setzen wir je eine Maus auf die Rechtecke in den Zellen $(i, \tau_1(i))$ und $(i, \tau_2(i))$. Ist $\tau_1(i) = \tau_2(i)$, so enthält das Rechteck der betreffenden Zelle zwei Mäuse.

Die grau unterlegten Flächen entsprechen den Termen der rechte Seite von Ungleichung (2.4): Die Stringlängen, die Überlappungen der Nebendiagonalen und die Überlappung (s_n, s_1). Wir nennen diese grau unterlegten Flächen *Löcher*, da sie die Zielpositionen unserer Mäuse sind. Die ellipsenförmig dargestellten Löcher innerhalb der Diagonalzellen nennen wir Diagonallöcher, die übrigen Löcher heißen Greedylöcher. In Abbildung 2.1 ist die unserem Beispiel entsprechende Startkonfiguration dargestellt.

Ziel ist es nun, die Positionen der Mäuse durch eine Folge legaler Züge so zu verändern, dass die Mäuse in den Löchern enden, wobei ein Loch letztlich genau eine Maus beherbergen kann. Legal soll ein Zug dann sein, wenn sichergestellt ist, dass die Summe der Werte der Felder, in denen sich die Mäuse befinden, nicht kleiner wird. Gelingt es eine solche Folge von Zügen zu finden, so ist Gleichung (2.4) für die vorliegende Instanz bewiesen.

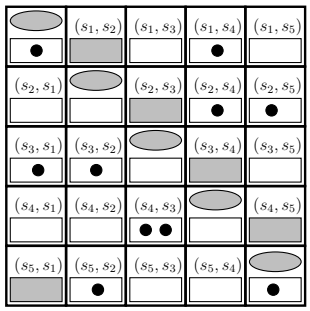


Abbildung 2.1: Beispiel einer Anfangskonfiguration

In den nächsten Abschnitten werden wir also verschiedene Ungleichungen benennen und aufzeigen, welchen legalen Zugfamilien sie in unserem Spiel entsprechen.

2.3.2 Unbedingte lineare Ungleichungen

Wir beginnen mit einer ersten unbedingten linearen Ungleichungsfamilie.

Lemma 2.2 (Inklusionen) *Für nichtleere Strings a und b gilt:*

$$(a, b) < |a|,\ (a, b) < |b| \text{ und } (a, a) = |a| - |p_a| < |a|.$$

Beweis: Nach Definition sind Überlappungen Längen *echter* Suffixe bzw. Präfixe der betreffenden Strings. Also sind sie strikt kleiner als die Längen der beiden beteiligten Strings. □

Wenn wir eine Maus von einer Zelle (i, j) mit $i \neq j$ in das Loch der Diagonalzelle (i, i) oder (j, j) bewegen, so wird ihr Wert nicht kleiner. Ein solcher Zug ist also stets legal. Wir nennen solche Züge *Diagonaleinzüge*.

Die dritte Ungleichung von Lemma 2.2 erlaubt es uns innerhalb einer Diagonalzelle eine Maus vom Rechteck, das die Eigenüberlappung (s_i, s_i) repräsentiert, in das Diagonalloch der Zelle zu verschieben. Da die Differenz aus Stringlänge $|s_i|$ und Eigenüberlappung (s_i, s_i) genau der Länge der Periode $|p_{s_i}|$ entspricht, nennen wir einen solchen Zug einen *Periodenverzicht*, die Länge der Periode wird aufgegeben. In 2.2 sehen wir je eine Ausführung der beiden Züge.

Als nächstes benennen wir eine einfache Version der Monge-Ungleichung (Mo81).

Lemma 2.3 (Erste Monge-Ungleichung) *Sind a, b und c Strings, dann gilt die Ungleichung*

$$(a, c) + (b, a) \leq |a| + (b, c). \tag{2.5}$$

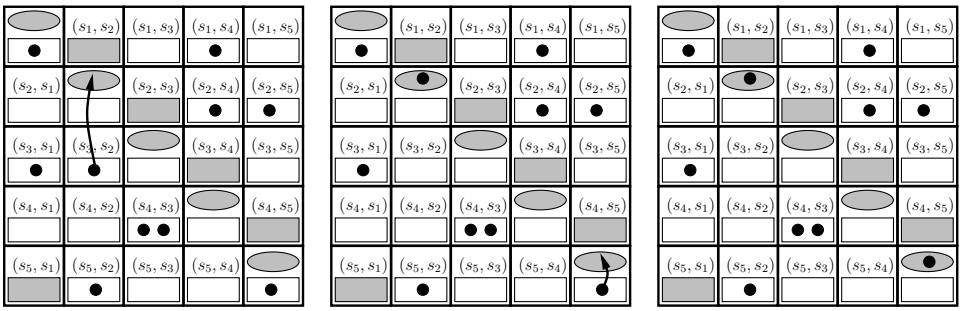
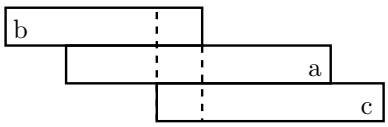


Abbildung 2.2: Die Anwendung eines Diagonaleinzugs und eines Periodenverzichts. Die Ungleichungen $(s_3, s_2) \leq |s_2|$ bzw. $(s_5, s_5) \leq |s_5|$ rechtfertigen die Züge. Durch ihre Anwendung verringert sich der *Wert der beteiligten Mäuse* nicht, genauer, der Wert der besetzten Positionen gewichtet mit der Anzahl ihrer Mäuse verringert sich nicht.

Beweis: Falls $|a| \geq (a, c) + (b, a)$ gilt, ist die Ungleichung trivialerweise erfüllt. Andernfalls greift das folgende Diagramm:



Die Summe $(b, a) + (a, c)$ ist nach Fallannahme größer als die Länge von a. Also überschneiden sich b und c indirekt. Das Intervall zwischen den gestrichelten vertikalen Linien bezeichnet einen echten Suffix von b, der auch ein echter Präfix von c ist. Seine Länge ist $(b, a) + (a, c) - |a|$. Also ist (b, c) mindestens von dieser Länge und wir erhalten $(b, c) \geq (b, a) + (a, c) - |a|$. □

Die Ungleichung aus Lemma 2.3 rechtfertigt eine Familie von Zügen mit jeweils zwei beteiligten Mäusen. Von zwei Mäusen, von denen sich eine in Zelle (i, j) und eine in Zelle (k, i) befindet (mit $k \neq i$ und $j \neq i$), kann eine in das Diagonalloch in (i, i) gezogen werden, während die andere in die Zelle (k, j) wandert. (Sollte $k = j$ gelten, landet die Maus auf dem Rechteck für die Eigenüberlappung in Zelle (k, k), nicht im dortigen Diagonalloch.)

Beachte, dass die zweitgenannte Bewegung nicht für sich genommen legal ist: Die Maus, die nach (k, j) wandert, könnte an Wert verlieren. Dieser Verlust würde aber nach Lemma 2.3 durch den Wertzuwachs der anderen Maus mindestens kompensiert. Einen solchen Zug nennen wir einen *Diagonal-Monge in* (i, i). In Abb. 2.3 setzen wir unser Spiel mit zwei aufeinanderfolgenden Anwendungen eines Diagonal-Monges fort.

Man bemerke, dass die bis hierher eingeführten Züge völlig unabhängig von der Greedyordnung sind. Da es uns schwerlich gelingen wird, Aussagen

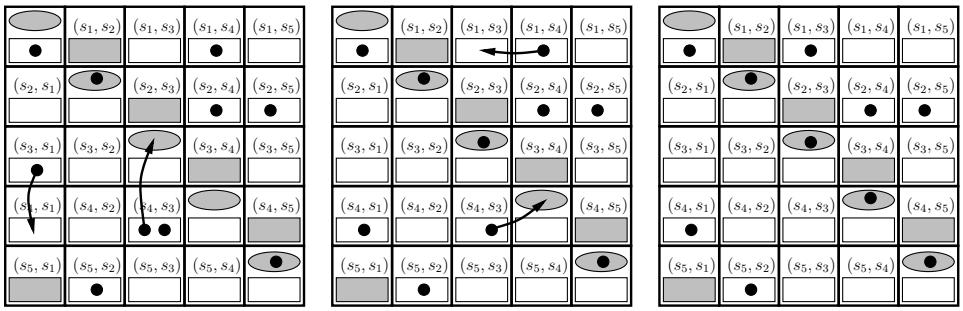


Abbildung 2.3: Die Anwendung zweier Diagonal-Monges in $(3, 3)$ bzw. $(4, 4)$. Die rechtfertigenden Ungleichungen sind $(s_4, s_3) + (s_3, s_1) \leq |s_3| + (s_4, s_1)$ bzw. $(s_4, s_3) + (s_1, s_4) \leq |s_4| + (s_1, s_3)$.

über die Greedy-Heuristik zu beweisen, ohne auf ihr *gieriges* Verhalten Bezug zu nehmen, werden die Ungleichungen der nächsten Abschnitte bedingte Ungleichungen sein, deren Prämisse vor der Verwendung der Ungleichung auf Gültigkeit überprüft werden muss.

2.3.3 Verwendung von Greedyordnungen

Zu einer Instanz für unser Spiel gehört neben den beiden Kreisüberdeckungen eine Greedyordnung. Nehmen wir für unser Beispiel die Greedyordnung, die durch die Permutation $(3, 4), (2, 3), (1, 2), (4, 5)$ beschrieben wird. Das heißt, die Greedy-Heuristik hat die 5 Strings wie folgt verarbeitet:

$$\left(\begin{array}{c}\boxed{s_1}\\ \boxed{s_2}\\ \boxed{s_3}\\ \boxed{s_4}\\ \boxed{s_5}\end{array}\right) \rightarrow \left(\begin{array}{c}\boxed{s_1}\\ \boxed{s_2}\\ \boxed{s_3, s_4}\\ \boxed{s_5}\end{array}\right) \rightarrow \left(\begin{array}{c}\boxed{s_1}\\ \boxed{s_2, s_3, s_4}\\ \boxed{s_5}\end{array}\right) \rightarrow$$

$$\left(\begin{array}{c}\boxed{s_1, s_2, s_3, s_4}\\ \boxed{s_5}\end{array}\right) \rightarrow \left(\begin{array}{c}\boxed{s_1, s_2, s_3, s_4, s_5}\end{array}\right)$$

Die Reihenfolge, in der die Paare $(i, i + 1)$ auftauchen, liefert uns Informationen über die uns unbekannten Überlappungen. Wählt die Greedy-Heuristik das Paar $(i, i + 1)$, so ist die Überlappung (s_i, s_{i+1}) mindestens so groß wie die Überlappung (s_j, s_k), falls die Verknüpfung der Strings s_j und s_k zu dem Zeitpunkt noch eine legale Wahl war. Abb. 2.4 zeigt die partielle Ordnung, mit der wir in unserem exemplarischen Spiel arbeiten können. Die Überlappung (s_3, s_4) ist größer als alle übrigen. Durch die Verschmelzung von s_3 und s_4 sind alle Überlappungen oberhalb der ersten gestrichelten Linie im weiteren Verlauf für die Greedy-Heuristik nicht mehr wählbar, da

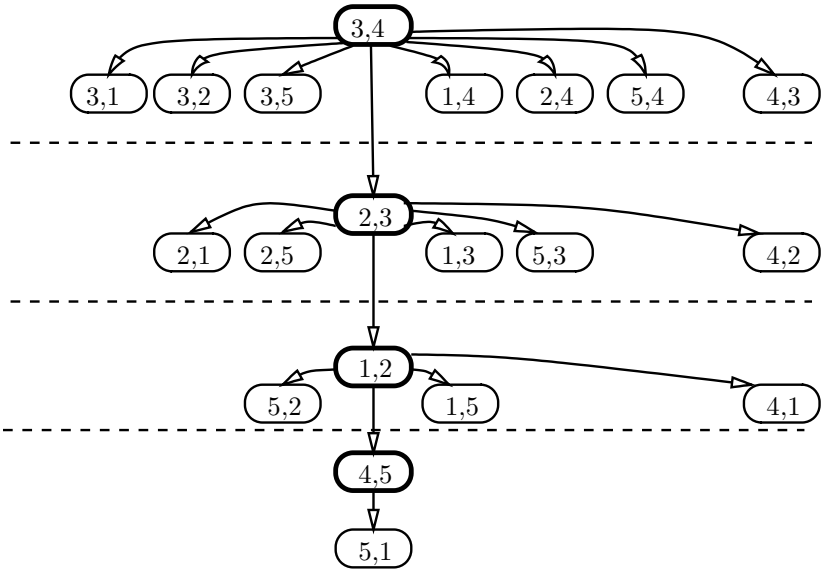


Abbildung 2.4: Ein Beispiel einer Greedyordnung mit resultierenden Rängen.

sie entweder s_3 einen weiteren Nachfolger zuordnen würden, oder s_4 einen weiteren Vorgänger zuordnen würden, oder einen Kreis schließen würden.

Durch die Wahl von (s_2, s_3) im zweiten Schritt fallen wieder mögliche Verknüpfungen weg. Im letzten Schritt schließlich hatte die Heuristik nur noch die beiden partiellen Superstrings (s_1, s_2, s_3, s_4) und s_5 und konnte sich nur noch zwischen den Alternativen (s_4, s_5) und (s_5, s_1) entscheiden.

Allgemein weisen wir jeder Zelle mit Ausnahme der Diagonalzellen einen Rang von 1 bis $n-1$ wie folgt zu.

Definition 2.5 *Wird die Überlappung (s_i, s_{i+1}) in der r-ten Iteration durch die Greedy-Heuristik ausgewählt, so erhält die Zelle $(i, i+1)$ den Rang $n-r$. Desweiteren erhält eine Zelle (a, b) mit $a \neq b \neq a+1$ den Rang $n-r$, wenn die Verknüpfung der Strings s_a und s_b durch die erfolgte Wahl in Iteration r unmöglich wird.*

In unserem Beispiel in Abb. 2.4 teilen die gestrichelten horizontalen Linien die Überlappungen also nach Rängen von Rang 4 (oben) bis Rang 1 (unten).

In Abb. 2.5 (links) sind die Ränge für unser Spiel eingetragen. Wir wissen nun also stets, dass eine Nebendiagonalzelle $(i, i+1)$ vom Rang r einen Wert repräsentiert, der maximal unter allen Zellen mit Rang $\leq r$ ist.

Für die im Kapitel 2.4 anstehenden Korrektheitsbeweise ist es notwendig, die Verteilung der Ränge auf die Zellen etwas genauer zu studieren. Vor dem r-ten Schleifendurchlauf der Greedy-Heuristik (Algorithmus 2.1) enthält die Menge S genau $n-r+1$ Strings. Seien $i_1, \ldots, i_{n-r+1}$ und $j_1, \ldots, j_{n-r+1}$ so

gewählt, dass die Greedy-Heuristik vor Schritt r die partiellen Superstrings $(s_{i_k}, \ldots, s_{j_k})$ für $1 \leq k \leq n-r+1$ enthält. Sei $i_1 = 1 < i_2 < i_3 < \cdots < i_{n-r+1} \leq n$ und $j_k = i_{k+1} - 1$ für $k \leq n-r$ und $j_{n-r+1} = n$.

Nehmen wir an, dass die Greedy-Heuristik die beiden partiellen Superstrings $(s_{i_k}, \ldots, s_{j_k})$ und $(s_{i_{k+1}}, \ldots, s_{j_{k+1}})$ im r-ten Schritt verknüpft. Dann erhält die Zelle $(j_k, i_{k+1}) = (j_k, j_k + 1)$ den Rang $n-r$. Der Rang $n-r$ wird auch allen Zellen (j_k, i_l) für $1 \leq l \leq n-r+1$ mit $l \neq k+1$ zugewiesen, da s_{j_k} nicht mehr zum Vorgänger eines anderen Strings gemacht werden kann. Diese Zellen befinden sich alle in der Zeile j_k des Spielbretts. Im Diagramm 2.4 sind diese Überlappungen jeweils links von den Zellen der Nebendiagonalen dargestellt.

Ebenso erhält jede Zelle (j_l, i_{k+1}) für $1 \leq l \leq n-r+1$ mit $l \neq k$ den Rang $n-r$, da $s_{i_{k+1}}$ nicht mehr zum Nachfolger eines anderen Strings werden kann. Die betreffenden Zellen befinden sich in der Spalte i_{k+1}. In Abb. 2.4 sind diese Überlappungen stets rechts von den Greedyzellen dargestellt.

Eine einzelne Zelle mit Rang $n-r$ findet sich zudem noch auf der Position (j_{k+1}, i_k). Dem String $s_{j_{k+1}}$ kann von nun an nicht mehr der String s_{i_k} als Nachfolger zugewiesen werden, da ansonsten ein Kreis geschlossen wird. Dieses isolierte Feld mit Rang $n-r$ ist in Abb. 2.4 jeweils ganz rechts außen angebracht. Es liegt also stets genau ein Feld mit Rang $n-r$ weder in der Zeile noch in der Spalte der Greedyzelle vom Rang $n-r$.

Dieses isolierte Feld, das aus dem Verbot, Kreise zu schließen, resultiert, ist ein entscheidender Unterschied zwischen der Suche nach kürzesten Superstrings und der Suche nach kürzesten Kreisüberdeckungen, wie wir in Abschnitt 2.5 noch sehen werden. Bei der Formulierung von Gewinnstrategien werden wir auf diese isolierten Felder ein besonderes Augenmerk richten müssen. Deshalb halten wir mit dem nächsten Lemma die genaue Position dieser Problemfelder fest.

Lemma 2.4 *Sei $(j, j+1)$ die Greedyzelle mit Rang$(j, j+1) = r$. Sei weiterhin i mit $i < j$ und Rang$(i, i+1) < r$ maximal gewählt: falls kein solches i existiert, sei $i = 0$. Sei außerdem k mit $k > j$ und Rang$(k, k+1) < r$ minimal gewählt: falls kein solches k existiert, sei $k = n$.*

Die Zelle mit Rang r, die sich weder in Zeile j noch in Spalte $j+1$ befindet, befindet sich in Position $(k, i+1)$.

Beweis: Da alle Greedyzellen $(i+1, i+2), (i+2, i+3), \ldots, (j-2, j-1), (j-1, j)$ sowie $(j+1, j+2), \ldots, (k-1, k)$ nach Wahl von i bzw. k höheren Rang haben, werden im $n-r$-ten Schritt der Heuristik die partiellen Superstrings $(s_{i+1}, \ldots, s_j)$ und $(s_{j+1}, \ldots, s_k)$ verbunden. Die Verknüpfung (s_k, s_{i+1}) schließt von diesem Moment an einen Kreis und scheidet daher aus der Menge der möglichen Wahlen aus. Damit erhält die Zelle $(k, i+1)$ den Rang r. □

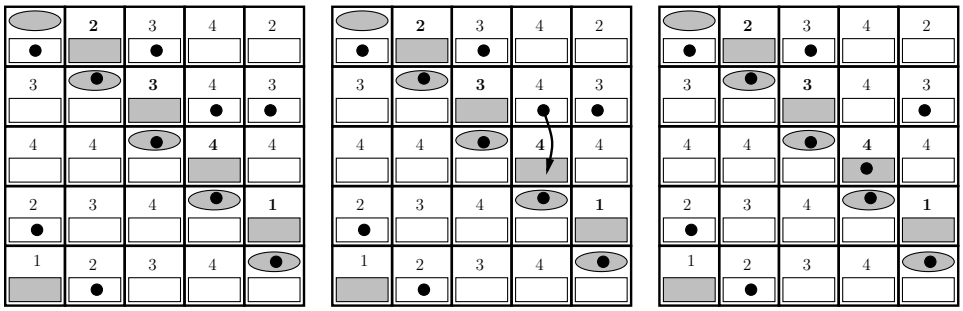


Abbildung 2.5: Die Ränge, die der angenommenen Greedyordnung entsprechen. Da das Loch (s_3, s_4) Rang 4 hat, gilt $(s_3, s_4) \geq (s_2, s_4)$.

2.3.4 Bekannte bedingte lineare Ungleichungen

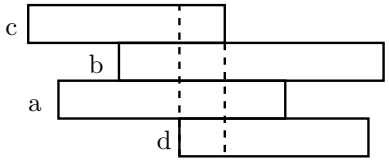
Zurück zu unserem exemplarischen Spiel: Die Greedyordnung liefert uns kanonisch eine neue Zugfamilie. Eine Maus darf von einer Zelle (j, k) in ein Loch der Nebendiagonale $(i, i+1)$ gezogen werden, wenn diese Nebendiagonalzelle mindestens den gleichen Rang hat wie das Startfeld. Einen solchen Zug nennen wir einen *Greedyeinzug*. In Abb. 2.5 wird unser exemplarisches Spiel mit einem solchen Einzug fortgesetzt.

Die nächste Ungleichung – wie bereits Lemma 2.3 – geht auf Gaspard Monge (Mo81) zurück.

Lemma 2.5 (Zweite Monge-Ungleichung) *Seien a, b, c und d Strings mit $(a, b) \geq (a, d)$ und $(a, b) \geq (c, b)$. Dann gilt*

$$(a, d) + (c, b) \leq (a, b) + (c, d). \tag{2.6}$$

Beweis: Der Beweis ähnelt dem von Lemma 2.3: Wenn $(a, b) \geq (a, d) + (c, b)$ gilt, folgt die Aussage sofort. Andernfalls können wir an dem folgenden Diagramm die Überlappungen $(c, b), (a, b)$ und (a, d) von oben nach unten verfolgen.



Die Summe aus (c, b) und (a, d) übertrifft (a, b) genau um die Länge des Segments zwischen den vertikalen Linien. Dieser Teil jedoch stellt einen echten Suffix von c dar, der auch ein echter Präfix von d ist. Folglich ist $(c, d) \geq (c, b) + (a, d) - (a, b)$ und die Aussage folgt. □

Den entsprechenden Zug nennen wir analog zum Diagonal-Monge einen *Greedy-Monge*, da die notwendigen Prämissen $(a, b) \geq (a, d)$ und $(a, b) \geq$

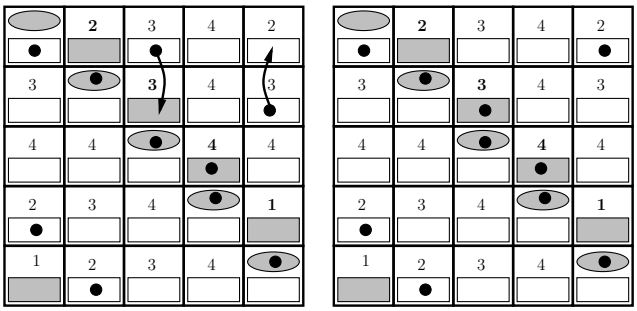


Abbildung 2.6: Ein Greedy-Monge in (s_2, s_3). Die Ränge geben an, dass (s_2, s_3) mindestens so groß wie (s_1, s_3) und (s_2, s_5) ist. Damit folgt die Ungleichung $(s_2, s_3) + (s_1, s_5) \geq (s_1, s_3) + (s_2, s_5)$ die wir als Bewegung der Mäuse umsetzen.

(c, b) im Spiel meistens dadurch verifiziert werden, dass (a, b) eine Greedyzelle ist und sowohl (a, d) wie auch (c, b) höchstens gleich großen Rang haben. Wir sprechen auch vom *Greedy-Monge in* $(i, i+1)$, wenn $(i, i+1)$ die Position der Überlappung (a, b) ist. Beachte auch hier, dass im Fall $c = d$ die Maus in Zelle (c, c) nicht ins Diagonalloch gezogen wird, sondern auf das Rechteck für die Eigenüberlappung (c, c). In Abb. 2.6 setzen wir unser Spiel mit einem Greedy-Monge fort.

Wir führen noch einige Familien von bedingten Ungleichungen ein, die wir allerdings erst im Abschnitt 2.4.4 benötigen werden.

Lemma 2.6 *Seien a, b und c Strings, dann gilt*

1. *Falls $(a, b) \geq (a, c)$, dann $(a, c) \leq (b, c) + |p_b|$,*
2. *Falls $(a, b) \geq (b, c)$, dann $(b, c) \leq (a, c) + |p_b|$,*
3. *Falls $(a, b) \geq (c, b)$, dann $(c, b) \leq (c, a) + |p_a|$,*
4. *Falls $(a, b) \geq (c, a)$, dann $(c, a) \leq (c, b) + |p_a|$.*

Beweis: Wir beweisen nur die ersten beiden Aussagen, da bei den letzten beiden lediglich die Orientierung der Paare vertauscht ist.

Alle Abschätzungen nutzen aus, dass Strings Periodizitäten an Teilstrings vererben. Desweiteren wird verwendet, dass, wenn ein String u einen x-periodischen Suffix der Länge r und v einen x-periodischen Präfix der Länge s hat, die Ungleichung $(u, v) \geq min\{s, r\} - |x|$ folgt.

Die erste Aussage des Lemma ergibt sich wie folgt: Die Prämisse $(a, b) \geq (a, c)$ garantiert, dass c einen p_b periodischen Präfix der Länge (a, c) besitzt. String b ist durchgehend p_b periodisch nach Definition. Also ist mit obigen Vorüberlegungen $(b, c) \geq min\{(a, c), |b|\} - |p_b| = (a, c) - |p_b|$.

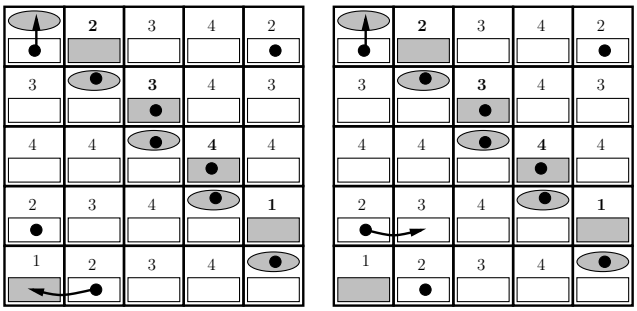


Abbildung 2.7: Zwei legale Anwendungen der Züge zu Lemma 2.6 (dritte bzw. vierte Ungleichung). Die Prämissen können anhand der Ränge verifiziert werden. Es ist stets $a = s_1$ und $b = s_2$. Im linken Diagramm ist $c = s_5$, im rechten Diagramm ist $c = s_4$. Die Differenz zwischen (s_1, s_1) und $|s_1|$ entsprcjt der Länge der Periode $|p_{s_1}|$ (vergl. Lemma 2.2).

Für die zweite Aussage des Lemmas liefert die Prämisse, dass a einen p_b periodischen Suffix der Länge (b, c) hat. String c hat trivialerweise einen p_b periodischen Präfix der Länge (b, c) und wir erhalten $(a, c) \geq (b, c) - |p_b|$. □

In Abb. 2.7 sind zwei mögliche Züge, die sich aus diesem Lemma ergeben, dargestellt. Allerdings führen wir keinen davon aus, da uns die Züge in eine Sackgasse führen würden.

2.3.5 Die Tripelungleichung

Wir haben nun einen Punkt erreicht, an dem das Musterspiel mit den bisher eingeführten Zügen nicht gewonnen werden kann. Auch durch einen geschickteren Einsatz der bisher vorgestellten Züge ist es nicht möglich, alle Mäuse in die Löcher zu bringen.

An dieser Stelle führen wir die Tripelungleichung (WS03) ein und zeigen auf, wie der ihr entsprechende Zug unser Musterspiel gewinnt.

Lemma 2.7 (Tripelungleichung) *Seien a, b, c, d und x Strings mit*

$$\max\{(a, x), (x, b)\} \geq (a, b), (x, d), (c, x).$$

Dann gilt

$$(a, b) + (x, d) + (c, x) \leq (a, x) + (x, b) + (c, d) + |p_x|.$$

Beweis: Wir führen einen Beweis durch Induktion nach (x, x). Für $(x, x) = 0$ haben wir $p_x = x$. Die Monge-Ungleichung (2.5) liefert uns dann

$$(x, d) + (c, x) \leq |x| + (c, d).$$

Da (a, b) nach Prämisse nicht größer ist als das Maximum von (a, x) und (x, b), folgt

$$(a, b) \leq (a, x) + (x, b)$$

und die Addition beider Ungleichungen liefert das gewünschte Ergebnis.

Sei nun die Aussage für alle x mit $(x, x) \leq k$ bereits gezeigt. Sei $(x, x) = k + 1$ und die Prämisse des Lemmas sei erfüllt.

Fall 1: $\max\{(a, x), (x, b)\} \geq (x, x)$. Sei o.B.d.A. $(a, x) \geq (x, b)$. Da (a, x) mindestens so groß ist wie (a, b) folgt

$$(x, x) + (a, b) \leq (a, x) + (x, b) \tag{2.7}$$

mit der Monge-Ungleichung (2.6), die nach Fallannahme anwendbar ist. Ebenfalls mit Monge-Ungleichung (2.5) gilt

$$(x, d) + (c, x) \leq |x| + (c, d). \tag{2.8}$$

Addiert man (2.7) und (2.8) und subtrahiert (x, x) auf beiden Seiten, erhält man die Aussage des Lemmas.

Fall 2: $(x, x) > (a, x), (x, b)$. Nach Prämisse ist (a, x) oder (x, b) mindestens so groß wie die Überlappungen $(a, b), (x, d)$ und (c, x). Mit der Fallannahme folgt, dass (x, x) strikt größer ist als $(a, b), (x, d)$ und (c, x). Damit können wir schließen:

$$\begin{aligned}
(a, x) &= (a, \langle x, x\rangle), \\
(x, b) &= (\langle x, x\rangle, b), \\
(c, x) &= (c, \langle x, x\rangle), \\
(x, d) &= (\langle x, x\rangle, d).
\end{aligned}$$

Aus der Prämisse des Lemmas folgt nun, dass

$$\max\{(a, \langle x, x\rangle), (\langle x, x\rangle, b)\} \geq (a, b), (\langle x, x\rangle, d), (c, \langle x, x\rangle)$$

gilt. Da $(\langle x, x\rangle, \langle x, x\rangle) < (x, x)$ gilt, können wir mit Induktionsannahme schließen:

$$(a, b) + (\langle x, x\rangle, d) + (c, \langle x, x\rangle) \leq (a, \langle x, x\rangle) + (\langle x, x\rangle, b) + (c, d) + |p_{\langle x,x\rangle}|$$

was äquivalent ist zu

$$(a, b) + (x, d) + (c, x) \leq (a, x) + (x, b) + (c, d) + |p_{\langle x,x\rangle}|.$$

Mit der Abschätzung $|p_{\langle x,x\rangle}| \leq |p_x|$ folgt die Aussage. □

Der Charme dieser bedingten Ungleichung, losgelöst vom Kontext des Mäusespiels, ist sicher eher subtiler Natur. Wir stellen den entsprechenden Zug zunächst in Abb. 2.8 als Diagramm dar.

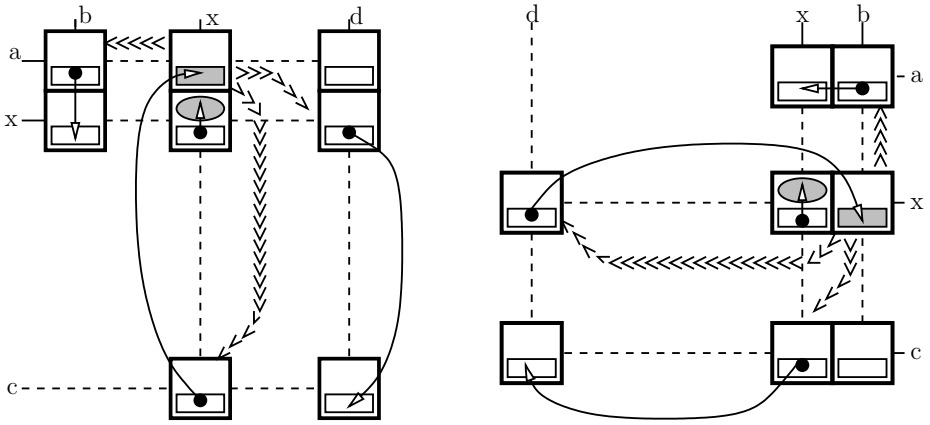


Abbildung 2.8: Der Tripelzug. Wir unterscheiden vertikale und horizontale Tripelzüge, je nachdem welche der Überlappungen (a, x) und (x, b) das Maximum stellt. Die notwendigen Dominanzen werden durch die Pfeillinien angedeutet. Da wir die benötigten Dominanzen aus der Greedyordnung gewinnen, wird die Zelle (a, x) bzw, (x, b) stets eine Zelle der Nebendiagonalen sein.

Schließlich zeigt Abb. 2.9, wie wir unser Musterspiel mit einem horizontalen Tripelzug gewinnen können. Damit haben wir Ungleichung (2.4) für diese spezielle Greedyordnung und für das vorgegebene Paar expandierender Kreisüberdeckungen bewiesen.

2.3.6 Die Sicht der linearen Programmierung

Wir haben für unsere Beispielinstanz den Nachweis von Ungleichung (2.4) durch eine geschickte Auswahl von bedingten und unbedingten Ungleichungen geführt, die alle linear in den Überlappungen und den Stringlängen sind. Wir können deshalb eine Brücke zur linearen Programmierung schlagen.

Definition 2.6 (Lineares Programm und Dualität) *Ein lineares Programm in kanonischer Darstellung hat die Form:*

$$(L) \ : \ \min \ c^T x, \text{ so dass } Ax \geq b \text{ und } x \geq 0.$$

Dabei ist $c \in \mathbb{R}^n$ der Zielvektor und $Ax \geq b$ mit $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$ das lineare Restriktionssystem. Das dazu duale Problem lautet

$$(D) \ : \ \max \ y^T b, \text{ so dass } yA \leq c \text{ und } y \geq 0.$$

(L) wird dann das primale Programm von (D) genannt.

Lemma 2.8 (Dualität) *Ist x eine Lösung für das primale Programm und y eine Lösung für das duale Programm. Sei weiter x_{opt} eine optimale Lösung*

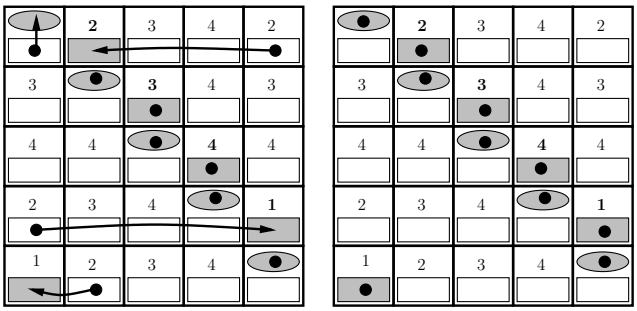


Abbildung 2.9: Ein horizontaler Tripelzug. Die Korrespondenz zu 2.8 (rechtes Diagramm) ist wie folgt: $x = s_1$, $a = d = s_5$, $b = s_2$ und $c = s_4$.

für das primale Programm und y_{opt} eine optimale Lösung für das duale Programm. Dann gilt:

1. $c^T \cdot x \geq b^T \cdot y$ *(schwache Dualität)*
2. $c^T \cdot x_{opt} = b^T \cdot y_{opt}$ *(starke Dualität).*

Für eine Instanz mit n Strings (Greedyordnung G und expandierendem Paar (C_1, C_2) von Kreisüberdeckungen konstruieren wir ein lineares Programm. Dabei nehmen wir an, dass C_i durch τ_i beschrieben ist.

- Für jede Überlappung (s_i, s_j) mit $1 \leq i, j \leq n$ legen wir eine nichtnegative Variable $x_{i,j}$ an. Desweiteren führen wir für jede Stringlänge $|s_i|$ mit $1 \leq i \leq n$ eine ebenfalls nichtnegative Variable l_i ein.

- Sei $U(n)$ die Menge der in Abschnitt 2.3.2 gesammelten unbedingten Ungleichungen. Zum Beispiel enthält $U(n)$ die Ungleichungen

  $$l_i + x_{k,j} - x_{i,j} - x_{k,i} \geq 0$$

  für einen Diagonal-Monge.

- Sei $B(n)$ die Menge der in den Abschnitten 2.3.4 und 2.3.5 hergeleiteten bedingten Ungleichungen. Zum Beispiel enthält $B(n)$ die Ungleichungen

  $$(x_{i,j} \geq x_{i,k} \wedge x_{i,j} \geq x_{h,j}) \rightarrow x_{i,j} + x_{h,k} \geq x_{i,k} + x_{h,j}$$

  für einen Greedy-Monge. Sei weiter $B_{\mathcal{G}}(n) \subset B(n)$ die Menge aller bedingten Ungleichungen, deren Bedingung durch die Greedyordnung G gesichert ist.

- In die Matrix A nehmen wir alle Ungleichungen aus $U(n) \cup B_{\mathcal{G}}(n)$ auf. Desweiteren fügen wir die Gleichung

$$\sum_{i=1}^{n} l_i - \sum_{i=1}^{n-1} x_{i,i+1} - x_{n,1} \quad = \quad 1$$

  in die Matrix auf, womit wir die Länge des zyklisch geschlossenen Superstrings auf 1 fixieren.

- Als Zielfunktion wählen wir

$$min : \sum_{i=1}^{n} 2 \cdot l_i - \sum_{i=1}^{n} x_{i,\tau_1(i)} - \sum_{i=1}^{n} x_{i,\tau_2(i)}$$

  also die Summe der Größen beider Kreisüberdeckungen.

Liefert dieses Programm für die Instanz $(\mathcal{G}, C_1, C_2)$ einen Wert ≥ 1, so bedeutet das, dass die in den verwendeten Ungleichungen modellierten Eigenschaften von Strings genügen, um auszuschließen, dass die Länge des zyklisch geschlossenen Superstrings die Summe der Längen beider Kreisüberdeckungen übertrifft. Diese Instanz ist also auf Grund der gesammelten Stringeigenschaften *unkritisch* für die Greedy-Conjecture.

In unserem Beispiel haben wir die Ungleichungen (Züge)

$$\begin{aligned}
(s_3, s_2) &\leq |s_2| \\
(s_5, s_5) &\leq |s_5| \\
(s_4, s_3) + (s_3, s_1) &\leq |s_3| + (s_4, s_1) \\
(s_4, s_3) + (s_1, s_4) &\leq |s_4| + (s_1, s_3) \\
(s_2, s_4) &\leq (s_3, s_4) \\
(s_1, s_3) + (s_2, s_5) &\leq (s_2, s_3) + (s_1, s_5) \\
(s_4, s_1) + (s_1, s_5) + (s_5, s_2) &\leq (s_4, s_5) + (s_1, s_2) + (s_5, s_1) + |p_1|
\end{aligned}$$

verwendet, die sich zu

$$\begin{aligned}
&(s_3, s_2) + (s_5, s_5) + 2(s_4, s_3) + (s_3, s_1) + (s_1, s_4) + (s_2, s_4) \\
&\qquad + \cancel{(s_1, s_3)} + (s_2, s_5) + \cancel{(s_4, s_1)} + \cancel{(s_1, s_5)} + (s_5, s_2) \\
&\leq |s_2| + |s_3| + |s_4| + |s_5| + \cancel{(s_4, s_1)} + \cancel{(s_1, s_3)} + (s_3, s_4) + (s_2, s_3) \\
&\qquad + \cancel{(s_1, s_5)} + (s_4, s_5) + (s_1, s_2) + (s_5, s_1) + |p_1|
\end{aligned}$$

aufsummieren. Addiert man schließlich auf beiden Seiten (s_1, s_1), so erhält man,

$$\underbrace{(s_1,s_1)+(s_2,s_4)+(s_3,s_2)+(s_4,s_3)+(s_5+s_5)}_{\text{vergl. } \tau_1}$$
$$+\underbrace{(s_1,s_4)+(s_2,s_5)+(s_3,s_1)+(s_4,s_3)+(s_5+s_2)}_{\text{vergl. } \tau_2}$$
$$\leq \sum_{i=1}^{5}|s_i|+\sum_{i=1}^{4}(s_i,s_{i+1})+(s_5,s_1),$$

was wir in die kompakte Form der Ungleichung (2.4) für unser Beispiel bringen können:

$$\sum_{i=1}^{5}(s_i,s_{\tau_1(i)})+\sum_{i=1}^{5}(s_i,s_{\tau_2(i)})\leq\sum_{i=1}^{5}|s_i|+\sum_{i=1}^{4}(s_i,s_{i+1})+(s_5,s_1).$$

Die Zahl der verfügbaren Ungleichungen wächst asymptotisch mit n^4, wie das folgende Lemma zeigt.

Lemma 2.9 *Werden n Strings betrachtet, so ist die Anzahl der Ungleichungen aufgeschlüsselt nach deren Art wie folgt gegeben:*

1.	*Zugfamilie*	*Anzahl*
2.	*Periodenverzicht*	n
3.	*Diagonaleinzug*	$2n^2-2n$
4.	*Diagonal-Monge*	n^3-2n^2+n
5.	*Greedyeinzug*	$\frac{1}{3}\left(n^3-4n+3\right)$
6.	*Greedy-Monge*	$\frac{1}{6}\left(2n^3-9n^2+13n-6\right)$
7.	*Tripelzug*	$\sum_{k=1}^{n-1}(2-\phi(k))\cdot(k-1)^3=\Theta(n^4)$

Dabei ist $\phi(k)\in\{0,1,2\}$ die Anzahl der zur Greedyzelle mit Rang k benachbarten Greedyzelle mit Rang größer k. Die Zahl der legalen Tripelzüge hängt also von der Greedyordnung ab, während die Mächtigkeiten aller anderen Zugfamilien lediglich von n abhängen.

Beweis:

1. Die Anzahl der Periodenverzichte entspricht trivialerweise der Zahl der Diagonalen: n.

2. In jede Diagonalzelle kann aus $(n-1)$ Feldern der Zeile und aus $(n-1)$ Feldern der Spalte eingezogen werden. Insgesamt $n\cdot(2n-2)$ Züge.

3. Für jede Diagonalzelle gibt es $(n-1)$ mögliche Startpositionen der Maus in der Zeile und $(n-1)$ Startpositionen für die Maus in der Spalte. Diese können frei kombiniert werden. Daraus ergeben sich $n \cdot (n-1)^2$ Züge.

4. Die Greedyzelle $(i, i+1)$ mit Rang k wird von der Heuristik gewählt, wenn noch $k+1$ Strings in der Menge S sind. Diese $k+1$ Strings bilden $k \cdot (k+1)$ geordnete Paare. Also dominiert die Greedyzelle vom Rang k genau $k \cdot (k+1) - 1$ andere Felder. Dies ergibt in Summe

$$\sum_{k=1}^{n-1} (k \cdot (k+1) - 1) = \ldots = \frac{1}{3}\left(n^3 - 4n + 3\right)$$

   Greedyeinzüge.

5. Die Greedyzelle mit Rang k dominiert außer sich selbst $(k-1)$ Felder seiner Spalte und $(k-1)$ Felder seiner Zeile. Diese können frei zu Startfeldern eines Greedy-Monges kombiniert werden. Wir erhalten also

$$\sum_{k=1}^{n-1} (k-1)^2 = \ldots = \frac{1}{6}\left(2n^3 - 9n^2 + 13n - 6\right)$$

   Greedy-Monges.

6. Betrachten wir die Zahl der vertikalen Tripelzüge, die durch die Greedyzelle vom Rang k ermöglicht werden. Sei diese Greedyzelle $(i, i+1)$. Die Greedyzelle dominiert außer sich selbst $(k-1)$ Felder der Zeile i und $(k-1)$ Felder der Spalte $i+1$. Für einen vertikalen Tripelzug ist zudem erforderlich, dass ein Feld der Zeile $i+1$ dominiert wird. Sollte sich auf $(i+1, i+2)$ eine Greedyzelle mit Rang größer k befinden, so existiert ein solches Feld nicht. Hat die Greedyzelle auf $(i+1, i+2)$ kleineren Rang (oder ist $i = n-1$), so dominiert $(i, i+1)$ genau k Felder dieser Zeile, wobei jedoch eines davon in derselben Spalte liegt wie das in Zeile i gewählte Feld. In diesem Fall existieren also $(k-1)^3$ vertikale Tripelzüge für $(i, i+1)$. Für horizontale Tripelzüge ist das Argument analog.

□

Für unsere Zwecke ist es glücklicherweise nicht nötig, das Optimum unseres linearen Programms zu bestimmen. Um eine Instanz als unkritisch für die verschärfte Greedy-Conjecture zu erkennen, reicht es festzustellen, dass das Optimum ≥ 1 ist. Aufgrund der schwachen Dualität genügt es also, eine Lösung mit Wert 1 für das duale Programm anzugeben. Das duale Programm hat die folgende Form und ist damit eine Beschreibung des Mäusespiels:

- Für jede Ungleichung aus $U(n) \cup B_{\mathcal{G}}(n)$ (also für jeden ausführbaren Zug) entsteht eine nicht negative Variable. Wir nennen die dem i-ten Zug zugeordnete Variablen z_i. Wir beabsichtigen die Interpretation, dass z_i angibt, wie häufig der i-te Zug ausgeführt wird. Desweiteren erhalten wir aus der Gleichung, die die Länge des Greedy-Superstrings auf 1 fixiert, eine Variable w, die auch negative Werte annehmen darf.

- Die Zielfunktion des dualen Programms ist sehr einfach, da der Vektor b unseres primalen Programms nur einen von Null verschiedenen Eintrag hat. Unsere Zielfunktion lautet $max : w$.

- Für jede der Variablen l_i und die $x_{i,j}$ des primalen Programms entsteht jeweils eine Restriktion, die den Spalten der Matrix A entsprechen. So ergibt sich für die primale Variable l_i :

$$\sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{ins Diagonalloch (i,i)}}} z_j \; + \; w \leq 2.$$

  Da uns nur die Existenz einer Lösung mit Zielwert mindestens 1 interessiert, setzen wir $w = 1$ und erhalten als Restriktion,

$$\sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{ins Diagonalloch (i,i)}}} z_j \leq 1,$$

  was der Forderung entspricht, dass höchstens eine Maus in das Diagonalloch in (i, i) gezogen werden darf. Für die Überlappungsvariablen, die im zyklisch geschlossenen Greedy-Superstring auftreten (also die $x_{i,i+1}$ und $x_{n,1}$) ergeben sich die Restriktionen

$$\sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{nach (i,i+1)}}} z_j \; - \sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{aus (i,i+1) heraus}}} z_j - w \leq 0,$$

  bzw.

$$\sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{nach (n,1)}}} z_j \; - \sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{aus (n,1) heraus}}} z_j - w \leq 0.$$

  Setzen wir auch hier $w = 1$ so entsteht die Forderung, dass durch die ausgeführten Züge höchstens eine Maus mehr in eine Greedyzelle hineingezogen werden darf als herausgezogen wird – die Kapazitätsschranke für die Löcher der Greedyzellen.

Für Überlappungsvariablen, die in den Kreisüberdeckungen auftreten (also den $x_{i,\tau_k(i)}$ für $k \in \{1,2\}$), ergeben sich:

$$\sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{nach } (i,\tau_k(i))}} z_j \; - \sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{aus } (i,\tau_k(i)) \text{ heraus}}} z_j \leq -1$$

Diese Forderung verlangt, dass nach Ausführung aller gewählten Züge eine Maus weniger auf den Startfeldern steht als zu Beginn, dass also alle Mäuse von den Startpositionen wegbewegt werden[2].

Für alle übrigen Variablen $x_{i,k}$ ergibt sich

$$\sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{nach } (i,k)}} z_j \; - \sum_{\substack{j \\ \text{Zug } j \text{ zieht eine Maus} \\ \text{aus } (i,k) \text{ heraus}}} z_j \leq 0,$$

Mäuse, die auf ein Feld gezogen werden, müssen es also auch wieder verlassen.

Das duale Programm ist also eine fraktionale Beschreibung unseres Mäusespiels, wobei wir bei der Interpretation als Brettspiel zwei implizite Einschränkungen vornehmen, die glücklicherweise bei der vorzustellenden Rangabstiegsmethode nicht zum Problem werden, derer man sich aber dennoch bewußt sein sollte.

1. Es ist prinzipiell möglich, dass eine Menge von Gleichungen existiert, die eine ausreichende Schranke liefern, dass aber gleichzeitig die entsprechenden Züge nicht in eine hintereinander ausführbare Reihenfolge gebracht werden können, weil sich die Züge zyklisch gegenseitig bedingen. In dem Fall wäre es nötig vorübergehend mit *negativen Mäusen* zu arbeiten, was das Spiel unübersichtlich machen würde.

2. Sollten fraktionale Lösungen für die dualen Variablen auftreten, so müsste man die Mäuse in Teilmäuse aufteilen, was ebenfalls das Spiel verkomplizieren würde.

Im nächsten Abschnitt beschreiben wir nun ein allgemeines Verfahren, wie das Mäusespiel auf einer Teilmenge der Greedyordnungen für allgemeine n gewonnen werden kann. Auf den Aspekt der linearen Programmierung, kommen wir im Abschnitt 2.6 noch einmal zurück.

[2]Sollte eine Zelle in beiden τ_k auftreten, so steht auf der rechten Seite eine -2. Ist eine Zelle $(i,\tau_k(i))$ von der Form $(j,j+1)$, ist sie also auch im Greedy-Superstring relevant, so ergibt sich auf der rechten Seite der Ungleichung eine 0. Ist die Zelle Teil beider τ_k und des Greedy-Superstrings, so erhalten wir rechts eine -1.

2.4 Die Rangabstiegsmethode

Die Frage, ob für eine gegebene Instanz (Greedyordnung, C_1 und C_2) eine betrachtete Menge von Ungleichungen genügt, Gleichung (2.4) zu beweisen, kann mittels linearer Programmierung getestet werden, wie wir gerade gesehen haben. Für ein gegebenes n können prinzipiell alle $(n-1)!n!^2$ Instanzen in endlicher Zeit ausprobiert werden. Beispielsweise haben wir enumerativ verifiziert, dass die klassische Greedy-Conjecture $|S_G| \leq 2 \cdot |S_{opt}|$ für Mengen mit bis zu $n = 7$ Strings gilt. Ein Gegenbeispiel müsste also aus mindestens acht Strings bestehen.

Um allerdings für allgemeine n und ganze Klassen von Instanzen Aussagen zu erzielen, sind allgemeine Argumente über die auftretenden Spielkonfigurationen nötig.

Die Rangabstiegsmethode wurde in (WS03) eingeführt und in (LW04; LW05) weiterentwickelt. Wir stellen sie hier in einer allgemeineren Form vor, die einen kompakteren Beweis erlaubt. In der hier gewählten Darstellung kann die Rangabstiegsmethode bei jeder Greedyordnung und für jedes Paar expandierender Kreisüberdeckungen ausgeführt werden. Der Algorithmus endet stets entweder mit einem gewonnenen Spiel oder mit einer expliziten Aufgabe.

In Abschnitt 2.4.1 führen wir einige erforderliche Begriffe ein, stellen die Invarianten vor, die den Korrektheitsbeweis liefern, und motivieren diese Invarianten auf intuitiver Ebene.

Abschnitt 2.4.2 enthält den Hauptteil der Rangabstiegsmethode und den Korrektheitsbeweis. Es wird gezeigt, dass die Rangabstiegsmethode die gewählten Invarianten bewahrt, die von ihr ausgewählten Züge stets existieren und, dass diese legal sind.

Im Abschnitt 2.4.3 wird gezeigt, dass der Algorithmus bei bilinearen Greedyordnungen niemals aufgibt. Damit ist dann der Beweis von Theorem 2.1 abgeschlossen. Abschnitt 2.4.4 beschreibt schließlich wie aus Theorem 2.1 das griffigere Korollar 2.1 folgt.

2.4.1 Vorbereitung

Wir führen zunächst die nötigen Begriffe ein und sammeln relevante Eigenschaften. Wesentlich für den Algorithmus ist das Konzept des Teilbretts und des Rahmens.

Definition 2.7 *Sei ein $n \times n$ Spielbrett gegeben und sei $I \subseteq \{1, \ldots, n\}$ eine Indexmenge. Dann setzen wir*

$$Brett(I) := \{(x, y) | x \in I, y \in I\}$$

und nennen $Brett(I)$ das Teilbrett von I. Sei weiter $B = Brett(I)$ ein Teil-

brett, dann bezeichnen wir

$$\begin{aligned} horiz(B) &:= \{(x,y)|x \in I, y \notin I\} \\ vert(B) &:= \{(x,y)|x \notin I, y \in I\} \\ Rahmen(B) &:= horiz(B) \cup vert(B) \end{aligned}$$

und nennen $horiz(B)$ bzw. $vert(B)$ den horizontalen bzw. vertikalen Rahmen von B.

Das folgende Lemma hält zwei wichtige Eigenschaften von Teilbrettern und Rahmen fest. Wir sagen, dass der Rahmen eines Teilbretts in einer bestimmten Spielkonfiguration *verwaist* ist, wenn sich keine Mäuse in Zellen des Rahmens befinden.

Lemma 2.10 *1. Befinden sich in jeder Zeile und jeder Spalte des Spielbretts gleich viele Mäuse, so gilt für jedes Teilbrett $B = Brett(I)$, dass horizontaler und vertikaler Rahmen von B die gleiche Zahl von Mäusen enthalten. Insbesondere sind vertikaler und horizontaler Rahmen stets beide verwaist oder beide nicht verwaist.*

2. Sei $\overline{I} = \{1, \ldots, n\} \setminus I$, so gilt

$$\begin{aligned} horiz(Brett(\overline{I})) &= vert(Brett(I)) \\ vert(Brett(\overline{I})) &= horiz(Brett(I)) \\ Rahmen(Brett(\overline{I})) &= Rahmen(Brett(I)) \end{aligned}$$

Beweis:

1. Sei k die Zahl der Mäuse in jeder Zeile und Spalte. Dann befinden sich in Zeilen i mit $i \in I$ insgesamt genau $|I| \cdot k$ Mäuse. Dieselbe Anzahl befindet sich in Spalten i mit $i \in I$. Ist z die Zahl der Mäuse im $Brett(I)$, so befinden sich im horizontalen und vertikalen Rahmen jeweils $|I| \cdot k - z$ Mäuse.

2. Nach Definition 2.7 gilt

$$\begin{aligned} horiz(Brett(\overline{I})) &= \{(x,y)|x \in \overline{I}, y \notin \overline{I}\} \\ &= \{(x,y)|x \notin I, y \in I\} \\ &= vert(Brett(I)) \end{aligned}$$

   Die weiteren Gleichungen folgen jetzt unmittelbar.

□

Definition 2.8 *Wir nennen einen Zug* rechteckig, *wenn er zwei Mäuse von Zellen (x_1, y_1) und (x_2, y_2) mit $x_1 \neq x_2$ und $y_1 \neq y_2$ in die Zellen (x_1, y_2) und (x_2, y_1) verschiebt.*

Zum Beispiel sind Greedy- und Diagonal-Monges rechteckig.

Beobachtung 1 *Auch einen Tripelzug*

$$(a,b),(x,d),(c,x),(x,x) \rightarrow (x,b),(c,d),(a,x),|x|$$

kann man sich als Hintereinanderausführung zweier rechteckiger Züge denken, von denen der erste für sich genommen allerdings keinen legalen Zug darstellt.

$$\begin{array}{rcl}(a,b),(x,x) & \rightarrow & (a,x),(x,b)\\ (c,x),(x,d) & \rightarrow & |x|,(c,d)\end{array}$$

Wenn wir also im Weiteren bemerken, dass eine Eigenschaft durch Ausführung rechteckiger Züge nicht zerstört wird, so werden auch Tripelzüge die Eigenschaft bewahren.

Die folgende Beobachtung ist wesentlich, da die Vermeidung verwaister Rahmen ein Schlüssel zur Korrektheit der Rangabstiegsmethode ist.

Lemma 2.11 *Bei einem rechteckigen Zug*

$$(x_1,y_1),(x_2,y_2) \rightarrow (x_1,y_2),(x_2,y_1)$$

wird der Rahmen eines Teilbretts $B = Brett(I)$ nur dann geleert, wenn eine Ausgangsposition im horizontalen Rahmen liegt, eine im vertikalen Rahmen liegt und genau eine Zielposition in B selbst liegt.

Beweis: Da der Rahmen von B annahmegemäß vor dem Zug nicht leer ist, muss er eine Maus enthalten haben und nach Lemma 2.10.1 müssen dann sowohl horizontaler wie auch vertikaler Rahmen eine Maus enthalten haben, und dabei muss es sich um die am Zug beteiligten Mäuse handeln. Gehört o.B.d.A. die Position (x_1,y_1) zum horizontalen Rahmen und (x_2,y_2) zum vertikalen Rahmen, so gilt $x_1 \in I$, $y_1 \notin I$, $x_2 \notin I$, $y_2 \in I$. Damit liegt (x_1,y_2) in B und (x_2,y_1) außerhalb von B und außerhalb des Rahmens von B. □

Zur Formulierung der Rangabstiegsmethode und der Invarianten weisen wir jeder Diagonalzelle (i,i) mit $1 \leq i \leq n$ einen Level zu.

Definition 2.9 *Für Diagonalzellen definieren wir Level wie folgt:*

$$\begin{array}{rcl} level(1,1) & := & rang(1,2)\\ level(i,i) & := & \max\{rang(i-1,i), rang(i,i+1)\} \text{ für } 2 \leq i \leq n-1\\ level(n,n) & := & rang(n-1,n)\end{array}$$

Die Rangabstiegsmethode

1. **Eingabe**: zwei expandierende Kreisüberdeckungen C_1, C_2 und eine Greedyordnung.
2. **Initialisierung**
   (a) Lege das Spielbrett entsprechend der Greedyordnung an.
   (b) Für i von 1 bis n und k von 1 bis 2
      i. Sei s_j Nachfolger von s_i in C_k.
      ii. Platziere eine Maus in der Zelle (i, j). Falls $i = j$ gilt, so wird die Maus auf dem Rechteck, das die Eigenüberlappung (s_i, s_i) repräsentiert, und nicht im Diagonalloch abgelegt.
   (c) Setze $r = n$;
3. Hauptschleife

Algorithmus 2.3: Die Initialisierung der Rangabstiegsmethode

Der Level einer Diagonalzelle entspricht also stets dem Rang der höherrangigen benachbarten Greedyzelle (oder ggf. dem Rang der einzigen benachbarten Greedyzelle). Wir können nun den Initialisierungsteil der Rangabstiegsmethode formulieren: Algorithmus 2.3.

Bevor wir mit der Hauptschleife die anzuwendenden Züge bestimmen, sollen zunächst die Invarianten aufgelistet werden, die den Korrektheitsbeweis tragen. Wir verifizieren, dass sie nach der Initialisierung gelten und begründen auf intuitiver Ebene, warum es sich dabei um bewahrenswerte Eigenschaften der Spielkonfiguration handelt.

I1 *Jede Zeile und jede Spalte des Spielbretts enthält genau zwei Mäuse.*

Diese Invariante ist erfüllt, da in jeder der beiden Kreisüberdeckungen jeder String genau einmal als Vorgänger und einmal als Nachfolger eines Strings auftritt.

Da es in jeder Zeile und Spalte auch genau zwei Zielpositionen gibt, bietet sich diese Invariante fast von selbst an. Außerdem erlaubt diese Invariante eine Argumentation mit Hilfe von Lemma 2.10.

Da alle Züge, die im Rahmen der Rangabstiegsmethode zum Einsatz kommen (Monges, Periodenverzicht und Tripelzug), die Zahlen der Mäuse pro Zeile und Spalte unverändert lassen, brauchen wir auf die Wahrung von $I1$ im weiteren Verlauf nicht gesondert zu achten. Beim

Nachweis der Existenz und der Verträglichkeit von Zügen wird sie sich jedoch als sehr nützlich erweisen.

I2 *Jede Greedyzelle G mit $rang(G) \geq r$ enthält eine Maus.*

Die Initialisierung setzt r auf n und damit ist die Invariante trivialerweise erfüllt, da überhaupt nur Ränge bis einschließlich $n-1$ vergeben werden. Der Algorithmus wird r schrittweise senken, daher der Name Rangabstiegsmethode, bis für $r = 1$ alle Greedyzellen eine Maus enthalten.

Diese Invariante beschreibt somit unseren Fortschritt.

I3 *Genau die Diagonallöcher in Zellen D mit $level(D) \geq r$ enthalten eine Maus.*

Anfänglich haben alle Diagonalen einen Level echt kleiner als r und in der Tat enthält kein Diagonalloch eine Maus. Zwar kann eine Diagonalzelle eine Maus beherbergen, was genau dann geschieht, wenn eine der Kreisüberdeckungen einen Einzelkreis enthält, wenn also ein String sein eigener Nachfolger ist. Dann allerdings wird die Maus in der Diagonalzelle nicht im Diagonalloch sondern auf dem Rechteck für die Eigenüberlappung des Strings abgelegt.

Diese Invariante ist in ihrer Bedeutung mit $I2$ verwandt. Während jedoch vorzeitig mit Mäusen besetzte Greedyzellen in $I2$ kein Problem darstellen und deswegen auch nicht verboten sind, darf hier ein Diagonalloch nicht vorzeitig gefüllt werden. Die Idee der Rangabstiegsmethode ist es, die Greedyzellen, die ja mit sinkendem Rang immer weniger eigene Anziehungskraft haben, unter Zuhilfenahme der inzidenten Diagonalzellen zu füllen. Daher ist mit der Kapazität der Diagonallöcher sparsam umzugehen.

I4 *Für alle Teilbretter $B = Brett(I)$ mit $\emptyset \subset I \subset \{1, \ldots, n\}$ ist $Rahmen(B)$ nicht verwaist.*

Besitzt ein Teilbrett $B = Brett(I)$ einen verwaisten Rahmen, so befinden sich alle $2(|I|)$ Mäuse der Zeilen $i \in I$ innerhalb von B. Es wird also in keiner der beiden Kreisüberdeckungen einem String aus $\{s_i | i \in I\}$ ein Nachfolger außerhalb dieser Menge zugeordnet. Beide Kreisüberdeckungen enthalten also eine Kreisüberdeckung für diese echte Teilmenge der Strings – im Widerspruch zur Expansionseigenschaft.

Halten wir weiter fest, dass es in der Gewinnstellung des Spiels keine Bretter mit verwaisten Rahmen gibt: Nach Lemma 2.10.2 gilt

$$Rahmen(I) = Rahmen(\overline{I}),$$

und deshalb genügt es zu zeigen, dass es kein $Brett(I)$ mit $1 \in I \subset \{1, \ldots, n\}$ gibt, dessen Rahmen in der Gewinnstellung verwaist ist. Sei $Brett(I)$ ein solches Brett, dann existiert ein minimales k mit $k \notin I$, da I eine echte Teilmenge von $\{1, \ldots, n\}$ ist. Desweiteren befindet sich die Maus in der Greedyzelle $(k-1, k)$ im horizontalen Rahmen von I und der horizontale Rahmen ist nicht leer. Letztlich darf es also keine Teilbretter mit verwaistem Rahmen geben.

Schließlich soll an dieser Stelle der Hinweis genügen, dass zwischenzeitlich verwaiste Rahmen nur schwer oder gar nicht wieder mit Mäusen belebt werden können.

Wir halten fest:

Lemma 2.12 *Nach der Initialisierungsphase der Rangabstiegsmethode gelten alle vier Invarianten.*

2.4.2 Die Hauptschleife

In diesem Abschnitt stellen wir die Hauptschleife der Rangabstiegsmethode vor (siehe Algorithmus 2.4).

Bevor wir in die Details einsteigen, soll die grundsätzliche Idee des Algorithmus kurz umrissen werden. In einem Schleifendurchlauf soll die Greedyzelle mit Rang r sowie Diagonallöcher mit Level r gefüllt werden. Da der Level einer Diagonalzelle sich nach der höherrangigen inzidenten Greedyzelle richtet, heißt das, dass stets die Greedyzelle vom Rang r und benachbarte Diagonallöcher, die noch nicht in früheren Iterationen gefüllt wurden, zum Füllen anstehen.

Ist die Greedyzelle $G = (g, g+1)$ ranghöher als $(g-1, g)$ und ranghöher als $(g+1, g+2)$, der Gipfel-Fall, dann haben beide angrenzenden Diagonalzellen (g, g) und $(g+1, g+1)$ den Level r und werden zusammen mit G gefüllt. (Dies gilt auch, wenn eine der Greedyzellen $(g-1, g)$ oder $(g+1, g+2)$ nicht existiert, da der Spielfeldrand erreicht ist, und G ranghöher als die existierende Greedyzelle ist.)

Ist die Greedyzelle $G = (g, g+1)$ genau einer benachbarten Greedyzelle rangmäßig überlegen und einer unterlegen, der Hang-Fall, so hat genau eine der beiden angrenzenden Diagonalzellen den Level r. Die andere hat einen höheren Rang und ist annahmegemäß schon in einer früheren Iteration mit einer Maus gefüllt worden. Dasselbe gilt, falls G nur eine benachbarte Greedyzelle besitzt und dieser rangmäßig unterlegen ist.

Im dritten Fall, dem Senken-Fall, ist $G = (g, g+1)$ von zwei ranghöheren Greedyzellen umgeben. In diesem Fall sind beide inzidenten Diagonallöcher bereits gefüllt, und es kann passieren, dass der Algorithmus aufgeben muss.

Im Abschnitt 2.4.3 werden wir verifizieren, dass bei bilinearen Greedyordnungen der Senken-Fall nicht auftritt bzw. nicht kritisch ist. Im weiteren

<table>
<tr><td colspan="8">Initialisierung (siehe Algorithmus 2.3)</td></tr>
<tr><td colspan="8">Für r von $n-1$ bis 1 wiederhole</td></tr>
<tr><td colspan="8">Setze $G = (g, g+1)$, so dass $rang(G) = r$ gilt</td></tr>
<tr><td colspan="8">Wieviele Diagonalzellen aus $\{(g,g),(g+1,g+1)\}$ haben Level r ?</td></tr>
<tr><td colspan="2">2 (Gipfel)</td><td colspan="3">1 (Hang)</td><td colspan="3">0 (Senke)</td></tr>
<tr><td colspan="2">Enthält Zelle (g,g) eine Maus?</td><td colspan="3" rowspan="3">Setze $D \in \{(g,g),(g+1,g+1)\}$
so dass $level(D) = r$</td><td>Enthält G eine Maus?</td><td colspan="2">nein</td></tr>
<tr><td>ja</td><td>nein</td><td>ja</td><td colspan="2" rowspan="3">Ist Greedy-Monge
in G verträglich?</td></tr>
<tr><td>Periodenverzicht
in (g,g) (1)</td><td>Diagonal-Monge
in (g,g) (2)</td><td rowspan="2"></td></tr>
<tr><td colspan="2">Setze $D = (g+1, g+1)$</td></tr>
<tr><td colspan="3">Enthält G eine Maus ?</td><td colspan="3">ja</td><td>nein</td></tr>
<tr><td colspan="3">ja</td><td colspan="2">nein</td><td>ja</td><td>nein</td></tr>
<tr><td colspan="2">Enthält D eine Maus?</td><td colspan="2">Enthält D eine Maus?</td><td rowspan="2"></td><td>Greedy-Monge</td><td>STOPP</td></tr>
<tr><td>ja</td><td>nein</td><td>nein</td><td>ja</td><td>in G (9)</td><td>(10)</td></tr>
<tr><td rowspan="3">Periodenverzicht
in D (3)</td><td rowspan="2"></td><td>Greedy-Monge in G (4)</td><td colspan="2">Ist Tripelzug in G und D verträglich?</td><td colspan="2" rowspan="3"></td></tr>
<tr><td colspan="2" rowspan="2">Diagonal-Monge in D
der die Maus in G
nicht bewegt (5)</td><td>nein</td><td>ja</td></tr>
<tr><td>Greedy-Monge in G (7)</td><td rowspan="2">Tripelzug (6)</td></tr>
<tr><td colspan="2">Periodenverzicht in D (8)</td></tr>
</table>

Algorithmus 2.4: Die Hauptschleife der Rangabstiegsmethode. Ein Zug heißt verträglich, wenn seine Ausführung $I4$ nicht verletzt. Es dürfen nur verträgliche Züge ausgeführt werden. Dass stets mindestens ein verträglicher Zug der vorgesehenen Familie existiert, ist noch zu zeigen.

Verlauf dieses Abschnitts werden wir sicherstellen, dass die vom Algorithmus geforderten Züge stets ausführbar sind und nach jedem Schleifendurchlauf unsere Invarianten $I1$ bis $I4$ gelten. Wir nennen einen Zug genau dann verträglich, wenn er keine der Invarianten zerstört.

Zunächst halten wir fest, dass weder Periodenverzichte, noch Diagonal- oder Greedy-Monges oder Tripelzüge die Anzahl der Mäuse pro Zeile und Spalte verändern. $I1$ ist also nicht in Gefahr.

Invariante $I2$ folgt, sobald die Existenz und Legalität der verlangten Züge etabliert ist: In jedem der Fälle wird, falls G nicht bereits eine Maus beherbergt, ein Zug vorgesehen, der eine Maus nach G befördert.

Für $I3$ gilt Ähnliches, was die Füllung der Diagonallöcher mit Level $\geq r$ angeht. In jedem der Fälle wird ein Zug vorgesehen, der eine Maus in das Diagonalloch bringt. Ist die Existenz dieses Zuges gesichert, so folgt auch $I3$. Für das Verbot von vorzeitig gefüllten Diagonallöchern (solchen mit $Level < r$) ist festzuhalten, dass falls eine Diagonalzelle mit $Level < r$ im Laufe der ausgewählten Züge eine Maus erhält, diese Maus im Rechteck für die Eigenüberlappung landet: Diagonallöcher werden nur bei Periodenverzichten, Diagonal-Monges und Tripelzügen gefüllt. Die Diagonalzellen, *in denen* die Züge ausgeführt werden, sind im Algorithmus aber stets vom Level r.

Wir können uns also bei den folgenden Betrachtungen der einzelnen vom Algorithmus vorgesehenen Züge auf Existenz, Legalität und die Verträglichkeit mit $I4$ beschränken. Wir haben uns in Lemma 2.12 davon überzeugt, dass die Invarianten vor dem ersten Schleifendurchlauf gelten und argumentieren nun induktiv.

Handeln wir zunächst die drei Stellen im Algorithmus ab, in denen ein **Periodenverzicht** vorgesehen ist: **(1)**,**(3)** und **(8)** in Algorithmus 2.4. Zu diesen Zügen kommt es jeweils nur dann, wenn die betreffende Diagonalzelle eine Maus enthält, und man überzeuge sich, dass der Level der betrachteten Diagonalen r ist. Da $I3$ vor dem aktuellen Schleifendurchlauf galt, befand sich diese Maus im Rechteck für die Eigenüberlappung und nicht im Loch der Diagonalzelle. Damit existiert der Periodenverzicht und Legalität ist trivial, da Periodenverzichte unbedingten Ungleichungen entsprechen. Die Verträglichkeit mit $I4$ ist gegeben, da nur eine Maus am Zug beteiligt ist (Lemma 2.11), die zudem noch in ihrer Zelle verbleibt.

Den Betrachtungen für die weiteren Züge stellen wir die folgenden drei wesentlichen Lemmata voran, auf die wir uns in jedem der Einzelfälle beziehen werden.

Lemma 2.13 *In einer Spielkonfiguration mögen die Invarianten $I1$ und $I4$ gelten. Weiter befinde sich in den Zellen (x_1, y_1), (x_2, y_2) und (x_2, y_3) mit $x_1 \neq x_2$ und $y_1 \neq y_2 \neq y_3 \neq y_1$ jeweils mindestens eine Maus. Dann verletzt*

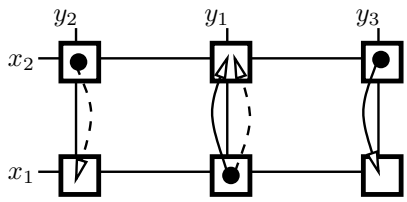


Abbildung 2.10: Illustration für Lemma 2.13

mindestens einer der rechteckigen Züge:

$$\begin{aligned}(x_1, y_1), (x_2, y_2) &\rightarrow (x_1, y_2), (x_2, y_1)\\ (x_1, y_1), (x_2, y_3) &\rightarrow (x_1, y_3), (x_2, y_1)\end{aligned}$$

Invariante I4 nicht.

Abbildung 2.10 stellt die im Lemma betrachtete Situation graphisch dar.

Beweis: Annahmegemäß gilt $I4$ in der Ausgangskonfiguration. Der Rahmen eines jeden Teilbretts ist also nicht verwaist. Sei im Widerspruch angenommen, dass beide Züge zur Verwaisung irgendeines Rahmens führen.

Der Zug $(x_1, y_1), (x_2, y_2) \rightarrow (x_1, y_2), (x_2, y_1)$ führe zur Verwaisung des Rahmens eines Teilbretts $Brett(I)$ mit $\emptyset \subset I \subset \{1, \ldots, n\}$. Nach Lemma 2.11 muss eine Maus im horizontalen und eine im vertikalen Rahmen von $Brett(I)$ liegen. Wegen Lemma 2.10.2 können wir $x_1 \in I$ annehmen, denn sonst wäre $x_1 \in \overline{I}$ und wir argumentieren mit $Brett(\overline{I})$ weiter, dessen Rahmen nach Lemma 2.10.2 ja auch geleert wird.

Damit ergibt sich das Gesamtbild $(x_1, y_1) \in horiz(Brett(I))$, $(x_2, y_2) \in vert(Brett(I))$, also $x_1, y_2 \in I$ und $x_2, y_1 \notin I$. Außerdem muss $y_3 \notin I$ gelten, da die Maus auf (x_2, y_3) anderenfalls im vertikalen Rahmen von $Brett(I)$ liegt und dort verbleibt.

Diese Überlegung kann für den Zug $(x_1, y_1), (x_2, y_3) \rightarrow (x_1, y_3), (x_2, y_1)$ und die Verwaisung des Rahmens eines Teilbretts $Brett(J)$ ebenso angestellt werden. Wieder nehmen wir $x_1 \in J$ an. Dann ist: $(x_1, y_1) \in horiz(Brett(J))$ und $(x_2, y_3) \in vert(Brett(J))$, also $x_1, y_3 \in J$ und $x_2, y_1 \notin J$. Außerdem muss $y_2 \notin J$ gelten, da die Maus auf (x_2, y_2) anderenfalls im vertikalen Rahmen von $Brett(J)$ liegt und dort verbleibt.

Zusammenfassend haben wir also folgende Situation:

$$\begin{aligned}x_1 &\in I \cap J\\ y_2 &\in I \setminus J\\ y_3 &\in J \setminus I\\ \{x_2, y_1\} &\subseteq \overline{I} \cap \overline{J}\end{aligned}$$

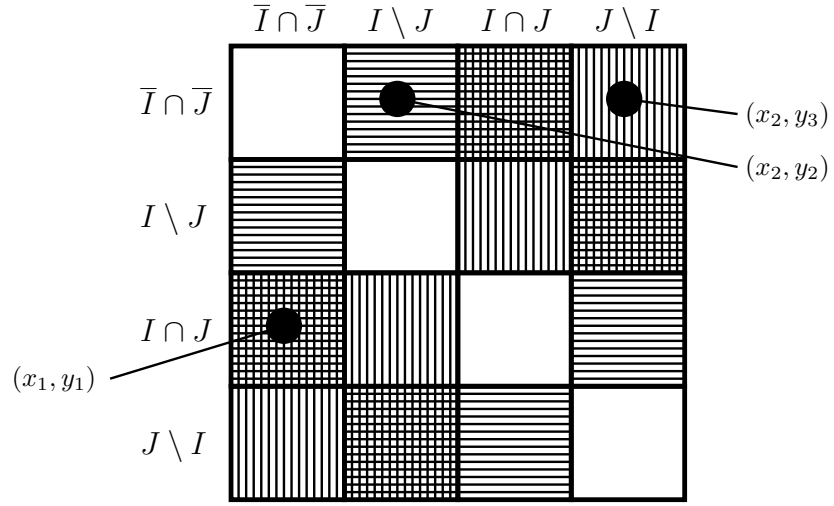


Abbildung 2.11: Die schematische Aufteilung des Spielbretts in 16 nichtleere Segmente in Abhängigkeit von den Indexmengen I und J.

Damit können wir, wie in Abbildung 2.11 dargestellt, das Spielbrett gedanklich in 16 nichtleere Segmente aufteilen. Die quer gestreiften Segmente bilden den Rahmen von $Brett(I)$, die längs gestreiften Segmente bilden den Rahmen von $Brett(J)$.

Nach Fallannahme befinden sich in den Rahmen der Bretter $Brett(I)$ und $Brett(J)$ außer den drei Mäusen auf $(x_1, y_1), (x_2, y_2)$ und (x_2, y_3) keine weiteren Mäuse. Damit enthält der horizontale Rahmen von $Brett(I \cap J)$ eine Maus und der vertikale Rahmen keine. Mit Lemma 2.10.1 widerspricht dies der Annahme, dass $I1$ und $I4$ vor dem Zug gegolten haben. □

Vertauscht man die Rollen von Zeilen und Spalten in diesen Betrachtungen, so erhält man analog das folgende Lemma.

Lemma 2.14 *In einer Spielkonfiguration mögen die Invarianten $I1$ und $I4$ gelten. Weiter befinde sich in den Zellen (x_1, y_1), (x_2, y_2) und (x_3, y_2) mit $x_1 \neq x_2 \neq x_3 \neq x_1$ und $y_1 \neq y_2$ jeweils mindestens eine Maus. Dann verletzt mindestens einer der rechteckigen Züge:*

$$\begin{aligned} (x_1, y_1), (x_2, y_2) &\rightarrow (x_1, y_2), (x_2, y_1) \\ (x_1, y_1), (x_3, y_2) &\rightarrow (x_1, y_2), (x_3, y_1) \end{aligned}$$

Invariante $I4$ nicht.

Schließlich stellen wir noch fest, dass Züge aus mehrfach belegten Zellen nie zum Problem werden können:

Lemma 2.15 *Befinden sich bei einem rechteckigen Zug*

$$(x_1, y_1), (x_2, y_2) \rightarrow (x_1, y_2), (x_2, y_1)$$

auf einer der Ausgangspositionen mehr als eine Maus, so verstößt der Zug nicht gegen $I4$.

Beweis: Angenommen $Rahmen(Brett(I))$ wird durch den Zug geleert. Nach Lemma 2.11 muss die erste Ausgangsposition dem horizontalen Rahmen und die zweite dem vertikalen Rahmen von $Brett(I)$ angehören. Da eine Ausgangsposition doppelt belegt ist, wird eine der Zellen weiterhin eine Maus beherbergen. □

Diese Überlegungen gilt es nun für die Züge im Einzelnen zur Anwendung zu bringen. Die Laufnummern der Schritte entsprechen den Markierungen in Algorithmus 2.4.

Der **Diagonal-Monge** in Schritt **(2)**: Wegen $I1$ enthält sowohl die Zeile g wie auch die Spalte g zwei Mäuse. Keine davon befindet sich nach Fallannahme in der Diagonalzelle (g, g). Damit handelt es sich also um vier verschiedene Mäuse. Die Legalität ist bei Diagonal-Monges prinzipiell gegeben, da Diagonal-Monges unbedingten linearen Ungleichungen entsprechen.

Befinden sich beide Mäuse der Zeile g oder beide Mäuse der Spalte g in derselben Zelle, so liefert Lemma 2.15 sofort die Verträglichkeit der Diagonal-Monges mit $I4$ und wir sind fertig.

Im anderen Fall wählen wir willkürlich eine Mausposition (x_1, g). Die beiden verschiedenen Positionen der Mäuse der Zeile g mögen sich in den Zellen (g, y_2) und (g, y_3) befinden. Setzen wir nun $x_2 = y_1 = g$ in Lemma 2.13, so erhalten wir, dass einer der beiden Züge

$$\begin{aligned} (x_1, g), (g, y_2) &\rightarrow (x_1, y_2), (g, g) \\ (x_1, g), (g, y_3) &\rightarrow (x_1, y_3), (g, g) \end{aligned}$$

verträglich mit $I4$ ist.

Bevor wir uns den Greedy-Monges und dem Tripelzug zuwenden können, formulieren wir das folgende Lemma, das uns garantiert, dass die Greedyzelle, die zur Füllung ansteht, die noch *frei herumlaufenden Mäuse* rangmäßig dominiert, dass wir also alle Mäuse, die wir bewegen wollen, auch bewegen können.

Lemma 2.16 *Es mögen die Invarianten $I1$ bis $I4$ zu Beginn des Schleifendurchlaufs für ein konkretes r gelten. Dann gibt es keine Mäuse in Zellen mit Rang größer r mit Ausnahme der $n - 1 - r$ Mäuse in den Greedyzellen mit Rängen $r + 1, \ldots, n - 1$.*

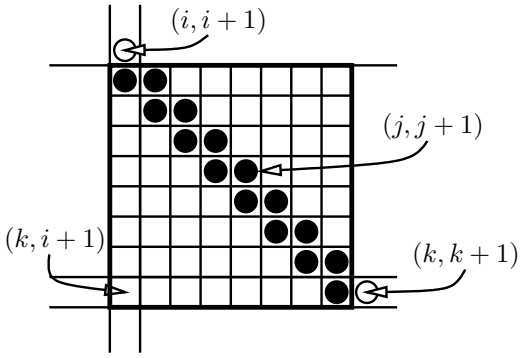


Abbildung 2.12: Illustration für Lemma 2.16

Beweis: Nehmen wir im Widerspruch an, es gibt eine Maus in einer Zelle mit Rang $z > r$ und diese Maus befindet sich nicht in der Greedyzelle vom Rang z. Sei $(j, j+1)$ diese Greedyzelle vom Rang z. Nach $I2$ ist $(j, j+1)$ bereits mit einer Maus gefüllt. Außerdem, nach $I3$, sind die Diagonallöcher (j, j) und $(j+1, j+1)$ links neben und unter dieser Greedyzelle bereits gefüllt, da ihr Level mindestens z ist. Wegen $I1$ befinden sich in der Zeile j und der Spalte $j+1$ jeweils genau zwei Mäuse. Diese befinden sich also in den Löchern $(j, j), (j, j+1)$ und $(j+1, j+1)$, und in Zeile j und Spalte $j+1$ gibt es somit keine weiteren Mäuse mehr. Nach Lemma 2.4 gibt es nur eine Zelle mit Rang z, die nicht in Zeile j oder Spalte $j+1$ liegt. Diese befindet sich in der Zelle $(k, i+1)$, wobei i und k wie in Lemma 2.4 beschrieben gewählt sind. Abbildung 2.12 verdeutlicht die Situation.

Nach Lemma 2.4 sind alle Greedyzellen von $(i+1, i+2)$ bis $(k-1, k)$ außer $(j, j+1)$ ranghöher als $(j, j+1)$. Wegen $I2$ und $I3$ sind die Löcher in den Greedyzellen nebst inzidenten Diagonallöchern sämtlich bereits gefüllt. Befindet sich nun wie angenommen in der Zelle $(k, i+1)$ eine Maus, so liegen alle Mäuse der Zeilen $i+1, \ldots, k$ in den Spalten $i+1, \ldots, k$. Damit wäre aber der Rahmen von $Brett(\{i+1, \ldots, k\})$ verwaist – im Widerspruch zu $I4$. □

Als nächstes betrachten wir den **Greedy-Monge** in Schritt **(4)**: Die Situation ist hier die, dass ein zu G benachbartes Diagonalloch bereits eine Maus enthält: Im Hang-Fall hat genau eine Diagonalzelle aus $\{(g, g), (g+1, g+1)\}$ den Level r, und die andere Diagonalzelle muss einen höheren Rang besitzen und ist damit bereits gefüllt. Im Gipfel-Fall besitzen beide Diagonalzellen den Rang r und die Diagonalzelle (g, g) ist in den Schritten (1) oder (2) mit einer Maus gefüllt worden. In jedem Fall befindet sich in der anderen benachbarten Diagonalzelle D keine Maus (weder im Loch, noch auf dem Rechteck für die Eigenüberlappung).

Sei o.B.d.A. die Diagonalzelle (g, g) bereits gefüllt und die Diagonalzelle

$D = (g{+}1, g{+}1)$ leer. Dann befindet sich in Zeile g noch eine Maus außerhalb der Diagonalen (sei sie in der Zelle (g, y_1)) und in Spalte $g + 1$ sind beide Mäuse außerhalb der Diagonalen in den Zellen $(x_2, g + 1)$ und $(x_3, g + 1)$. Wegen Lemma 2.16 befinden sich diese drei Mäuse alle in Zellen mit Rang höchstens r. Die beiden Greedy-Monges

$$\begin{aligned}(g, y_1), (x_2, g + 1) &\rightarrow (g, g + 1), (x_2, y_1)\\ (g, y_1), (x_3, g + 1) &\rightarrow (g, g + 1), (x_3, y_1)\end{aligned}$$

sind also beide mit der Kraft der Greedyzelle G legal. Gilt $x_2 = x_3$, so liefert Lemma 2.15 die Verträglichkeit mit $I4$. Im anderen Fall sichert Lemma 2.13 die Existenz eines verträglichen Zuges unter den beiden.

Der **Diagonal-Monge** in Schritt **(5)** ist wieder unkritisch, was die Legalität angeht, da Diagonal-Monges unbedingten Ungleichungen entsprechen. Es befinden sich zwei Mäuse in der Zeile von D und zwei Mäuse in der Spalte von D, und da D selbst annahmegemäß keine Maus enthält, handelt es sich dabei um vier verschiedene Mäuse. Eine davon liegt auf G und soll dort bleiben. Liegen zwei der vier Mäuse in derselben Zelle, so liefert wieder Lemma 2.15 die Verträglichkeit. Andernfalls gibt es zwei verschiedene Greedy-Monges und, je nachdem ob sich D unter oder links neben G befindet, liefert entweder Lemma 2.13 oder Lemma 2.14 die Verträglichkeit von mindestens einem Zug mit $I4$.

Für den **Tripelzug** an Position **(6)** können wir für die Legalität wieder auf Lemma 2.16 verweisen. Nach Annahme enthält D eine Maus im Rechteck für die Eigenüberlappung, und G enthält keine Maus. Das Loch der anderen benachbarten Diagonalzelle von G ist bereits gefüllt: Wenden wir wieder die Argumentation in der Analyse von Schritt (4) an. Es bleibt also sowohl in der Zeile wie auch in der Spalte von G eine Maus übrig, die von der Greedyzelle G gemäß Lemma 2.16 auch dominiert wird. Dasselbe gilt für die dritte am Tripel beteiligte Maus, die sich nicht auch auf D befinden kann, da dann der Rahmen des Teilbretts, das nur D enthält, verwaist wäre.

Verträglichkeit des Zuges mit $I4$ ist im Algorithmus zur Bedingung gemacht, braucht also hier nicht überprüft werden.

Der **Greedy-Monge** in Schritt **(7)** gelangt zur Ausführung, wenn der Tripel in G und D gegen $I4$ verstößt. Abbildung 2.13 stellt die Situation, in der es entweder zum Tripel (6) oder zum Greedy-Monge (7) und Periodenverzicht (8) kommt, für den Fall $D = (g + 1, g + 1)$ dar. Durch Lemma 2.16 wissen wir, dass G alle relevanten Zellen dominiert. Wir betrachten für die weitere Argumentation den an sich nicht legalen Zug

$$(g + 1, g + 1), (g, y_2) \rightarrow (g, g + 1), (g + 1, y_2). \tag{2.9}$$

Dieser Zug ist im ersten Diagramm von Abbildung 2.13 eingezeichnet. Man beachte, dass dieser Zug rechteckig ist, und eine der Mäuse eine Diagonalzelle

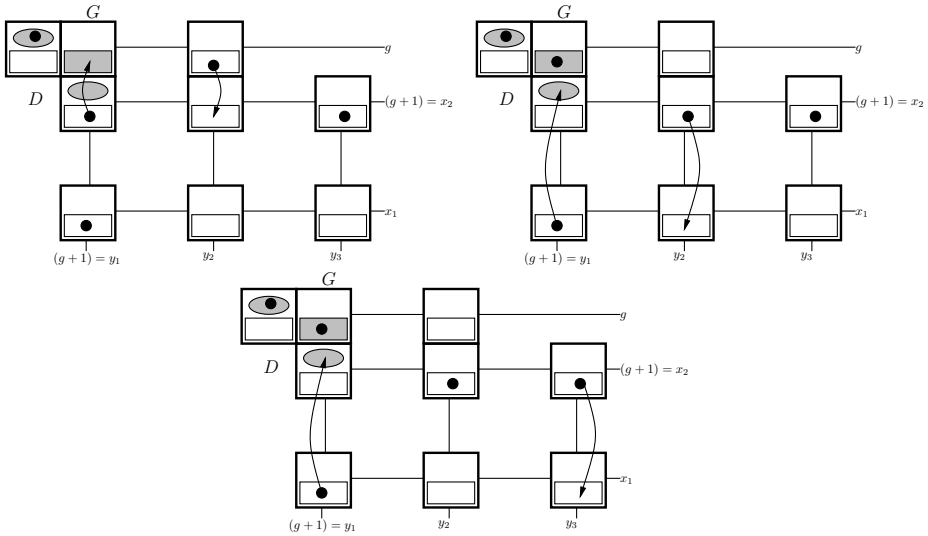


Abbildung 2.13: Schematische Darstellung der Situation bei den Schritten (6),(7) und (8).

verlässt. Da Diagonalzellen nicht zu den Rahmen irgendwelcher Teilbretter gehören können, ist dieser Zug trotz seiner Illegalität mit $I4$ verträglich.

In der Situation nach diesem Zug gibt es zwei mögliche Diagonal-Monges, die in den nächsten beiden Diagrammen eingezeichnet sind:

$$(x_1, g+1), (g+1, y_2) \quad \rightarrow \quad |s_{g+1}|, (x_1, y_2) \tag{2.10}$$
$$(x_1, g+1), (g+1, y_3) \quad \rightarrow \quad |s_{g+1}|, (x_1, y_3) \tag{2.11}$$

Nach Lemma 2.13 ist einer der beiden verträglich (sofern nicht $y_2 = y_3$ gilt und Lemma 2.15 greift).

In Kombination mit dem oben angegebenen illegalen Zug ergeben diese beiden Diagonal-Monges jedoch genau die beiden möglichen Verhaltensweisen des Algorithmus. Zug 2.9 gefolgt von 2.10 ergibt

$$(g+1, g+1), (g, y_2), (x_1, g+1) \rightarrow (g, g+1), |s_{g+1}|, (x_1, y_2),$$

was dem Greedy-Monge (7) mit anschließendem Periodenverzicht (8) entspricht. Zug 2.9 gefolgt von 2.11 ergibt als

$$(g+1, g+1), (g, y_2), (x_1, g+1), (g+1, y_3) \rightarrow (g, g+1), |s_{g+1}|, (x_1, y_3), (g+1, y_2)$$

den Tripelzug in G und D. Der Tripel ist nach Fallannahme aber nicht verträglich, also muss Zug 2.11 unverträglich sein. Damit ist 2.10 verträglich und somit auch der Greedy-Monge mit anschließendem Periodenverzicht, was zu zeigen war.

Für den **Greedy-Monge** in Schritt **(9)** ist Verträglichkeit im Algorithmus explizit abgefragt worden. Da beide benachbarten Diagonallöcher bereits eine Maus enthalten und G selbst keine Maus beherbergt, gibt es sowohl in der Zeile wie auch der Spalte von G genau eine noch bewegliche Maus, die auch wegen Lemma 2.16 zu einem Greedy-Monge verwendet werden können.

Ist der einzig mögliche Greedy-Monge nicht verträglich mit $I4$, muss die Rangabstiegsmethode in Schritt **(10)** die weiße Fahne hissen.

Wir fassen zusammen.

Lemma 2.17 *Die Rangabstiegsmethode ist wohldefiniert und bis zu ihrer Terminierung gelten alle Invarianten.*

Beweis: Gemäß Lemma 2.12 gelten die Invarianten nach der Initialisierung. Den Erhalt der Invarianten $I1$ bis $I3$ haben wir global verifiziert. Die Legalität der verlangten Züge und deren Verträglichkeit mit $I4$ wurden einzeln für jeden Zug eines Schleifendurchlauf gezeigt. Induktiv folgt, dass alle Invarianten bis zum Ende des Algorithmus erhalten bleiben, selbst dann, wenn die Rangabstiegsmethode im STOPP-Zustand endet. □

Zur Verdeutlichung lassen wir die Rangabstiegsmethode nun auf unserer Beispielinstanz aus Abschnitt 2.3 laufen. Abbildungen 2.14 und 2.15 zeigen einen von drei möglichen Verläufen.

2.4.3 Bilineare Greedyordnungen

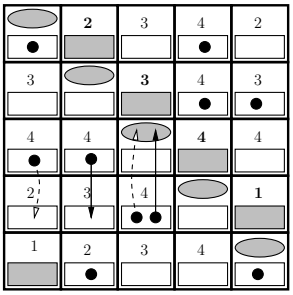
Im letzten Abschnitt haben wir gesehen, dass die Rangabstiegsmethode bis zu ihrer Terminierung alle Invarianten bewahrt. Der Algorithmus endet entweder nach der letzten Schleifeniteration oder vorzeitig durch das STOP Kommando im Senken-Fall.

Wir verifizieren nun, dass die Rangabstiegsmethode für bilineare Greedyordnungen in einem gewonnenen Spiel endet. Damit ist der Beweis für Theorem 2.1 komplett.

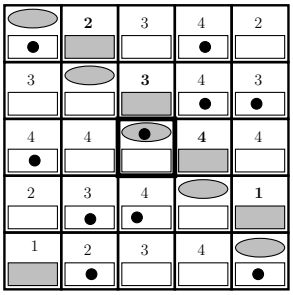
Lemma 2.18 *Die Rangabstiegsmethode gewinnt das Mäusespiel für bilineare Greedyordnungen für jedes Paar expandierender Kreisüberdeckungen.*

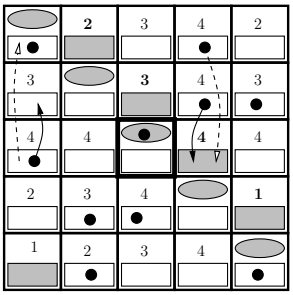
Beweis: Der Schlüssel zum Beweis ist die Handhabung des Senken-Falls.

Wir beginnen mit dem Spezialfall einer linearen Greedyordnung. Eine Greedyordnung ist linear, wenn der Algorithmus niemals zwei zusammengesetzte Strings der Eingabe verknüpft. Wann ist eine Greedyzelle $(i, i+1)$ eine Senke? Wenn sie von zwei höherrangigen Greedyzellen $(i-1, i)$ und $(i+1, i+2)$ umgeben ist. Das heißt zu dem Zeitpunkt, als der partielle Superstring, der mit s_i endet, mit dem, der mit s_{i+1} beginnt, verknüpft worden ist, war s_{i-1} bereits mit s_i und s_{i+2} bereits mit s_{i+1} verknüpft. Dann wären

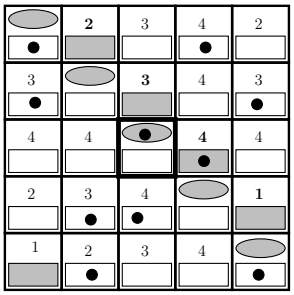


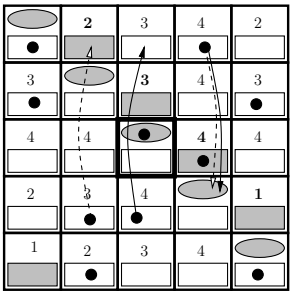
In der ersten Iteration $r = 4$ sind wir im Gipfel-Fall. Wir haben zwei mögliche Diagonal-Monges (2), die wegen Lemma 2.15 auch beide verträglich sind. Wir wählen den Zug mit den durchgezogenen Pfeilen.



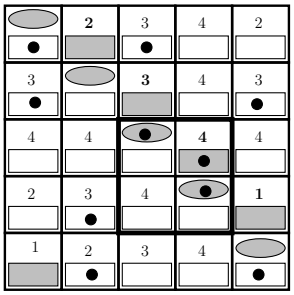


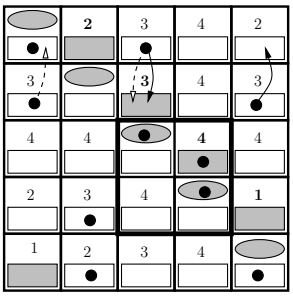
$G = (3, 4)$ und $D = (4, 4)$ sind beide frei von Mäusen. Ein Greedy-Monge (7) ist gefordert. Der Zug mit den gestrichelten Pfeilen führt zur Verwaisung des Rahmens von $Brett(\{1\})$. Also ist der andere auszuführen.





Beim Diagonal-Monge (8) ist wieder der gestrichelte Zug unverträglich, da der Rahmen von $Brett(\{3, 4\})$ verwaist. Wir haben also den anderen auszuführen. Damit endet der Schleifendurchlauf für $r = 4$.





Für $r = 3$ liegt der Hang-Fall vor. Einer der Greedy-Monges (7) führt zur Verwaisung des Rahmens von $Brett(\{1\})$. Wir ziehen also den anderen.

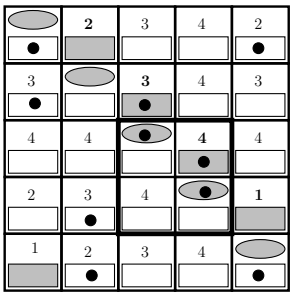
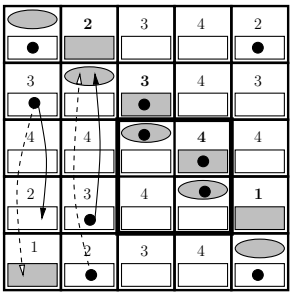
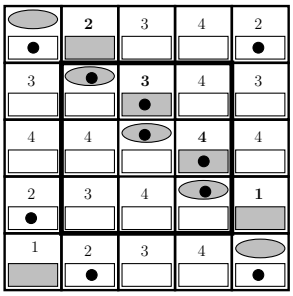
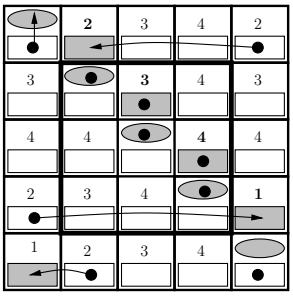


Abbildung 2.14: Testlauf für die Rangabstiegsmethode.

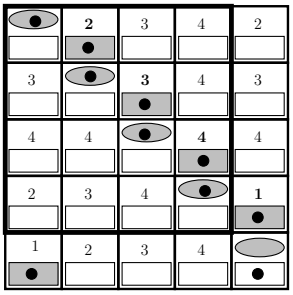


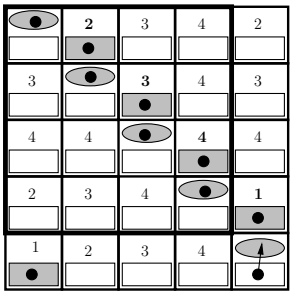
Beim sich anschließenden Diagonal-Monge (8) leert der gestrichelt dargestellte Zug den Rahmen von $Brett(\{2,3,4\})$. Wir sind also wieder auf den anderen festgelegt. Der zweite Durchlauf endet.





Für $r = 2$ liegt wieder der Hang-Fall vor. D enthält eine Maus und der Tripel (6), da er verträglich ist, wird gefordert. (Man beachte, dass der einzige mögliche Greedy-Monge unverträglich wäre.)





In der letzten Iteration $r = 1$ sind wir im Hang-Fall, G und D enthalten schon je eine Maus und wir werden zum Periodenverzicht (3) geführt. Die Rangabstiegsmethode endet mit einem gewonnenen Spiel.

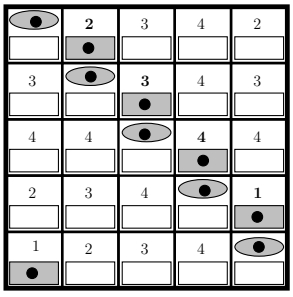


Abbildung 2.15: Testlauf für die Rangabstiegsmethode.

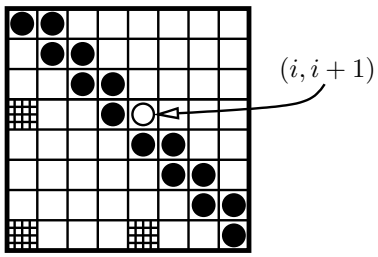


Abbildung 2.16: Der Senkenfall bei bilinearen Greedy-Ordnungen

aber beide von der Heuristik gewählten Strings zusammengesetzt gewesen – im Widerspruch zur Annahme einer linearen Ordnung.

Also tritt der Senken-Fall bei linearen Greedyordnungen nicht auf und die Rangabstiegsmethode läuft bis zur letzten Schleifeniteration durch und bewahrt dabei nach Lemma 2.17 die Invarianten. Das heißt, am Ende sind alle Löcher gefüllt ($I2$ und $I3$), und wegen $I1$ befindet sich die verbleibende Maus an Position $(n, 1)$, wo sie bei einem gewonnenen Spiel auch landen soll.

Bei einer echten bilinearen Greedyordnung, also einer, die nicht linear ist, werden im letzten Schritt zwei zusammengesetzte Strings verbunden. Im letzten Schleifendurchlauf für $r = 1$ tritt also der Senken-Fall ein. Sei $(i, i+1)$ die Greedyzelle mit Rang 1. Dann sind vor dem Schleifendurchlauf mit $r = 1$ alle Greedyzellen mit Rang 2 bis $n-1$ bereits gefüllt ($I2$), ebenso wie sämtliche Diagonalen ($I3$).

Damit ist in allen Zeilen außer i und n klar, wo sich die beiden Mäuse der Zeile befinden. In Zeile i ist eine Maus im Diagonalloch in (i, i). In Zeile n befindet sich eine Maus im Diagonalloch von (n, n). Ebenso ist in jeder Spalte außer 1 und $i+1$ klar, wo sich die beiden Mäuse befinden. In Spalte 1 ist nur die Maus im Diagonalloch von $(1, 1)$ festgelegt, in Spalte $i+1$ nur die Maus in $(i+1, i+1)$. Abbildung 2.16 stellt die Situation dar. Die beiden verbleibenden Mäuse müssen sich, da $I1$ gewahrt ist, auf den drei schraffierten Feldern oder in der Greedyzelle $(i, i+1)$ selbst befinden, wobei sie einander diagonal gegenüber sitzen müssen. Die beiden Mäuse können sich aber nicht auf den Positionen $(i, 1)$ und $(n, i+1)$ befinden, da dann der Rahmen von $Brett(\{1, \ldots, i\})$ verwaist wäre. Also liegen die Mäuse auf $(i, i+1)$ und $(n, 1)$. Die Rangabstiegsmethode stellt fest, dass nichts mehr zu tun ist, und der Algorithmus endet erfolgreich. □

Wir bestimmen nun die Anzahl der linearen und bilinearen Greedyordnungen.

Lemma 2.19 *Für n Strings gibt es $(n-1)!$ Greedyordnungen, von denen 2^{n-2} linear und $4^{n-3} + 2^{n-3}$ bilinear sind.*

Beweis: Die Zahl der Greedyordnungen entspricht der Zahl der Reihenfolgen, in die die Paare $(1,2),\ldots,(n-1,n)$ gebracht werden können.

Die Zahl der linearen Greedyordnungen zählt man wie folgt: Alle Greedyzellen mit Rang $> r$ bilden ein zusammenhängendes Teilstück der Nebendiagonale. Die Greedyzelle mit Rang r kann sich nun entweder oberhalb dieses Blocks oder unterhalb dieses Blocks anschließen. Zwischen diesen beiden Möglichkeiten ist $n-2$ mal zu wählen. Damit ist die Zahl der linearen Greedyordnungen eben 2^{n-2}.

Sei bei einer *echten* bilinearen Greedyordnung, also einer bilinearen Ordnung, die nicht linear ist, die Zelle $(i,i+1)$ vom Rang 1. Es werden die Strings $s_1,\ldots,s_i$ und die Strings $s_{i+1},\ldots,s_n$ in linearer Weise zusammengesetzt. Im letzten Schritt werden diese beiden partiellen Superstrings verknüpft.

Die Zahl der bilinearen Greedyordnungen lässt sich wie folgt angeben, wobei die Summenlaufvariable i jeweils die Zeile der Greedyzelle mit Rang 1 beschreibt.

$$\#(Bilinear) = 2^{n-2} + \sum_{i=2}^{n-2} \binom{n-2}{i-1} \cdot 2^{i-2} \cdot 2^{n-i-2}$$

Der Ausdruck 2^{n-2} zählt die linearen Greedyordnungen, die wir als Spezialfall abspalten. Ist die Greedyzelle mit Rang 1 in Zeile i, so beschreibt der Binomialkoeffizient die Zahl der Möglichkeiten, unter den restlichen $n-2$ Rängen diejenigen für die $i-1$ Positionen oberhalb von i auszuwählen. Die nächsten beiden Faktoren sind die Zahl der linearen Greedyordnungen oberhalb und unterhalb von Zeile i. Wir vereinfachen:

$$\begin{aligned}
\#(Bilinear) &= 2^{n-2} + \sum_{i=2}^{n-2} \binom{n-2}{i-1} \cdot 2^{i-2} \cdot 2^{n-i-2} \\
&= 2^{n-2} + \sum_{i=2}^{n-2} \binom{n-2}{i-1} 2^{n-4} \\
&= 2^{n-2} + 2^{n-4} \cdot \sum_{i=2}^{n-2} \binom{n-2}{i-1} \\
&= 2^{n-2} + 2^{n-4} \cdot \left(\sum_{i=0}^{n-2} \binom{n-2}{i} - \binom{n-2}{0} - \binom{n-2}{n-2} \right) \\
&= 2^{n-2} + 2^{n-4} \cdot \left(2^{n-2} - 2\right) \\
&= 2^{n-2} + 2^{2n-6} - 2^{n-3} \\
&= 4^{n-3} + 2^{n-3}
\end{aligned}$$

□

2.4.4 Beweis zu Korollar 2.1

Sei S_G der von der Greedy-Heuristik für die Menge $\mathcal{S} = \{s_1, \ldots, s_n\}$ gelieferte Superstring mit

$$|S_G| = \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n-1} (s_i, s_{i+1}).$$

Sei $S_{opt} = S_\pi$ der optimale Superstring mit

$$|S_{opt}| = \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n-1} (s_{\pi(i)}, s_{\pi(i+1)})$$

und C_{opt} eine optimale Kreisüberdeckung mit

$$|C_{opt}| = \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n} (s_i, s_{\tau(i)})$$

für die passende Bijektion τ. Dann schreibt sich die Aussage $|S_G| \leq |S_{opt}| + |C_{opt}|$ von Korollar 2.1 ausführlich als

$$\sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n-1} (s_i, s_{i+1}) \leq 2 \cdot \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n-1} (s_{\pi(i)}, s_{\pi(i+1)}) - \sum_{i=1}^{n} (s_i, s_{\tau(i)}),$$

was wir zu

$$\sum_{i=1}^{n-1} (s_{\pi(i)}, s_{\pi(i+1)}) + \sum_{i=1}^{n} (s_i, s_{\tau(i)}) \leq \sum_{i=1}^{n} |s_i| + \sum_{i=1}^{n-1} (s_i, s_{i+1}),$$

umformen. Während bei Ungleichung (2.4) beide Seiten $2n$ Terme enthalten, sind es hier auf beiden Seiten $2n - 1$. Der Greedy-Superstring S_G ist nicht zyklisch geschlossen, nutzt also die Eigenüberlappung (s_n, s_1) nicht aus. Analog sollte die Zelle $(n, 1)$ nicht als finale Mäuseposition auftreten. Genauso ist der uns unbekannte optimale Superstring S_{opt} nicht zyklisch geschlossen, benutzt also die Überlappung $(s_{\pi(n)}, s_{\pi(1)})$ nicht.

Um die Rangabstiegsmethode benutzen zu können, sind wir zu S_G° und S_{opt}° übergegangen, haben also quasi eine künstliche Maus (auf Position $(\pi(n), \pi(1))$) und ein künstliches Loch (Zelle $(n, 1)$) eingefügt. Wir haben jetzt zu zeigen, wie eine siegreiche Zugfolge für das so erweitertes Spiel in eine siegreiche Zugfolge für das ursprüngliche Spiel mit $2n - 1$ Mäusen und $2n - 1$ Löchern überführt werden kann. Ist dieser Schritt geleistet, so liefert die Rangabstiegsmethode auch Korollar 2.1. Das folgende Lemma liefert die nötigen Argumente.

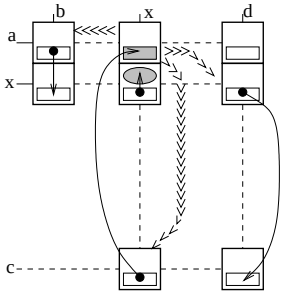
Lemma 2.20 *Sei A eine Konfiguration für das Mäusespiel, also eine Verteilung der $2n$ Mäuse auf dem Spielbrett. Sei B eine weitere Spielkonfiguration und sei A durch eine Folge von Diagonal- oder Greedy-Monges, Tripelzügen oder Periodenverzichten in B überführbar. Dann kann jede Konfiguration A', die aus A durch das Wegnehmen einer Maus hervorgeht, durch eine Folge legaler Züge in eine Konfiguration B' überführt werden, die aus B durch das Wegnehmen einer Maus hervorgeht.*

Beweis: Es genügt, das Lemma für Überführungen von A nach B, bestehend aus einem einzigen Zug, zu zeigen. Sei B also durch einen Zug Z von A aus erreichbar, und gehe A' aus A durch das Wegnehmen einer Maus M hervor.

Nimmt Maus M am Zug Z nicht teil, dann streiche diese Maus in B, um B' zu erhalten, und B' ist von A' aus mit Z zu erreichen. Ist die fehlende Maus M die einzige am Zug Z beteiligte Maus (d.h. Z ist ein Periodenverzicht), so lösche M in B, um B' zu erhalten, und es ist $A' = B'$.

Ist Z ein Diagonal-Monge, der M und eine weitere Maus M' in die Diagonalzelle (i, i) und die Zelle (j, k) bringt, so streiche in B die Maus in Zelle (j, k). A' ist in B' überführbar, indem man M' als Diagonaleinzug nach (i, i) bringt. Der Fall, dass Z ein Greedy-Monge ist, ist analog auf den Greedyeinzug zurückzuführen.

Falls Z ein Tripelzug ist, so muss unterschieden werden, welche Rolle M spielt. Wir beschreiben den vertikalen Tripel. Der horizontale ist analog zu behandeln.



Fall 1: M ist auf Position (x, x): Lösche (x, b) aus B'. A' kann durch den Diagonal-Monge

$$(x, d), (c, x) \rightarrow |x|, (c, d)$$

und den Greedyeinzug $(a, b) \rightarrow (a, x)$ in B' überführt werden.

Fall 2: M ist auf Position (x, d) [auf Position (c, x)]. Lösche (c, d) aus B, um B' zu erhalten. Um B' von A' aus zu erreichen, führe zunächst einen Greedyeinzug von (c, x) [von (x, d)], nach (a, x) aus. Dann benutzen wir den Zug aus Lemma 2.6, Teil 1, um die Maus von (a, b) nach (x, b) zu bringen, wobei wir mit der Periode in (x, x) bezahlen.

Fall 3: M ist in der Zelle (a, b). Wir löschen (x, b) aus B, um B' zu erhalten. Sei o.B.d.A $(x, d) \geq (c, x)$. Dann darf nach Lemma 2.6, Teil 4, (c, x) nach (c, d) gezogen werden, wenn wir dabei die Periode in (x, x) aufgeben. Die Maus von (x, d) wandert dann als Greedyeinzug nach (a, x). □

Durch Lemma 2.20 kann also nun ein gewonnenes Spiel für S_G°, S_{opt}° und C_{opt} in ein Spiel überführt werden, in dem die künstliche Maus in $(\pi(n), \pi(1))$ nicht verwendet wird, und eine Zielposition frei bleibt. Handelt es sich bei dieser Zielposition um $(n, 1)$, so sind wir sofort fertig. Ist das vakante Loch in einer Greedyzelle, so kann die Maus von $(n, 1)$ als Greedyeinzug dorthin gebracht werden, da Zelle $(n, 1)$ immer Rang 1 hat, also allen Greedyzellen rangmäßig unterlegen ist. Ist ein Diagonalloch vakant, so kann die Maus von $(n, 1)$ mit einem Greedyeinzug zunächst auf eine der leeren Diagonalzelle benachbarten Greedyzelle gebracht werden und von dort mit einem Diagonaleinzug ins Diagonalloch.

Damit haben wir ein gewonnenes Spiel für S_G, S_{opt} und C_{opt} erhalten, und Korollar 2.1 folgt.

2.5 Optimale Kreisüberdeckungen als Mäusespiel

Wir zeigen in diesem Abschnitt, wie die (bereits bekannte (BLTY94)) Tatsache, dass die modifizierte Greedy-Heuristik (Algorithmus 2.2) optimale Kreisüberdeckungen liefert, mit Hilfe des Mäusespiels bewiesen werden kann. Ist τ_G die Bijektion, die die von der Greedy-Heuristik gefundene Kreisüberdeckung beschreibt, soll also

$$|C_{\tau_G}| = |C_{opt}|$$

gezeigt werden, was wir äquivalent als

$$\forall \tau \ : \ |C_{\tau_G}| \leq |C_\tau|$$

und ausführlich

$$\begin{aligned} \forall \tau \ : \ \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n} (s_i, s_{\tau_G(i)}) \ &\leq \ \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n} (s_i, s_{\tau(i)}) \\ \forall \tau \ : \ \sum_{i=1}^{n} (s_i, s_{\tau(i)}) \ &\leq \ \sum_{i=1}^{n} (s_i, s_{\tau_G(i)}) \end{aligned}$$

schreiben können. Die letzte Ungleichung entspricht der Ungleichung (2.4) im Falle der verallgemeinerten Greedy-Conjecture.

Wir erkennen die ersten wesentlichen Änderungen.

- Für die Verifikation der Optimalität der Greedy-Heuristik für Kreisüberdeckungen sind nur n Terme durch n andere Terme abzuschätzen. Es gibt also nur n Mäuse und n Zielfelder.

- Die Stringlängen spielen keine Rolle. Wir arbeiten nur mit Überlappungen zweier Strings bzw. mit Eigenüberlappungen.

Ein Effekt, der das Mäusespiel für Kreisüberdeckungen unübersichtlicher macht, ist die Position der Löcher. Während wir bei den Strings o.B.d.A. annehmen konnten, dass die Greedy-Heuristik den String $s_1, \ldots, s_n$ liefert, sich die Löcher also auf den Nebendiagonalen befinden, so wissen wir hier nicht, wieviele und wie große Kreise die Greedy-Heuristik schließt. Wir belassen es also dabei, dass von den n Löchern in jeder Zeile und jeder Spalte genau eines ist, ohne dass wir genauer wissen wo.

Dass die Greedy-Heuristik Kreise schließen darf, hat für uns die folgenden wesentlichen Konsequenzen:

- Die Eigenüberlappungen (s_i, s_i) sind Bestandteil der Greedyordnung. Da in Algorithmus 2.2 $a = b$ anders als bei der Stringvariante erlaubt ist, ist eine Eigenüberlappung allen Zellen unterlegen, die von der Greedy-Heuristik vor dem Zeitpunkt, in der die Eigenüberlappung illegal wurde, gewählt worden sind. Hier erhalten also auch Diagonalzellen einen Rang, der sich auf die betreffende Eigenüberlappung bezieht.

- Alle Felder mit Rang r befinden sich in der Zeile oder der Spalte des Lochs mit Rang r: (Vergleiche im Gegensatz dazu Lemma 2.4.) Werden zwei partielle Superstrings $s_i, \ldots, s_j$ und $s_{j+1}, \ldots, s_k$ vereint, so fällt die Überlappung (s_k, s_i) im Kontext von Kreisüberdeckungen nicht aus der Menge der noch nutzbaren Überlappungen.

Natürlich behalten alle im Abschnitt 2.3.1 eingeführte Züge ihre Gültigkeit, da diese Eigenschaften von Strings modellieren. Der Greedy-Monge allein genügt uns hier aber. Algorithmus 2.5 beschreibt das vergleichsweise simple Vorgehen zum Gewinn des Spiels. Wir sammeln die nötigen Argumente für die Korrektheit: Wir haben nach der Initialisierung eine Maus pro Zeile und Spalte, woran sich auch im Spielverlauf nichts ändert. Ist die Greedyzelle vom Rang r mit einer Maus gefüllt, so befindet sich nirgendwo sonst auf dem Brett noch eine Maus in einer Zelle vom Rang r: Alle Felder mit Rang r befinden sich in derselben Zeile oder derselben Spalte wie die gefüllte Greedyzelle.

Sieht der Algorithmus die Füllung des Lochs mit Rang r vor, so sind alle höherrangigen Löcher bereits gefüllt. Es gibt also keine frei herumlaufenden Mäuse mehr, die von der Greedyzelle mit Rang r nicht bewegt werden könnten. Die Ausführbarkeit der Greedy-Monges ist also sichergestellt. Ist das letzte Loch gefüllt, so ist das Spiel gewonnen. Damit haben wir einen alternativen Beweis für das folgende Lemma.

Lemma 2.21 *Die modifizierte Greedy-Heuristik 2.2 findet optimale Kreisüberdeckungen für Strings.*

Die Rangabstiegsmethode für Kreisüberdeckungen

1. **Eingabe**: Ein Greedy-Kreisüberdeckung C_G nebst entsprechender Greedyordnung und eine Kreisüberdeckung C_τ.
2. **Initialisierung**
   (a) Lege das Spielbrett an. In Zelle (i, j) ist ein Loch, wenn s_j direkter Nachfolger von s_i in C_G ist. Vergib die Ränge gemäß der Greedyordnung.
   (b) Für i von 1 bis n
      i. Sei s_j der Nachfolger von s_i in C_τ.
      ii. Platziere eine Maus in der Zelle (i, j). Ist $i = j$ so wird die Maus auf dem Rechteck für die Eigenüberlappung eingebracht.
3. **Hauptschleife** Für r von $n - 1$ bis 1
   (a) Falls das Loch vom Rang r noch keine Maus enthält:
      i. Greedy-Monge in der Greedyzelle vom Rang r.

Algorithmus 2.5: Die Rangabstiegsmethode für Kreisüberdeckungen.

2.6 Die Grenzen der betrachteten Ungleichungssysteme

In diesem Abschnitt sollen die Grenzen der verwendeten Ungleichungsfamilien (bzw. in der Sprache des Mäusespiels: die Grenzen der dargestellten Züge) ausgelotet werden.

Bereits in Abschnitt 2.3.6 haben wir gezeigt, dass die verschärfte Greedy-Conjecture für eine konkrete Instanz (Greedyordnung und expandierendes Paar von Kreisüberdeckungen) mittels linearer Programmierung verifiziert ist, wenn das dort beschriebene lineare Programm ein Optimum ≥ 1 liefert.

Nehmen wir nun an, dass das dort beschriebene lineare Programm für eine Instanz eine Lösung kleiner als 1 liefert. Das heißt, es ist dem linearen Programm gelungen, den Variablen für die Stringlängen und den Überlappungen Werte zuzuweisen, so dass der Ausdruck, der die Länge des Greedy-Superstrings beschreibt, die Summe der Größen der Kreisüberdeckungen übertrifft. In diesem Fall sind die ins lineare Programm eingeflossenen Gleichungen nicht ausreichend, um die Greedy-Conjecture für die vorliegende Instanz zu verifizieren.

Als Erstes ist hier daran zu erinnern, dass das lineare Programm von

der Menge der bedingten Ungleichungen $B(n)$ nur die Teilmenge $B_{\mathcal{G}}(n)$ der bedingten Ungleichungen enthält, deren Prämisse a priori durch die Greedyordnung verifiziert werden konnte. Es ist möglich, dass die Wertzuweisung durch das lineare Programm die Prämissen weiterer bedingter Ungleichungen erfüllt, deren Konklusion jedoch nicht. Soll gezeigt werden, dass eine bestimmte Menge von bedingten und unbedingten Ungleichungen inhärent ungenügend ist, um die Greedy-Conjecture in der ein oder anderen Form zu beweisen, muss dieser Effekt eliminiert werden.

In (WS03) ist es gelungen, eine Matrix zu finden, die sämtliche unbedingten Ungleichungen aus Abschnitt 2.3.2 (also Inklusionen wie $(a, b) < |b|$, Periodenverzichte und Diagonal-Monges) ebenso erfüllt wie die Greedy-Monges, einer linearen Greedyordnung und die Ungleichungen der Greedyordnung selbst. Dennoch ergibt die Matrix einen Approximationsfaktor von $2\frac{1}{3}$. Tabelle 2.2 zeigt diese Matrix, die aber nicht von Strings erzeugbar ist, da die Tripelungleichung verletzt ist.

In den Diagonalzellen ist oben die Stringlänge $|s_i|$ und unten die Eigenüberlappung (s_i, s_i) eingetragen, in den übrigen Zellen ist oben der Rang der Zelle und unten die Überlappung notiert.

Man erkennt, dass die Stringlängen die größten Einträge ihrer Zeilen und Spalten sind, damit sind die Inklusionsbeziehungen $(a, b) < |a|, |b|$ erfüllt. Weiterhin sind die Einträge verträglich mit der linearen Greedyordnung

$$(1, 2), (2, 3), \ldots, (8, 9).$$

Sodann läßt sich enumerativ, etwa durch den Einsatz von Rechenkraft und eines Testprogramms, verifizieren, dass jede 2×2 Teilmatrix die Monge-Ungleichung erfüllt.

Die Summe der Stringlängen ist $15i + 44$. Die vom Greedy-Superstring verwendeten Überlappungen der Nebendiagonale addieren sich zu $8i + 20$. Gelänge es eine Menge von 9 Strings zu konstruieren, die die Vorgaben der Matrix realisieren, so würde die Greedy-Heuristik einen Superstring der Länge

$$15i + 44 - (8i + 29) = 7i + 24$$

liefern. Der Superstring S_π mit $\pi = (1, 9, 2, 6, 3, 8, 4, 7, 5)$, der die Überlappungen in den mit zwei Mäusen versehenen Zellen ausnutzt, hätte dagegen eine Länge von nur

$$15i + 44 - (12i + 12) = 3i + 32.$$

Da i beliebig gewählt werden kann, ließe sich der Approximationsfaktor von unten beliebig nahe an $2\frac{1}{3}$ zwingen.

Für den Nachweis, dass auch diese alternative Reihenfolge π der Strings bei der gegebenen Greedyordnung kein Gegenbeispiel zulässt, müssen also Eigenschaften von Strings, die über simple Inklusionen und Monge-Ungleichungen hinausgehen, verwendet werden. Ein Beweis der Greedy-Conjecture

2i + 5 **(2i+5)**	8 2i + 4	8 i	8 0	8 0	8 i	8 0	8 0	● 8 ● 2i
8 0	**3i + 8** **(3i+8)**	7 2i + 4	7 i	7 i	● 7 ● 2i + 3	7 i	7 i	7 0
7 0	8 0	**2i + 8** **(2i+8)**	6 i + 4	6 i + 1	6 i	6 i + 1	● 6 ● i + 1	6 0
6 0	8 0	7 i	**i + 7** **(i+7)**	5 i + 4	5 i	● 5 ● i + 3	5 0	5 0
5 0	8 0	7 i	6 0	**i + 6** **(i+6)**	4 i + 2	4 0	4 0	4 0
4 0	8 0	● 7 ● 2i + 2	6 i	5 i	**2i + 5** **(2i+5)**	3 i + 2	3 i	3 0
3 0	8 0	7 i + 1	6 0	● 5 ● i + 2	4 i + 1	**i + 4** **(i+4)**	2 0	2 0
2 0	8 0	7 i	● 6 ● i + 1	5 i + 1	4 i	3 i + 1	**i + 1** **(i+1)**	1 0
1 0	● 8 ● 2i	7 i	6 0	5 0	4 i	3 0	2 0	**2i** **(2i)**

Tabelle 2.2: Eine Matrix, die alle Inklusionen, Periodenverzichte und Monge-Ungleichungen einer linearen Greedyordnung erfüllt und dennoch einen Approximationsfaktor von $2\frac{1}{3}$ verspricht. Sie belegt die Notwendigkeit der Tripelungleichung selbst für die klassische Form der Greedy-Conjecture für lineare Greedyordnungen.

selbst in der schwächeren klassischen Form ist mit Inklusionen, den Ungleichungen der Greedyordnung und Monge-Ungleichungen alleine also auch für lineare Greedyordnungen nicht möglich.

Die Tripelungleichung ist also in diesem Sinne wesentlich für unser Ergebnis.

Dass es für einen allgemeinen Beweis der klassischen Greedy-Conjecture weiterer – über die Tripelungleichung hinausgehende – Eigenschaften bedarf, belegt Tabelle 2.3, die in (LW04; LW05) angegeben wurde.

Die Darstellung entspricht der von Tabelle 2.2. Die zugrundeliegende Greedyordnung ist $(7,8)$, $(3,4)$, $(2,3)$, $(9,10)$, $(1,2)$, $(8,9)$, $(4,5)$, $(6,7)$, $(5,6)$. Diese Ordnung ist nicht bilinear, da bereits im sechsten Schritt, der Wahl von $(8,9)$, zwei zusammengesetzte Strings verknüpft werden. Alle Inklusionen, Monge-, Greedyordnungs- **und** alle Tripelungleichungen sind erfüllt, wie man sich enumerativ überzeugen kann. Die Längen der Strings addieren sich zu $30i + 62$. Die vom Greedy-Superstring ausgenutzten Überlappungen der Nebendiagonale addieren sich zu $17i+28$. Der von der Greedy-Heuristik gelieferte Superstring hätte also eine Länge von

$$30i + 62 - (17i + 28) = 13i + 34.$$

Der alternative Superstring S_π für $\pi = (1, 2, 10, 9, 3, 8, 7, 4, 6, 5)$ nutzt Überlappungen mit einer Gesamtlänge von $24i+28$ und kommt so auf eine Länge von

$$30i + 62 - (24i + 28) = 6i + 34.$$

Der Approximationsfaktor könnte also auf $2\frac{1}{6}$ getrieben werden, wenn es gelänge Strings mit den Vorgaben der Tabelle 2.3 zu konstruieren.

Damit ist nachgewiesen, dass für einen allgemeinen Beweis der klassischen Greedy-Conjecture über die Tripelungleichung hinausgehende Eigenschaften von Strings einbezogen werden müssen.

Ist man an der starken Form aus Theorem 2.1 interessiert, so stellt sich der erste nicht abgedeckte Fall bereits bei $n = 5$ Strings ein. Tabelle 2.4 stellt ihn dar.

Wieder sind alle besprochenen Ungleichungen inklusive der Tripelungleichung erfüllt. Die Greedyordnung ist $(2,3)$,$(4,5)$,$(3,4)$,$(1,2)$. Die Summe der Stringlängen ist $38i+10$. Die Überlappungen der Nebendiagonale sind $16i+2$. Damit kommt der Greedy-Superstring auf eine Länge von

$$38i + 10 - (16i + 2) = 22i + 8.$$

Der alternative Superstring S_π mit $\pi = (5, 4, 3, 2, 1)$ – die geschlossenen Kreise in Tabelle 2.4 – hat eine Länge von

$$38i + 10 - (25i + 2) = 13i + 8.$$

i + 5	● 5 ●	7	8	5	5	5	9	5	6
(0)	i+4	i+3	i+1	0	0	0	i+1	i+2	i+1
7	**3i + 10**	7	8	7	7	7	9	7	● 7 ●
i+1	**(i+1)**	3i+9	3i+3	i	i	2i+2	3i+3	3i+5	3i+4
8	8	**4i + 10**	8	8	8	8	● 9 ●	8	8
0	0	**(0)**	4i+4	i	i	3i	4i+2	i	i
5	7	8	**4i + 5**	3	● 3 ●	3	9	4	6
0	0	0	**(i)**	i	i	0	i	i	i
3	5	7	8	**i + 1**	1	2	9	4	6
0	0	0	0	**(0)**	0	0	0	0	0
2	5	7	8	● 3 ●	**i + 1**	2	9	4	6
0	0	0	i	i	**(i)**	0	i	i	i
9	9	9	● 9 ●	9	9	**4i + 6**	9	9	9
0	0	0	4i+2	i	i	**(3i)**	4i+4	i	i
4	5	7	8	4	4	● 9 ●	**5i + 6**	4	6
0	0	i+2	4i	i	i	4i+2	**(4i)**	i+1	i
6	6	● 7 ●	8	6	6	6	9	**4i + 10**	6
i+1	i+1	3i+8	3i+3	i	i	2i+2	3i+3	**(3i+4)**	3i+6
1	5	7	8	3	2	4	9	● 6 ●	**3i + 8**
0	i+2	2i+4	2i+2	0	0	i+1	2i+2	3i+6	**(2i+2)**

Tabelle 2.3: Die Matrix, die alle Inklusionen, Monge-, Greedyordnungs- und Tripelungleichungen erfüllt und dennoch einen Approximationsfaktor von $2\frac{1}{6}$ verspricht. Für die klassische Greedy-Conjecture auf allgemeinen Ordnungen sind also Eigenschaften über die Tripelungleichung hinaus nötig.

5i + 2 (5i ○)	1 0	4 i	2 2i	● 3 5i
● 4 3i	**8i + 2** (6i ○)	4 7i+1	4 2i	4 3i
2 2i	● 4 7i	**8i + 2** (6i ○)	2 2i	3 2i
3 0	3 2i+1	● 4 3i+1	**7i + 2** (4i+1)	3 ○ 7i+1
1 0	2 2i	4 3i	● 3 ○ 7i+1	**10i + 2** (7i)

Tabelle 2.4: Für die verallgemeinerte Greedy-Conjecture bedarf es bereits bei $n = 5$ Argumenten, die über die Tripelungleichung hinausgehen.

Die Kreisüberdeckung $\{(s_1), (s_2), (s_3), (s_4, s_5)\}$ – die offenen Kreise in Tabelle 2.4 – nutzt Überlappungen mit einem Gesamtgewicht von $31i + 2$ aus, kommt also auf eine Länge von

$$38i + 10 - (31i + 2) = 7i + 8.$$

Damit sind alternativer Superstring und Kreisüberdeckung zusammen nur $20i + 16$ groß und damit kleiner als der von der Greedy-Heuristik gefundene Superstring.

2.7 Zusammenfassung

Das zentrale Thema dieses Kapitels ist das Problem kürzester Superstrings. Seit der bahnbrechenden Arbeit von Blum et al. war bekannt, dass der Approximationsfaktor der Greedy-Heuristik höchstens vier beträgt. Die bis heute unwiderlegte Vermutung, dass die Greedy-Heuristik sogar 2-approximativ ist, steht sogar noch etwas länger im Raum (TU88). Erst 2005 ist die obere Schranke auf 3,5 verbessert worden (KS05). Fortschritte auf dem Gebiet des Problems kürzester Superstrings bestanden in der Zwischenzeit im wesentlichen in der Entwicklung von Approximationsalgorithmen, deren garantierter Approximationsfaktor sich im Laufe der Zeit von 4 langsam bis 2.5 (Sw99) verbesserte. Diese Algorithmen werden dabei mit sinkendem Approximationsfaktor kontinuierlich filigraner und aufwendiger, wodurch sie in immer schärferem Kontrast zur sehr simplen, aber nicht hinreichend gut verstandenen, Greedy-Heuristik erscheinen.

Wir haben versucht, das Verständnis darüber zu vertiefen, wann und warum die Greedy-Heuristik innerhalb der Faktor-2-Schranke verbleibt, und welche Eingaben für sie überhaupt potenziell schwierig sind.

Wesentlich für unser Vorgehen waren die Greedyordnungen. Jede Mengen von n Strings liefert eine von $(n-1)!$ Greedyordnungen und wir haben nur aus der Greedyordnung ableitbares Wissen über die Eingabestrings eingesetzt. Es hat sich gezeigt, dass bei bilinearen Greedyordnungen dieses Wissen ausreicht, um eine verschärfte Form der Greedy-Conjecture, nämlich

$$|S_G| \leq |S_{opt}| + |C_{opt}|,$$

zu beweisen. Durch die Verwendung von Greedyordnungen konnten wir bedingte Ungleichungen verwenden. Da alle etablierten Ungleichungsfamilien wie Inklusions- oder Monge-Ungleichungen sowie die neu gefundene Tripelungleichung linear in den Stringlängen und Überlappungen sind, haben wir lineare Programmierung zum raschen Verifizieren der Greedy-Conjecture für eine gegebene Instanz eingesetzt.

Um Aussagen über allgemeine Stringanzahlen n machen zu können, müssen jedoch ganze Familien von linearen Programmen analysiert werden. Hier haben wir mit dem Mäusespiel eine kombinatorische Anschauung für das duale Problem gefunden und können durch ein gewonnenes Spiel, mittels schwacher Dualität, eine Instanz als unkritisch für die Greedy-Conjecture erkennen.

Durch die Angabe einer Gewinnstrategie für das Mäusespiel bei bilinearer Greedyordnung konnte dann die Gültigkeit der verschärften Greedy-Conjecture für diese Teilklasse bewiesen werden, wobei wir sogar zeigen, dass die Länge des Greedy-Superstrings durch die Summe der Größen zweier Kreisüberdeckungen mit der Expansionseigenschaft beschränkt ist.

Für die Optimalität der Greedy-Heuristik bei der Suche nach minimalen Kreisüberdeckungen konnten wir mit Hilfe des Mäusespiels einen alternativen Beweis finden.

Darüberhinaus kann die Formulierung als lineares Programm genutzt werden, um zügig interessante Instanzen ausfindig zu machen. Die Tripelungleichung wurde beispielsweise gefunden, nachdem festgestellt wurde, dass die Sackgassen in den mit Monge-Ungleichungen allein nicht gewinnbaren Mäusespielen strukturelle Ähnlichkeiten aufweisen. Die Nützlichkeit der Tripelungleichung erkannt zu haben, erscheint verglichen mit dem Beweis der Ungleichung selbst ein mindestens ebenso großer Beitrag. Die Evaluierung des linearen Programms für eine gegebenen Konfiguration ist die Kernfunktion des von Uli Laube im Rahmen der Arbeit an dem Projekt entwickelten Software Tools SINDBAD (La04). Abb. 2.17 zeigt die Benutzeroberfläche von SINDBAD. Durch enumeratives Durchsuchen von interessanten Eingaberäumen, können nicht abgedeckte Instanzen relativ schnell ausfindig gemacht werden. Die in Abschnitt 2.6 angegebene Matrix zum Aufzeigen der

Grenzen der Tripelungleichung basieren auf Suchergebnissen von SINDBAD. Weitere Features werden in (LW04) beschrieben.

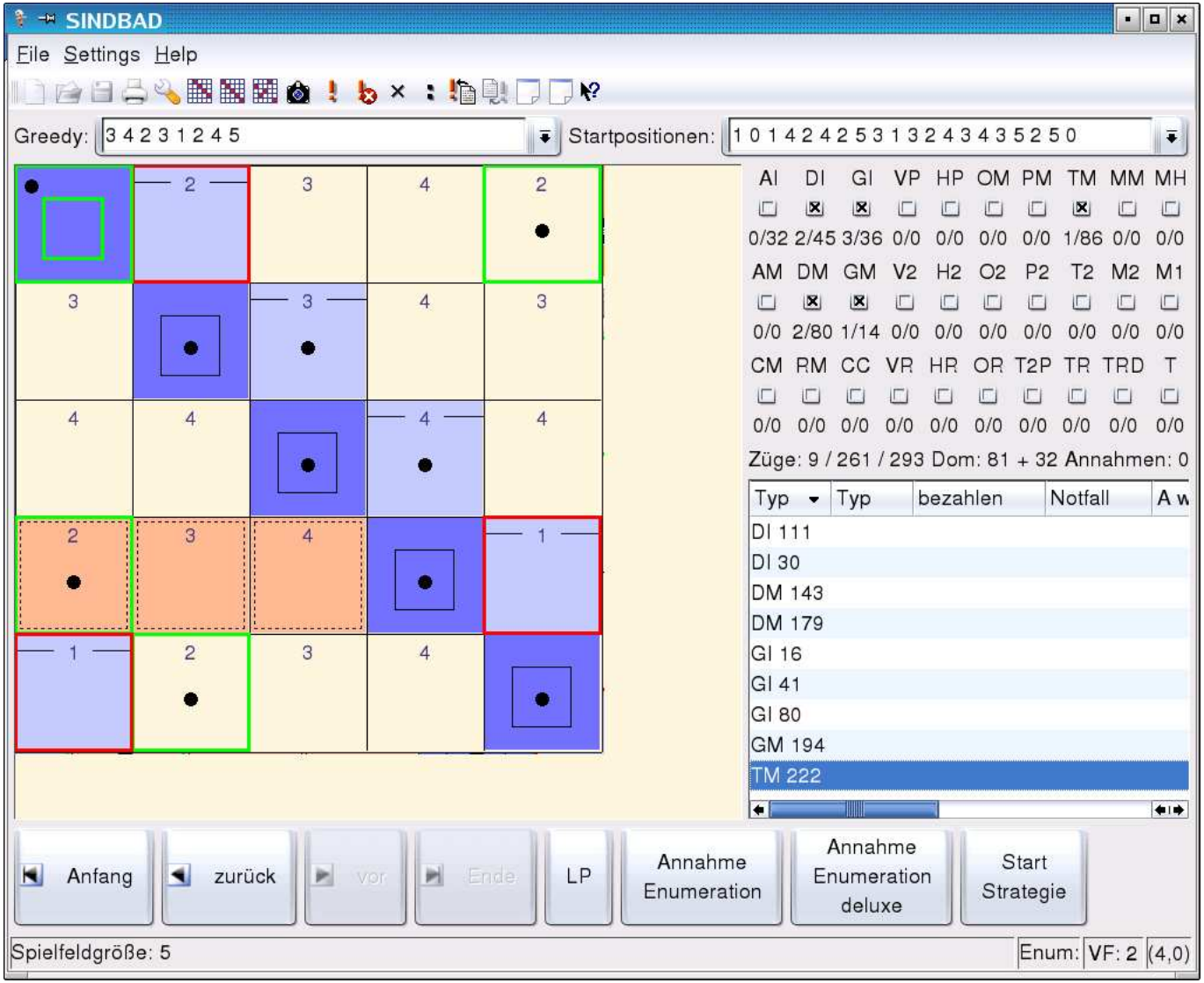


Abbildung 2.17: Die Benutzeroberfläche von SINDBAD (La04)

Kapitel 3

Packet Queueing

In diesem Kapitel werden Queueing Strategien für den Datentransport in Netzwerken untersucht. Informationsblöcke, im Folgenden einfach Pakete genannt, sollen zwischen Knoten des Netzwerks ausgetauscht werden. Routingverfahren weisen den Paketen einen einfachen Weg von ihrem Ursprungsort zum Bestimmungsort zu. Queueing Strategien, um die es in diesem Kapitel geht, regeln die *Vorfahrt* zwischen mehreren Paketen einer Queue, also den Paketen, die zur selben Zeit dieselbe Verbindung nutzen möchten.[1]

Anders als beim Problem kürzester Superstrings, bei dem Heuristiken als Reaktion auf die inhärente Härte des Problems herangezogen werden, ist hier der Einsatz von Heuristiken vor allem durch drei anwendungsbedingte Forderungen gerechtfertigt.

Lokalität: Die einzelnen Knoten eines Netzwerkes müssen autark arbeiten. Ein einzelner Rechner ist niemals über den kompletten Zustand des Netzwerks informiert. Insbesondere weiß ein Rechner nicht, welche Pakete andernorts eingefügt worden sind, und welche Verbindungen gerade besonders hoch belastet sind.

Online-Betrieb: Eingaben erfolgen bei laufendem Betrieb und es ist nicht möglich, bereits getroffene Entscheidungen wieder zurückzunehmen. Ein Rechner weiß nicht, welche Pakete in der Zukunft eingefügt werden. Entsprechend wird ein Rechner zwangsläufig Entscheidungen treffen, die sich im Nachhinein als nachteilig erweisen.

Einfachheit : Die Weiterleitung von Paketen muss mit hoher Geschwindigkeit erfolgen. Muss ein Rechner zur Ermittlung seines weiteren Vorgehens aufwendige Berechnungen anstellen, so ist der Vorteil der gefundenen Lösung unter Umständen bereits durch diese Rechenzeit aufgehoben.

[1] Wir benutzen hier den Begriff Queue als Beschreibung der Menge der wartenden Pakete und nicht wie üblicherweise als Bezeichnung einer First-In-First-Out Datenstruktur der Menge.

Das konkrete Problem, das in diesem Abschnitt untersucht wird, ist die Analyse von Queueing Strategien im Adversarial Queueing Modell, das als Methode zur Modellierung von Worst-Case-Szenarios vorgeschlagen wurde (BKR$^+$01) und in den letzten 10 Jahren den formalen Rahmen für eine Vielzahl von Untersuchungen geliefert hat (BKR$^+$01; AAF$^+$01; Ga03; AR02; BOR04; BG03; KMS03; AZ04; AFGZ01; RT04). Im Abschnitt 3.1 wird das Modell genauer eingeführt, und wesentliche bekannte Fakten werden benannt. In den darauffolgenden Abschnitten wird die Klasse von Strategien ohne Zeitnahme eingeführt und analysiert. Diese größtenteils aus (We05) stammenden Resultate sind im Einzelnen:

1. Jede Queueing Strategie ohne Zeitnahme kann zu einer exponentiell großen Queue und damit zu exponentiell großer Verzögerung (im Durchmesser und der Knotenzahl des Netzwerks) gezwungen werden. (Theorem 3.1, Abschnitt 3.2). Dies war bisher nur für konkrete prominente Strategien bekannt.

2. Es wird eine neue Technik zur Feststellung der Stabilität von Queueing Strategien ohne Zeitnahme vorgestellt, die Aufschichtungskreise. (Theorem 3.2, Abschnitt 3.3). Mit ihrer Hilfe können bekannte Stabilitätsbeweise prominenter Strategien vereinheitlicht werden und weitere Stabilitätsergebnisse erzielt werden.

3. Die große Teilklasse distanzbasierter Queueing Strategien wird in Abschnitt 3.4 untersucht. Hier gelingt eine vollständige Klassifizierung aller 1-stabilen und universell stabilen Strategien. (Theorem 3.3)

4. Bei jeder Strategie ohne Zeitnahme können Pakete endlos im Netzwerk steckenbleiben, wenn mit Rate 1 eingefügt wird. (Lemma 3.3, Abschnitt 3.5).

3.1 Das Adversarial Queueing Modell

Im Adversarial Queueing Modell (BKR$^+$01) wird ein Netzwerk durch einen Graphen repräsentiert. Die Knoten sind die Router, und die Kanten sind die bestehenden Verbindungen zwischen den Routern. Vereinfachend wird angenommen, dass alle zu transportierenden Pakete gleiche Größe haben, und alle Verbindungen genau ein Paket pro Zeittakt transportieren können. Beide Annahmen stellen keine wesentliche Einschränkung dar (BOR04). Startknoten, Zielknoten, Zeitpunkt der Bereitstellung des Pakets am Startknoten sowie der zu benutzende Pfad vom Start zum Ziel werden von einem Gegner gewählt.

Dem Gegner wird auferlegt, dass nur einfache Pfade zugewiesen werden. Kein Paket durchläuft also denselben Knoten zweimal, was einer sinnlosen Schleife entspräche. Selbstverständlich muss die Macht eines solchen

Gegners darüber hinaus eingeschränkt werden, da es für den Gegner sonst ein Leichtes wäre, durch Überbelastung einzelner Kanten jede noch so gute Queueing Strategie in die Knie zu zwingen; Übersteigt irgendwo im System die Nachfrage dauerhaft die Kapazität, so kann das Problem nicht durch kluge Organisation gelöst werden.

Definition 3.1 *Ein (r, b)-Gegner fügt während eines jeden Zeitintervalls I für jede Kante e höchstens $r \cdot |I| + b$ Pakete, die Kante e traversieren, in das Netzwerk ein. Wir nennen r die Rate des Gegners und b seine Burstiness.*

Für $r > 1$ kann ein Gegner dauerhaft mehr Pakete einfügen, die eine bestimmte Kante traversieren sollen, als die Kante in derselben Zeit bedienen kann. Jeder Versuch durch eine raffinierte Queueing Strategie die Größe des Staus vor dieser Kante zu beschränken, ist also konzeptionell zum Scheitern verurteilt.

Für $r \leq 1$ allerdings zeigen verschiedene Queueing Strategien sehr unterschiedliches Verhalten, was ihre Fähigkeit betrifft, Paketstaus der Größe nach zu beschränken: Die Strategie Nearest-To-Source (NTS) beispielsweise, die immer dem Paket mit der geringsten Zahl bisher traversierter Kanten Vorfahrt gewährt, hält auf jedem Graphen G gegen jeden $(1, b)$-Gegner die Gesamtzahl der Pakete im System beschränkt (Ga03); für jeden Graphen G und jeden Wert für b gibt es eine Schranke $c_{G,b}$, so dass die Zahl der Pakete im System $c_{G,b}$ nicht übersteigt, unabhängig davon wie lange das System dem $(1, b)$-Gegner ausgesetzt ist.

Im Kontrast dazu gibt es aber selbst für beliebig kleine Raten $r > 0$ sogar planare Graphen, auf denen die populäre Strategie First-In-First-Out (FIFO) unbeschränkt große Staus produziert (BG03; KMS03); das heißt, die Zahl der Pakete im System kann mit der Zeit über jede beliebige Schranke getrieben werden.

Den Begriff der Robustheit einer Strategie gießen wir in die folgende Definition.

Definition 3.2 (Stabilität)

1. *Eine Queueing Strategie ist r-stabil, wenn für jeden Graphen G und jedes $b \in \mathbb{N}$ eine Schranke $c_{G,r,b}$ existiert, so dass für jede Folge von Einfügungen durch einen (r, b)-Gegner in G die Gesamtzahl der Pakete in G niemals $c_{G,r,b}$ übersteigt.*

2. *Eine Queueing Strategie ist universell stabil, wenn sie r-stabil für jedes $r < 1$ ist.*

In (AAF+01) und darauf folgenden Arbeiten wird eine Reihe prominenter Queueing Strategien auf Stabilität untersucht. Tabelle 3.1 gibt einen Überblick.

Alle prominenten Queueing Strategien sind *greedy* Queueing Strategien, in dem Sinne, dass keine Kantenkapazitäten verschwendet werden.

Strategie	Bevorzugt stets das Paket, das...	Bekanntes & Referenz
NTS Nearest-To-Source	bisher die geringste Anzahl von Kanten bereits überquert hat	universell stabil (AAF$^+$01), 1-stabil (Ga03)
FFS Farthest-From-Source	die größte Anzahl von Kanten schon überquert hat	instabil für jedes $r > \frac{1}{\sqrt{2}}$ (AAF$^+$01)
NTG Nearest-To-Goal	die geringste Zahl von Kanten noch zu überqueren hat	instabil für beliebige $r > 0$ (BKR$^+$01)
FFG Farthest-From-Goal	die größte Anzahl von Kanten noch zu überqueren hat	universell stabil (AAF$^+$01), 1-stabil (Ga03)
FIFO First-In-First-Out	schon am längsten vor der betreffenden Kante wartet	instabil für alle $r > 0$ (BG03) selbst auf planaren Graphen (KMS03)
LIFO Last-In-First-Out	am wenigsten vor der fraglichen Kante wartet	instabil für jedes $r > \frac{1}{\sqrt{2}}$ (AAF$^+$01)
SIS Shortest-In-System	die geringste Zeit im System ist	universell stabil (AAF$^+$01)
LIS Longest-In-System	die längste Zeit im System ist	universell stabil (AAF$^+$01), instabil bei $r = 1$ (BKR$^+$01)

Tabelle 3.1: Prominente Queueing Strategien und ihre Stabilität

Definition 3.3 *Eine Queueing Strategie ist eine* greedy *Queueing Strategie, wenn zu jedem Zeitpunkt und für jede Kante e ein Paket Kante e überquert, falls mindestens ein Paket in der Queue von e wartet.*

Abgesehen von der Zahl der Pakete im System ist auch die Transportzeit eines Pakets, also die Zeitspanne zwischen Einfügung des Pakets durch den Gegner und Ankunft des Pakets am Zielknoten, von Interesse. Für den Fall $r < 1$ stellt das folgende Lemma eine Beziehung zwischen den Zielen *kleine Paketzahl im System* und *kurze Transportzeit* für greedy Queueing Strategien her und führt beide auf die Schlüsselgröße der Queuegröße zurück. Die Queuegröße einer Kante beschreibt dabei die Zahl der zur Überquerung der Kante bereitstehenden Pakete. Das folgende Lemma stellt die bekannten Beziehungen zwischen diesen Größen zusammen.

Lemma 3.1 *Sei $G = (V, E)$ ein Netzwerk mit längstem Weg l_{max}. Sei weiter S eine greedy Queueing Strategie. Dann sind folgende Aussagen für einen (r, b)-Gegner A mit $r < 1$ äquivalent.*

A1 Bei Anwendung von S auf G gegen A ist die Zahl der Pakete in G durch eine Schranke, die nur von $l_{max}, |E|, r$ und b abhängt, beschränkt.

A2 Bei Anwendung von S auf G gegen A ist die Transportzeit jedes Pakets in G durch eine Schranke, die nur von $l_{max}, |E|, r$ und b abhängt, beschränkt.

A3 Bei Anwendung von S auf G gegen A ist für jede Kante e die Zahl der Pakete, die zu einem Zeitpunkt t für die Überquerung von e bereitstehen – die Queuegröße von e zur Zeit t – durch eine Schranke, die nur von $l_{max}, |E|, r$ und b abhängt, beschränkt.

A4 Strategie S ist stabil gegen A.

Beweis: A1 und A4 sind nach Definition äquivalent. Wir vollziehen den Ringschluss $A1 \rightarrow A2 \rightarrow A3 \rightarrow A1$.

A1 → A2: Sei die Zahl der Pakete im System mit $c_{l_{max},|E|,r,b}$ beschränkt. Bezeichne W die Wartezeit, die ein Paket p auf die Überquerung einer Kante e warten muss. Steht p vor e bereit, darf aber e nicht überqueren, so überquert, da S greedy ist, ein anderes Paket e. Da die zugewiesenen Wege einfach sind, kann jedes andere Paket das Paket p nur einmal aufhalten. Nach Annahme befinden sich zu dem Zeitpunkt, an dem p erstmalig für Kante e bereitsteht, höchstens $c_{l_{max},|E|,r,b}$ Pakete im System. Diese Schranke plus die maximal $r \cdot W + b$ während der Wartezeit W eingefügten Pakete sind eine obere Schranke für W, was den Ansatz

$$W \leq c_{l_{max},|E|,r,b} + r \cdot W + b$$

und damit

$$W \leq \frac{c_{l_{max},|E|,r,b} + b}{1 - r}.$$

rechtfertigt. Ist die Wartezeit von p an jeder Kante derart begrenzt, so ergibt sich $l_{max} \cdot \frac{c_{l_{max},|E|,r,b}+b}{1-r}$ als obere Schranke für die Transportzeit von p.

A2 $\rightarrow$ **A3**: Sei die Transportzeit mit $c'_{l_{max},|E|,r,b}$ beschränkt. Nehmen wir nun an, dass zu einem Zeitpunkt mehr als $c'_{l_{max},|E|,r,b}$ Pakete vor einer Kante e warten. Dann kann das zuletzt bediente Paket keine Transportzeit von höchstens $c'_{l_{max},|E|,r,b}$ aufweisen, da allein die Wartezeit vor e und die Überquerung von e mehr Zeit in Anspruch nimmt. Also ist auch die maximale Queuegröße mit $c'_{l_{max},|E|,r,b}$ beschränkt.

A3 $\rightarrow$ **A1**: Gilt für jede Kante e, dass die Zahl der vor ihr wartenden Pakete höchstens $c''_{l_{max},|E|,r,b}$ ist, so ist die Gesamtzahl der Pakete im Graph höchstens $|E| \cdot c''_{l_{max},|E|,r,b}$. □

Damit können wir uns im Fall $r < 1$ auf die Analyse der Queuegröße beschränken. Allerdings impliziert Stabilität, also eine begrenzte Queuegröße, nicht die praktische Anwendbarkeit der Strategie. Denkt man etwa in Dimensionen des Internets, so ist eine Transportzeit, die exponentiell in der Zahl der Knoten ist, nicht akzeptabel. Für den praktischen Einsatz müssen Paketzahlen, Transportzeiten und Queuegrößen nicht nur beschränkt sondern auch durch eine hinreichend kleine Schranke beschränkt sein. Dass diese Schranken polynomiell in $\frac{1}{1-r}$ und der Größe von G ist, ist dabei eine Minimalanforderung. Als Kennziffer für die Größe von G dient dabei die Knotenzahl, der Durchmesser oder die Länge eines längsten einfachen Weges.

Beobachtung 2 *Der Ringschluss von Lemma 3.1 zeigt auch, dass, wenn eine der Größen Gesamtpaketzahl, maximale Zustellzeit und Queuegröße polynomiell in $\frac{1}{1-r}$ und der Größe des Graphen ist, dass dann alle Größen polynomiell beschränkt sind.*

In (AAF+01) wird gezeigt, dass die universell stabilen Strategien SIS, NTS und FTG jeweils zu exponentiellen Queuegrößen gezwungen werden können. LIS ist die einzige prominente Strategie, bei der die Frage nach der Queuegröße noch offen ist. In (AR02) wurde gezeigt, dass LIS auf gerichteten azyklischen Graphen eine Queuegröße, die linear in der Länge des längsten einfachen Weges des Graphen ist, nicht überschreitet. In (AZ04) wird gezeigt, dass ein Beweis für die Linearität der Queuegrößen unter LIS, so sie denn gilt, konzeptionell neue Ansätze enthalten müsste, da die exponentiellen unteren Schranken für SIS, NTS und FTG alle mit Methoden erzielt wurden, die für den Fall von LIS inhärent zu schwach sind.

Bereits in (AAF+01) wurde eine randomisierte Strategie vorgestellt, die mit hoher Wahrscheinlichkeit nur polynomielle Queuegröße erfordert. Hier

scheitert eine praxistaugliche Derandomisierung allerdings daran, dass jeder Router über den Zustand aller anderen Router informiert sein muss, um sein weiteres Vorgehen deterministisch zu berechnen. Dies steht im Widerspruch zur eingangs motivierten Forderung nach Lokalität. In (AFGZ01) ist es schließlich gelungen, eine dezentral derandomisierbare Strategie zu entwickeln, allerdings muss ein Router hier Berechnungen durchführen, die zu aufwendig sind für einen Prozess, der sich schnell und im Hintergrund abspielen sollte. Diese Strategie verletzt also die verlangte Einfachheit.

Die Frage, ob es eine einfache, *natürliche* Queueing Strategie gibt, die mit polynomieller Queuegröße auskommt, und insbesondere die Frage, ob LIS eine solche Strategie ist, ist vielleicht die wichtigste offene Frage im Kontext des Adversarial Queueing.

Wir lösen uns von der Untersuchung einzelner prominenter Strategien und untersuchen ganze Klassen von Strategien. Allgemein kann man sich eine Queueing Strategie durch eine Prioritätsfunktion repräsentiert denken, die das vorhandene Wissen über das Netzwerk und vorhandene Informationen über ein Paket auf eine Priorität des Pakets abbildet. Die Strategie gibt dann stets einem Paket mit maximaler Priorität den Vorzug. Klassen von Strategien entstehen dann natürlicherweise dadurch, dass man charakterisiert, welche Informationen die Prioritätsfunktion als Argument erhält. Wir betrachten Strategien ohne Zeitnahme. Zunächst soll diese Klasse formal eingeführt werden.

Definition 3.4 *Eine Queueing Strategie S_f arbeitet ohne Zeitnahme, wenn sie einem Paket p die Priorität $f(G, P, a)$ zuweist, wobei G der Graph des Netzwerks, P der Pfad von p und a die Zahl der bereits von p traversierten Kanten ist. Wir nennen Strategien, die ohne Zeitnahme arbeiten, WTS-Strategien (without time stamping.)*

Prominente WTS-Strategien sind Nearest-To-Source (NTS), Farthest-From-Source (FFS), Nearest-To-Goal (NTG), und Farthest-To-Goal (FTG) mit Prioritätsfunktionen

$$\begin{aligned} f_{NTS}(G,P,a) &= -a \\ f_{FFS}(G,P,a) &= a \\ f_{NTG}(G,P,a) &= a - |P| \\ f_{FTG}(G,P,a) &= |P| - a. \end{aligned}$$

Für diese Klasse können wir weitreichende Ergebnisse erzielen. Insbesondere zeigen wir im Abschnitt 3.2 (Theorem 3.1), dass jede WTS-Strategie zu einer Queuegröße von $2^{\Omega(\sqrt{n})}$ gezwungen werden kann, wobei n die Knotenzahl ist. Der Durchmesser d der herangezogenen Graphenfamilie ist dabei asymptotisch gleich $\sqrt{n}$, womit die Queuegröße $2^{\Omega(d)}$ beträgt. Auch der längste einfache Weg im Graphen hat Länge $\theta(\sqrt{n})$.

Dieses Resultat legt nahe, dass die Verwaltung von Zeiten notwendig ist, wenn man an subexponentiellen Queuegrößen interessiert ist, da die einzigen *sinnvollen* Größen, die in WTS-Strategien keinen Eingang finden, Zeiten sind, etwa die Lebenszeit eines Pakets – wie in LIS verwendet – oder die aktuelle Wartezeit eines Pakets – wie in FIFO verwendet.

Die Ergebnisse der Abschnitte 3.3 und 3.4 klären die Frage der universellen Stabilität und der 1-Stabilität für *distanzbasierte* Strategien, eine große Teilklasse der WTS-Strategien, der alle oben genannten prominenten Strategien angehören.

Für das komplette Kapitel treffen wir die folgenden Vereinbarungen, die denen aus (BKR$^+$01) entsprechen:

- G ist stets der Graph des betrachteten Netzwerks, $\mathcal{P}_G$ ist die Menge aller einfachen Pfade in G. Schließlich seien n, m, d, l_{max} die Zahl der Knoten, die Zahl der Kanten, der Durchmesser und die Länge eines längsten Weges im Graphen.

- Wir nehmen an, dass das Netzwerk in sich nicht überlappenden, aufeinanderfolgenden Schritten arbeitet und jeder Schritt in drei sich nicht überlappende, aufeinanderfolgende Teilschritte zerfällt:

  1. Einfügungen: Der Gegner fügt Pakete ein und weist den eingefügten Paketen Pfade zu, wobei er seine Restriktionen einhält.
  2. Transport: Pakete überqueren Kanten.
  3. Absorption: Pakete, die Ihren Zielort erreicht haben, werden aus dem System entfernt.

  Also kann ein Paket in dem Schritt, in dem es eingefügt wird, auch sofort seine erste Kante durchlaufen, falls es von der Queueing Strategie dafür ausgewählt wird.

- Wir benutzen $Q_e(t)$, um die Menge der Pakete zu bezeichnen, aus denen die Queueing Strategie im Schritt t dasjenige wählt, das Kante e überqueren darf. Genauer besteht $Q_e(t)$ also aus

  – den im Schritt t an der Quelle von e eingefügten Paketen, die e überqueren sollen,

  – den Paketen aus $Q_e(t-1)$ mit Ausnahme desjenigen, das e in Schritt $t-1$ überquert hat,

  – und Paketen, die in der Transportphase von Schritt $t-1$ die Quelle von e erreicht haben und über e weiter wandern möchten.

- Gelegentlich benutzen wir in unseren Konstruktionen Mehrfachkanten. Diese können jedoch bei Bedarf durch das Einfügen von Dummy-Knoten beseitigt werden.

- Wir nehmen an, dass immer, wenn mehr als ein Paket in einer Warteschlange maximale Priorität hat, dieses Patt schlimmstmöglich aus Sicht der Strategie aufgelöst wird.

3.2 Queuegrößen von WTS-Strategien

Wir zeigen nun, dass jede WTS-Strategie zu exponentieller Queuegröße gezwungen werden kann.

Theorem 3.1 *Es gibt eine Familie G_k von Graphen mit $n = 2k^2 + 6$ Knoten, Durchmesser $d = 4k - 4$ und längstem einfachen Weg der Länge $l_{max} = 4k$, auf der jede WTS-Strategie gegen jeden Gegner mit $r > 0.5$ und $b > \frac{2rk}{2r-1}$ zu einer Queuegröße von $2^{\Theta(k)}$ gezwungen werden kann. Damit stimmt die Queuegröße asymptotisch mit $2^{\Theta(d)}$, $2^{\Theta(l_{max})}$ und $2^{\Theta(\sqrt{n})}$ überein.*

Beweis: $G_k = (V_k, E_k)$ sei wie folgt gegeben: G_k besteht im Wesentlichen aus k^2 Kopien des Gatters G_{ij} (vergl. Abb. 3.1). G_k hat $2k^2+6$ Knoten und der Durchmesser beträgt $4k$.

Sei R_i die Menge der Pfade von X nach Z, die ausschließlich die Gatter der Zeile i durchlaufen. Sei C_j die Menge der Pfade von X' zu Z', die ausschließlich Gatter der j-ten Spalte durchlaufen. In unserer Konstruktion arbeiten wir ausschließlich mit diesen Zeilen- und Spaltenpfaden.

Für jede der x-Kanten x^l_{ij} bestimmen wir einen *dominanten Pfad*. Dazu halten wir zunächst fest, dass Paketen auf einem Pfad $P \in R_i$, der x^l_{ij} benutzt, in der Queue von Kante x^l_{ij} eine Priorität von $f(G_k, P, 2j)$ zugewiesen wird, da solche Pakete in der Queue von x^l_{ij} genau $2j$ Kanten bereits traversiert haben. Entsprechend wird allen Paketen auf einem Pfad $P \in C_j$, der x^l_{ij} benutzt, in der Queue von x^l_{ij} eine Priorität von $f(G_k, P, 2i)$ zugewiesen.

Wir nennen einen Pfad D dominant für $x^l_{i,j}$, wenn $D \in R_i \cup C_j$ gilt, D die Kante x^l_{ij} benutzt und Paketen, die entlang D unterwegs sind, in der Queue von x^l_{ij} die maximale Priorität, also

$$\max\left\{\max_{P\in R_i}\{f(G_k,P,2j)|P \text{ benutzt } x^l_{ij}\}, \max_{P\in C_j}\{f(G_k,P,2i)|P \text{ benutzt } x^l_{ij}\}\right\}$$

zugewiesen wird.

Falls ein dominanter Pfad für x^l_{ij} Element von R_i ist, so nennen wir die Kante x^l_{ij} *zeilendominiert*, andernfalls ist sie *spaltendominiert*. Darüberhinaus nennen wir Gatter $G_{i,j}$ zeilendominiert [spaltendominiert], wenn mindestens $k+1$ seiner x-Kanten zeilendominiert [spaltendominiert] sind. Also ist jedes Gatter entweder zeilen- oder spaltendominiert.

Wir konzentrieren uns im Weiteren auf eine Zeile, in der mindestens die Hälfte aller Gatter spaltendominiert sind, oder eine Spalte, in der mindestens die Hälfte aller Gatter zeilendominiert sind. Dass eine solche Zeile oder

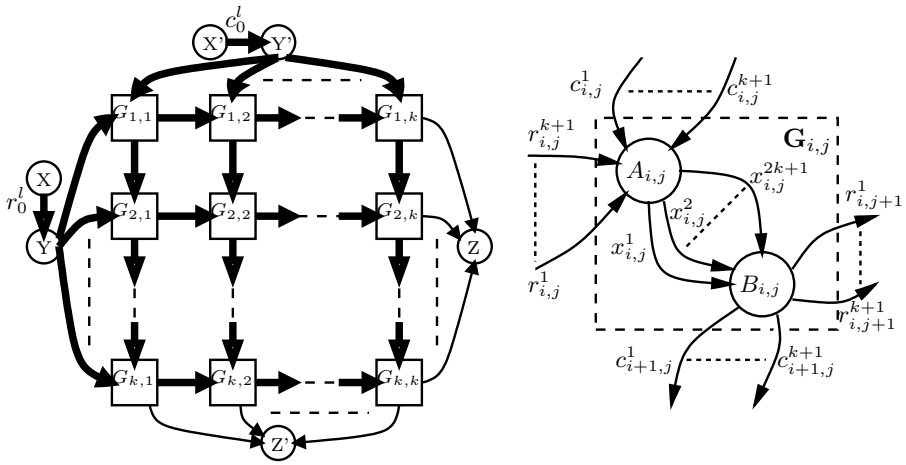


Abbildung 3.1: Graph G_k enthält k^2 Gatter $G_{i,j}$ sowie die zusätzlichen Knoten X, Y, Z, X', Y' und Z' angeordnet und verbunden wie links angedeutet. Ein dicker Pfeil steht hier für $k+1$ parallele Kanten in der jeweiligen Richtung. Jedes Gatter besteht aus zwei internen Knoten: $A_{i,j}$, in dem alle eingehenden Kanten enden, und $B_{i,j}$, dem alle ausgehenden Kanten entspringen. Diese Knoten sind durch $2k+1$ gerichtete Kanten, $x_{i,j}^1, \ldots, x_{i,j}^{2k+1}$ von $A_{i,j}$ nach $B_{i,j}$ verbunden. Die eingehenden Zeilenkanten, die entweder aus $G_{i,j-1}$ bzw. aus Y kommen, benennen wir mit $r_{i,j}^l$ für $1 \leq l \leq k+1$. Die eingehenden Spaltenkanten, die entweder aus $G_{i-1,j}$ bzw. Y' kommen, benennen wir mit $c_{i,j}^l$ für $1 \leq l \leq k+1$. Die ausgehenden Zeilenkanten [Spaltenkanten] der letzten Spalte $G_{i,k}$ [Zeile $G_{k,j}$] nach Z [Z'] heißen $r_{i,k+1}$ [$c_{k+1,j}$]. Die $k+1$ Kanten von X nach Y [X' nach Y'] schließlich nennen wir r_0^l [c_0^l] für $1 \leq l \leq k+1$.

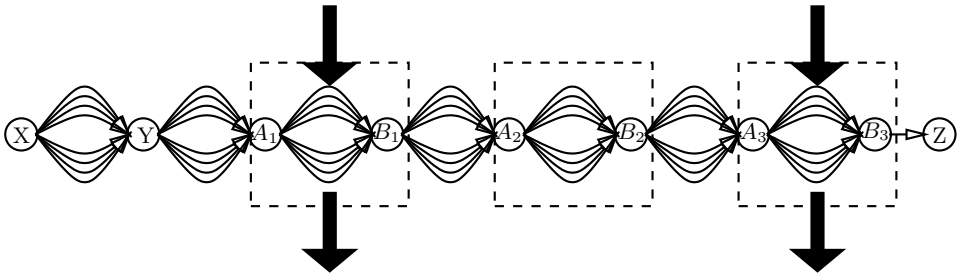


Abbildung 3.2: Die Situation in Zeile i_0 exemplarisch für $k = 3$. In den spaltendominierten Gattern kann durch vertikal kreuzende Pakete der Verkehr beliebig ausgebremst werden.

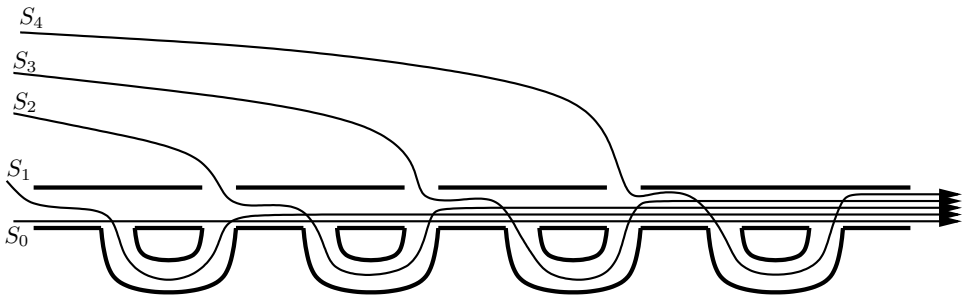


Abbildung 3.3: Das Verhältnis der zu konstruierenden Pfade relativ zueinander.

Spalte existiert folgt mit dem Schubfachprinzip daraus, dass entweder mindestens die Hälfte aller Gatter zeilendominiert oder mindestens die Hälfte aller Gatter spaltendominiert ist.

Sei o.B.d.A. Zeile i_0 eine solche Zeile. Dann existieren $q \geq \frac{k}{2}$ spaltendominierte Gatter in Zeile i_0. Ebenso o.B.d.A. nehmen wir an, dass in spaltendominierten Gattern $G_{i_0,j}$ die Kanten $x^1_{i_0,j}, \ldots, x^{k+1}_{i_0,j}$ spaltendominiert sind. Für $k = 3$ skizziert Abb. 3.2 die Situation.

Wir bestimmen als Nächstes eine Menge von Pfaden aus R_i, die wir später zum Einfügen von Paketen verwenden werden. Abb. 3.3 stellt das Ziel schematisch dar: Pakete, die auf verschiedenen Pfaden eingefügt werden, sollen zu einer großen Menge wartender Pakete zusammengeführt werden. Es existiert ein Hauptpfad, in den alle Pfade S_i an verschiedenen Punkten einmünden. Ein Pfad ist bis zu seiner Einmündung kantendisjunkt zu allen anderen Pfaden. Bei seiner Einmündung durchläuft er zunächst eine Kante des Hauptpfades mit den anderen Pfaden zusammen, hat danach eine Kante exklusiv für sich und verschmilzt dann endgültig mit den übrigen Pfaden. Unsere folgende Konstruktion stellt sicher, dass ein einmündender Pfad in

der ersten gemeinsamen Kante niedrigste Priorität hat. Die Wahl geeigneter Pfade leistet Algorithmus 3.1.

Man beachte, dass die Wahlen von e und e' in Algorithmus 3.1 jeweils wohldefiniert sind, da zu Beginn für jedes j mindestens $k+1$ legale $r^l_{i_0,j}$ und $x^l_{i_0,j}$ Kanten existieren. Höchstens $k-1$ dieser Kanten wurden bei vorangegangenen Schleifendurchläufen im Schritt 2(b)ii gelöscht, bevor e und e' zu wählen sind.

Wir können nun mit dem Einfügen von Paketen beginnen. Die Einfügungen vollziehen sich in q Phasen, die den spaltendominierten Gattern entsprechen. In Phase t kommen die eben konstruierten Pfade S_t und D_t zum Einsatz. Seien $G_{i_0,j_1}, G_{i_0,j_2}, \dots G_{i_0,j_q}$ die spaltendominierten Gatter der Zeile i_0. Um den Prozess in Gang zu setzen, nutzen wir einmalig die Burstiness aus und fügen in einem ersten Schritt b Pakete auf S_0 ins System ein. Diese Paketmenge nennen wir X_0.

Phase t: Phase t beginnt, wenn das erste Paket von X_{t-1} die Queue der Kante $r^l_{i_0,j_t}$ erreicht und dauert $|X_{t-1}|$ Schritte. In Phase t fügen wir jeweils mit Rate r Pakete in S_t und D_t ein. Man beachte, dass S_t und D_t kantendisjunkt sind und diese Einfügungen also legal sind: S_z und D_z haben lediglich das Gatter G_{i_0,j_t} gemein. S_z erreicht und verlässt es auf Zeilenkanten, D_z auf Spaltenkanten. In Schritt 2(b)iii wird sichergestellt, dass S_z und D_z verschiedene x-Kanten zur Durchquerung des Gatters benutzen.

In der Queue von $r^l_{i_0,j_t}$ treffen die Pakete aus S_t erstmalig auf die Pakete aus X_{t-1}. Ihre Priorität in dieser Queue ist nicht größer als die Priorität jedes Paketes der Menge X_{t-1}, da jedes Paket aus X_{t-1} auf einem der Pfade $S_0, \dots, S_{t-1}$ unterwegs ist, und da diese Pfade alle legal waren, als S_t unter den legalen Pfaden minimal gewählt wurde.

Also kann keines der $r|X_{t-1}|$ Pakete auf S_t das Gatter G_{i_0,j_t} in Phase t durchqueren, da die Eingangskante durchgehend von Paketen aus X_{t-1} benutzt wird.

Nach $2i_0$, also höchstens $2k$ Schritten der Phase t erreicht das erste auf D_t eingefügte Paket das Gatter G_{i_0,j_t}. Von diesem Zeitpunkt an bis zum Ende der Phase (das sind mindestens $|X_{t-1}|-2k$ Schritte) erreichen $r\cdot(|X_{t-1}|-2k)$ Pakete auf dem dominanten Pfad das Gatter G_{i_0,j_t}. Nach Definition des dominanten Pfades und nach Konstruktion sind diese Pakete an der Kante $x^l_{i_0,j_t}$ allen Paketen aus X_{t-1} überlegen. Beachte dazu, dass alle Pakete aus X_{t-1} auf Zeilenpfaden wandern. Also schaffen es mindestens $r\cdot(|X_{t-1}|-2k)$ Pakete aus X_{t-1} nicht, das Gatter G_{i_0,j_t} in Phase t zu durchqueren.

Sei X_t die Vereinigung dieser verbliebenen X_{t-1} Pakete und der neu auf S_t eingefügten Pakete. Dann ist

$$|X_t| \geq r(|X_{t-1}| - 2k) + rX_{t-1} = 2r|X_{t-1}| - 2rk.$$

Für $r > \frac{1}{2}$ und hinreichend großes b (i.e. $|X_0| = b > \frac{2rk}{2r-1}$) ist jede Menge X_t um einen multiplikativen Faktor größer als X_{t-1}.

Wahl der Pfade

1. Die Menge L enthalte die Kante $r_{i_0,k+1}$, die Kanten r^l_0 sowie $r^l_{i_0,j}$ und $x^l_{i_0,j}$ für $1 \leq j \leq k$ und $1 \leq l \leq k+1$. Wir nennen Kanten in L *legal* und nennen einen Pfad legal, wenn er nur legale Kanten verwendet. Sei $z := q$.

   Wir verwenden z um die spaltendominierten Gatter von q nach 1 hinunterzuzählen.

2. **Für** j **von** k **bis** 1 in absteigender Reihenfolge wiederhole:

   (a) Wähle $e \in \{r^l_{i_0,j} | 1 \leq l \leq k+1\} \cap L$ beliebig.
   Kante e ist also ein *legaler* Eingang in das Gatter $G_{i_0,j}$.

   (b) **Wenn** $G_{i_0,j}$ spaltendominiert ist:

      i. Sei S_z ein legaler Pfad, der e benutzt, und unter allen legalen Pfaden, die e benutzen, eine minimale Priorität in Q_e erhält. (I.e., $f(G_k, S_z, 2j-1)$ ist minimal).
      Kante e stellt die erste gemeinsame Kante des Pfades S_z mit den übrigen Pfaden dar.

      ii. Entferne die Kanten, die S_z vor e durchläuft, aus L.
      Dadurch wird sichergestellt, dass S_z bis zu seiner Einmündung kantendisjunkt zu allen noch zu wählenden Pfaden ist.

      iii. Wähle $e' \in (\{x^l_{i_0,j} | 1 \leq l \leq k+1\} \setminus S_z)$.
      Kante e' ist also eine legale Kante, die von S_z nicht benutzt wird.

      iv. Sei D_z der dominante Pfad von e'.
      Da e' spaltendominiert ist, existiert ein dominanter Pfad aus C_k.

      v. Setze $z := z - 1$.

   (c) **Sonst**: Wähle $e' \in \{x^l_{i_0,j} | 1 \leq l \leq k+1\} \cap L$ beliebig.
   In den zeilendominierten Gattern können wir keine Pfade einmünden lassen.

   (d) Entferne alle Kanten $r^l_{i_0,j} \neq e$ und $x^l_{i_0,j} \neq e'$ mit $1 \leq l \leq k+1$ aus L.

   Damit ist sichergestellt, dass alle im weiteren Verlauf gewählten Pfade über e und e' laufen. Der gemeinsame Suffix aller noch zu wählenden Pfade vergrößert sich also.

3. Wähle einen legalen Pfad S_0 beliebig.

Algorithmus 3.1: Die Wahl geeigneter Pfade.

Da $q = \Theta(k) = \Theta(d) = \Theta(l_{max})$ gilt, hat die letzte Menge X_q die Größe $2^{\Theta(k)} = 2^{\Theta(d)} = 2^{\Theta(l_{max})} = 2^{\Theta(\sqrt{n})}$, was zu zeigen war. □

3.3 Aufschichtungskreise

Aufschichtungskreise sind eine neue Technik zum Nachweis der 1-Stabilität von Queueing Strategien. Durch sie können Stabilitätsbeweise einzelner prominenter Strategien vereinheitlicht werden und ganze Klassen 1-stabiler Strategien identifiziert werden.

Auf intuitiver Ebene stellt sich die Argumentation wie folgt dar: Angenommen eine Strategie S ist auf einem Graphen G instabil bei Rate $r = 1$. Dann kann durch passende Einfügungen jede vorgegebene Zahl von Paketen ins System gebracht werden. Folglich muss es mindestens eine Kante e geben, deren Queue über jede vorgegebene Schranke anwachsen kann. Da außerdem die Zahl der einfachen Pfade, die eine bestimmte Kante enthalten, in jedem Graphen konstant ist, gibt es sogar einen einzelnen Pfad P, der e enthält, und entlang dem beliebig viele der vor e wartenden Pakete unterwegs sind.

Damit die Zahl der Pakete entlang eines Pfades aber groß wird, ist es nötig, dass für einen längeren Zeitraum mehr Pakete entlang des Pfades eingefügt werden als den Pfad am Zielknoten verlassen. Folglich müssen blockierende Pakete zur Wirkung kommen, die die Pakete auf P aufhalten. Da wir aber während der Blockaden weiter Pakete in P einfügen müssen, um die Paketzahl auf P zu steigern, können die blockierenden Pakete nicht gleichzeitig eingefügt werden, da sie ja mindestens eine Kante mit P gemein haben.

Ein *wesentlicher Teil* der nötigen blockierenden Pakete muss also schon im System vorhanden sein, wenn die Paketzahl auf P beginnt anzuwachsen. Wir kommen so zu weiteren Pfaden, die zu einem bestimmten früheren Zeitpunkt große Mengen von Paketen enthalten haben müssen. Da die Menge der einfachen Pfade in jedem Graphen endlich ist, muss sich in dieser Argumentation irgendwann ein Kreis schließen. Eine solche zyklische Anordnung von Pfaden nennen wir dann einen Aufschichtungskreis und können per Kontraposition auf 1-Stabilität schließen, wann immer eine Strategie S auf einem Graphen G keine Aufschichtungskreise zulässt.

Wesentlich ist der Begriff des Pfad-Präfixes.

Definition 3.5 *Sei $P = (e_1, e_2, \ldots, e_z)$ ein einfacher Pfad. Dann besteht der Pfad-Präfix $[P, a]$ für $a \leq z$ aus den Kanten $(e_1, e_2, \ldots, e_a)$.*

Sei $Q_e^P(t)$ die Untermenge der Pakete aus $Q_e(t)$, denen Pfad P zugewiesen ist, und sei $Q^P(t)$ die Menge aller Pakete, die zur Zeit t im System sind und denen ebenfalls Pfad P zugewiesen ist. (Also gilt $Q^P(t) = \bigcup_{i=1}^{z} Q_{e_i}^P(t)$.)

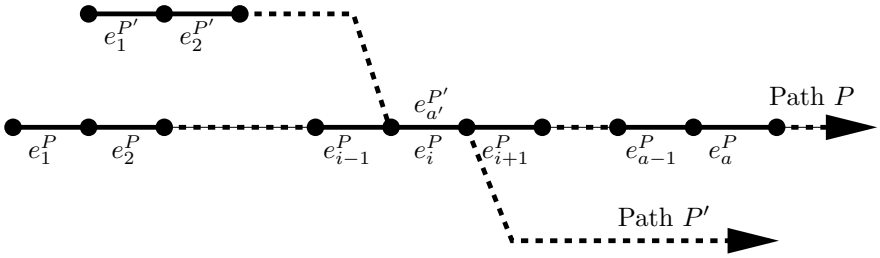


Abbildung 3.4: Veranschaulichung von Lemma 3.2. Die gemeinsame Kante ist $e_{a'}^{P'} = e_i^P$.

Weiter sei $Q^{[P,a]}(t) = \bigcup_{i=1}^{a} Q_{e_i}^P(t)$ die Menge der Pakete auf dem Pfad-Präfix $[P, a]$, die "auf P unterwegs sind".

Lemma 3.2 zeigt, dass es für jeden *schwer beladenen* Pfad-Präfix $[P, a]$ einen weiteren schwer beladenen Pfad-Präfix $[P', a']$ gibt, der in einer gemeinsamen Kante e Vorfahrt vor $[P, a]$ hat. Dieses Lemma werden wir dann, bei vorliegender Instabilität, iteriert anwenden, um auf die Existenz von Aufschichtungskreisen zu schließen. Abb. 3.4 illustriert die Aussage von Lemma 3.2.

Lemma 3.2 *S_f sei eine WTS-Strategie mit Prioritätsfunktion f, die auf einem Graphen G benutzt wird. Sei weiter angenommen, dass ein $(1, b)$-Gegner zum Zeitpunkt 1 beginnend bis zum Zeitpunkt t Einfügungen vornimmt und anschließend ein Pfad-Präfix $[P, a]$ existiert, so dass $|Q^{[P,a]}(t)| \geq c$ gilt, wobei c eine Konstante größer als b ist. Dann existiert ein Pfad-Präfix $[P', a']$ so dass,*

- *P' gegenüber P in einer gemeinsamen Kante $e = e_i^P = e_{a'}^{P'}$ Vorfahrt hat und*
- *der Pfad-Präfix von P', der mit e endet, zu einem Zeitpunkt vor t mindestens*

$$\frac{c}{2 \cdot 3^{l_{max}} \cdot |\mathcal{P}_\mathcal{G}|} - \frac{b}{|\mathcal{P}_\mathcal{G}|}$$

Pakete enthalten hat.

Beweis: Angenommen $|Q^{[P,a]}(t)| \geq c$ gilt. Dann wählen wir t_0 minimal, so dass es zur Zeit t_0 ein $i \leq a$ gibt, derart dass gilt

$$|Q_{e_i}^P(t_0)| > \frac{c}{3^{a-i+1}}. \tag{3.1}$$

Ein solches $t_0 \leq t$ existiert, da andernfalls $|Q_{e_j}^P(t)| \leq \frac{c}{3^{a-j+1}}$ für jedes

$j \leq a$ gelten würde und folglich

$$\begin{aligned} |Q^{[P,a]}(t)| &= \sum_{j=1}^{a} |Q_{e_j}^P(t)| \\ &\leq \sum_{j=1}^{a} \frac{c}{3^{a-j+1}} \\ &= c \cdot 3^{-a} \cdot \sum_{j=1}^{a} 3^{j-1} \\ &= c \cdot 3^{-a} \frac{3^a - 1}{2} \\ &< \frac{c}{2} \text{ Widerspruch.} \end{aligned}$$

Sei i der minimale Index, der Ungleichung (3.1) erfüllt. Nun wählen wir t_1 maximal mit $t_1 < t_0$, so dass $Q_{e_i}^P(t_1) = \emptyset$. Auch diese Wahl ist wohldefiniert, da alle Queues zur Zeit 0 leer sind. Die Minimalität von t_0 und $t_1 < t_0$ ausnutzend erhalten wir

$$\begin{aligned} |Q^{[P,i-1]}(t_1)| &= \sum_{j=1}^{i-1} |Q_{e_j}^P(t_1)| \\ &\leq \sum_{j=1}^{i-1} \frac{c}{3^{a-j+1}} \\ &\leq \frac{c}{2 \cdot 3^{a-i+1}}. \end{aligned}$$

Da $|Q_{e_i}^P(t_0)| > \frac{c}{3^{a-i+1}}$ gilt, waren also höchstens die Hälfte der Pakete in $Q_{e_i}^P(t_0)$ zur Zeit t_1 bereits im System.

Wir konzentrieren uns auf das Zeitintervall $J = (t_1, t_0]$ und nehmen an, dass y Pakete während J in P eingefügt werden. Sei x die Zahl der Pakete auf Pfad P, die e_i während J traversieren. Dann lässt sich folgern:

$$y - x \geq \frac{c}{2 \cdot 3^{a-i+1}} \tag{3.2}$$

folgern. Da $Q_{e_i}^P(t)$ auf Grund der Maximalität von t_1 während J nicht leer ist, überquert in jedem Schritt ein Paket die Kante e_i, das mindestens die Priorität $f(G, P, i-1)$ hat.

Nach Ungleichung 3.2 wächst das Paketaufkommen im Pfad-Präfix $[P, i-1]$ während J um den Beitrag $y - x \geq \frac{c}{2 \cdot 3^{a-i+1}}$. Die Gesamtbilanz, also die Differenz zwischen eingefügten und in e_i abgeflossenen Paketen während J ist höchstens die Burstiness b. Damit müssen während J mindestens $y - x - b$ Pakete mehr über von P verschiedene Wege, mit Vorfahrt über P in e_i, abgeflossen sein als Pakete über von P verschiedene Wege hinzugekommen sind.

Damit sind zum Beginn von Zeitintervall J, also zum Zeitpunkt t_1, mindestens $\frac{c}{2 \cdot 3^{l_{max}}} - b$ Pakete auf von P verschiedenen Pfaden, die Vorfahrt gegenüber P in e_i haben, im System. Diese Pakete befinden sich auf ihren jeweiligen Pfaden vor e_i.

Das Schubfachprinzip liefert schließlich, dass es einen Pfad P' mit mindestens gleicher Priorität wie P in e_i gibt, der zur Zeit t_1 mindestens

$$\frac{c}{2 \cdot 3^{l_{max}} \cdot |\mathcal{P}_G|} - \frac{b}{|\mathcal{P}_G|}$$

dieser Pakete vor e_i enthält.

Wählen wir abschließend noch a', so dass $e_{a'}^{P'} = e_i$ gilt, haben wir das Lemma gezeigt. □

Eine iterierte Anwendung des Lemmas 3.2 führt nun direkt zu den Aufschichtungskreisen.

Definition 3.6 (Aufschichtungskreis) *Ein Aufschichtungskreis für eine WTS-Strategie mit Prioritätsfunktion f in einem Graphen G ist eine Menge von Pfaden $P_1, P_2, \ldots, P_r \in \mathcal{P}_G$ in G mit jeweils zwei ausgezeichneten Kanten $e_{x_i}^{P_i}$ und $e_{y_i}^{P_i}$ in jedem Pfad und den folgenden Eigenschaften:*

- $\forall_{1 \leq i \leq r}\ x_i \leq y_i$*: Kante* $e_{x_i}^{P_i}$ *liegt in* P_i *vor* $e_{y_i}^{P_i}$*,*
- $\forall_{1 \leq i \leq r-1}\ e_{y_i}^{P_i} = e_{x_{i+1}}^{P_{i+1}}$ *und* $e_{y_r}^{P_r} = e_{x_1}^{P_1}$*: Die zweite ausgezeichnete Kante von P_i ist die erste ausgezeichnete Kante von P_{i+1}, und die zweite ausgezeichnete Kante von P_r ist die erste ausgezeichnete Kante von P_1.*
- $\forall_{1 \leq i \leq r-1}\ f(G, P_i, y_i) \geq f(G, P_{i+1}, x_{i+1}) \wedge f(G, P_r, y_r) \geq f(G, P_1, x_1)$*: Pfad P_i hat Vorfahrt gegenüber P_{i+1} an ihrer gemeinsamen, ausgezeichneten Kante. Pfad P_r hat Vorfahrt gegenüber P_1 an ihrer gemeinsamen ausgezeichneten Kante.*

Abb. 3.5 illustriert das Konzept eines Aufschichtungskreises. Wir können nun das Hauptresultat dieses Abschnitts formulieren.

Theorem 3.2 *Wenn eine WTS-Strategie S_f mit Prioritätsfunktion f instabil für $r = 1$ auf einem Graphen G ist, dann existiert unter den Pfaden von G ein Aufschichtungskreis für S_f.*

Beweis: Wir definieren die folgende Rekursion, um Lemma 3.2 hinreichend häufig anwenden zu können:

$$\begin{aligned} A(0) &= b \\ A(i) &= \frac{A(i+1)}{2 \cdot 3^{l_{max}} \cdot |\mathcal{P}_G|} - \frac{b}{|\mathcal{P}_G|}. \end{aligned}$$

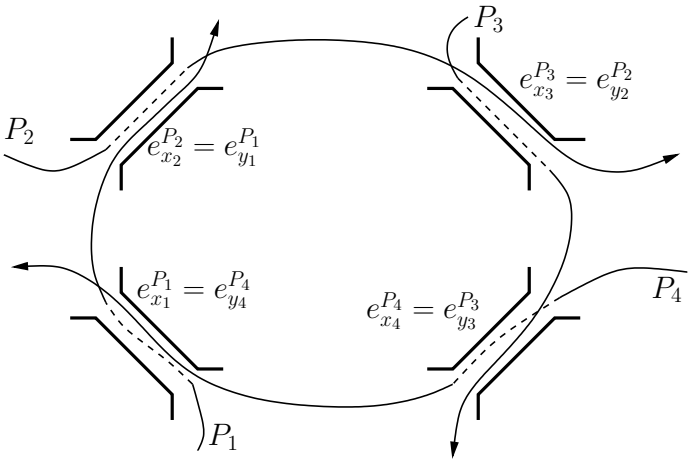


Abbildung 3.5: Schematische Darstellung eines Aufschichtungskreises bestehend aus vier Pfaden. Die vier ausgezeichneten Kanten sind als Kanäle dargestellt. Die Vorfahrt wird durch den Linientyp wiedergegeben: der Pfad mit der durchgehenden Linie hat gegenüber dem mit der gestrichelten Linie Vorfahrt.

Da wir Instabilität annehmen, existiert eine Folge von Einfügungen nach der G

$$|\mathcal{P}_G| \cdot A(d \cdot |\mathcal{P}_G|)$$

Pakete enthält. Also enthält mindestens ein Pfad P_0 in G nach der Folge von Einfügungen $A(l_{max} \cdot |\mathcal{P}_G|)$ Pakete. Wir wählen Pfad P_0 und seine Länge $a_0 := |P_0|$ als ersten Pfad-Präfix $[P_0, a_0]$.

Wir wenden Lemma 3.2 nun iterativ an: Wenn wir einen Pfad-Präfix $[P_i, a_i]$ haben, dem $A(l_{max} \cdot |\mathcal{P}_G| - i)$ Pakete aufgezwungen werden können, dann liefert Lemma 3.2 einen Pfad P_{i+1}, der $[P_i, a_i]$ schneidet und Vorfahrt an diesem Schnittpunkt hat. Wählen wir a_{i+1} gemäß Lemma 3.2, wissen wir, dass dem Pfad-Präfix $[P_{i+1}, a_{i+1}]$

$$\frac{A(l_{max} \cdot |\mathcal{P}_G| - i)}{2 \cdot 3^{l_{max}} \cdot |\mathcal{P}_G|} - \frac{b}{|\mathcal{P}_G|} = A(l_{max} \cdot |\mathcal{P}_G| - (i+1))$$

Pakete aufgezwungen werden können.

Wir können die Anwendung von Lemma 3.2 $l_{max} \cdot |\mathcal{P}_G|$ mal iterieren, was eine obere Schranke auf der Zahl der Pfadpräfixe in G ist. Also muss im Laufe der Anwendungen ein Kreis geschlossen werden und das Theorem folgt. □

Per Kontraposition folgt sofort, dass eine Strategie, die in keinem Graphen Aufschichtungskreise bildet, 1-stabil ist. Das folgende Korollar enthält Nearest-To-Source und Farthest-To-Goal als Spezialfälle.

Korollar 3.1 *Angenommen eine WTS-Strategie S_f mit Prioritätsfunktion f ist gegeben. Wenn f entlang jedes Pfades P in G streng monoton fallend ist, i.e.*

$$f(G, P, i) > f(G, P, i + 1),$$

dann ist S_f 1-stabil.

Beweis: Angenommen, G enthielte einen Aufschichtungskreis bezüglich S_f, seien dann $P_1, \ldots, P_r$, $x_1, \ldots, x_r$ und $y_1, \ldots, y_r$ entsprechend Definition 3.6 gewählt. Dann folgt mit

$$f(G, P_1, x_1) > f(G, P_1, y_1) \geq f(G, P_2, x_2) > \cdots > f(G, P_r, y_r) \geq f(G, P_1, x_1)$$

ein Widerspruch. Gemäß Theorem 3.2 ist S_f 1-stabil. □

Beobachtung 3 *Angenommen, eine WTS-Strategie, die nicht von der Zahl der bereits zurückgelegten Kanten abhängt, wird auf G eingesetzt. Die Strategie hängt also nur vom Pfad eines Pakets und vom Graphen ab. Wenn die Strategie verschiedene Prioritäten an verschiedene Pfade zuweist, dann ist die Strategie 1-stabil.*

Diese Klasse von Strategien ist zwar groß, aber die willkürliche Zuweisung von Prioritäten an Pfade kann kaum als sinnvoll gelten.

3.4 Distanzbasierte Strategien

Wir nennen eine Strategie distanzbasiert, wenn die zugrundeliegende Prioritätsfunktion nur von der Zahl x der bereits traversierten Kanten und der Länge y des Pfades eines Pakets abhängt. NTS,NTG,FTS und FTG sind distanzbasierte Strategien.

Für diese Teilklasse der WTS-Strategien gelingt eine vollständige Klassifizierung aller universell stabilen und 1-stabilen Strategien.

Theorem 3.3 *Sei f die Prioritätsfunktion einer distanzbasierten Queueing Strategie S_f. Dann ist S_f genau dann 1-stabil, wenn*

$$\forall (x, y)\ 1 \leq x < y\ :\ f(x, y) < f(x - 1, y). \tag{3.3}$$

Andernfalls ist S_f nicht universell stabil.

Beweis: Die 1-Stabilität gegeben (3.3) folgt sofort aus Korollar 3.1, da f entlang jeden Pfades streng monoton fallend ist.

Für die Rückrichtung, den Nachweis der Instabilität gegeben

$$\neg(3.3) : \exists x, y f(x, y) \geq f(x - 1, y),$$

betten wir den so genannten Baseball Graph aus (AAF$^+$01) in eine Umgebung aus Zubringer- und Abnehmerpfaden ein. Auch das Schema der Einfügungen ist etwas aufwendiger als das in (AAF$^+$01), folgt aber ähnlichen Ideen.

Der Beweis nutzt aus, dass S_f nur dann als stabil anzusehen ist, wenn es für jede mögliche Wahl von Paketen gleicher Priorität die Stabilität garantiert. Rivalisieren also zwei Pakete gleicher Priorität um eine Kante, so können wir das weiterzuleitende Paket passend für unsere Zwecke wählen.

Wir wählen ein x und ein y, durch die (3.3) verletzt wird. Wir arbeiten mit dem Graphen aus Abb. 3.6.

Wir nehmen an, dass es einen Zeitpunkt t gibt, so dass, wenn ab t keine neuen Pakete mehr eingefügt werden, in jedem der nächsten s Schritte ein Paket Kante e_0 traversiert. Sei S die Menge dieser Pakete und es sei $|S| = s$. Die Pakete in S haben vor e_0 genau x Kanten traversiert und sind auf einem Pfad der Länge y unterwegs, der in X_B endet.

Wir erstellen eine Folge von Paket Einfügungen (Phase 1,2 und 3), so dass, wenn nach Phase 3 keine neuen Pakete mehr eingefügt werden, noch für mehr als s Schritte Pakete die Kante e_0' überqueren, die auf einem Pfad der Länge y unterwegs sind, Knoten X_B' als Ziel haben, und genau x Kanten vor e_0' traversiert haben.

Daraus folgt dann die Instabilität, da der Prozess beliebig häufig wiederholt werden kann. Bevor wir die Einfügungen im Einzelnen darstellen, sei festgehalten, dass wir in allen drei Phasen nur solche Pakete verwenden, die über einen Zubringerpfad laufen, genau drei der *zentralen Knoten* (i.e. A, B, C, A', B' oder C') durchlaufen und über einen Abnehmerpfad zu ihrem Ziel laufen. Damit hat jedes Paket einen Gesamtpfad der Länge

$$(x-1) + 2 + (y-x-1) = y.$$

In ihrem ersten zentralen Knoten haben die Pakete jeweils $x-1$ Kanten bereits überquert, im zweiten zentralen Knoten sind es x viele.

Die folgende Tabelle gibt die nötigen Paketeinfügungen in kompakter Form wieder.

Phase	Dauer	Eingefügte Menge & Pfad	Größe
Eins	s	$F : (Y_A \xrightarrow{*} A \xrightarrow{e_0} B \xrightarrow{e_1} A' \xrightarrow{*} X_{A'})$	rs
Zwei	rs	$G_1 : (Y_A \xrightarrow{*} A \xrightarrow{e_0} B \xrightarrow{e_2} A' \xrightarrow{*} X_{A'})$	$r^2 s$
		$G_2 : (Y_B \xrightarrow{*} B \xrightarrow{e_1} A' \xrightarrow{e_0'} B' \xrightarrow{*} X_{B'})$	$r^2 s$
Drei	$r^2 s$	$H_1 : (Y_B \xrightarrow{*} B \xrightarrow{e_2} A \xrightarrow{e_0'} B' \xrightarrow{*} X_{B'})$	$r^3 s$
		$H_2 : (Y_C \xrightarrow{*} C \xrightarrow{e_4} B \xrightarrow{e_1} A' \xrightarrow{*} X_{A'})$	$r^3 s$

In Phase eins überquert nach Annahme in jedem Schritt ein Paket aus S die Kante e_0. Wenn die Pakete der Menge F den Knoten A erreichen, so

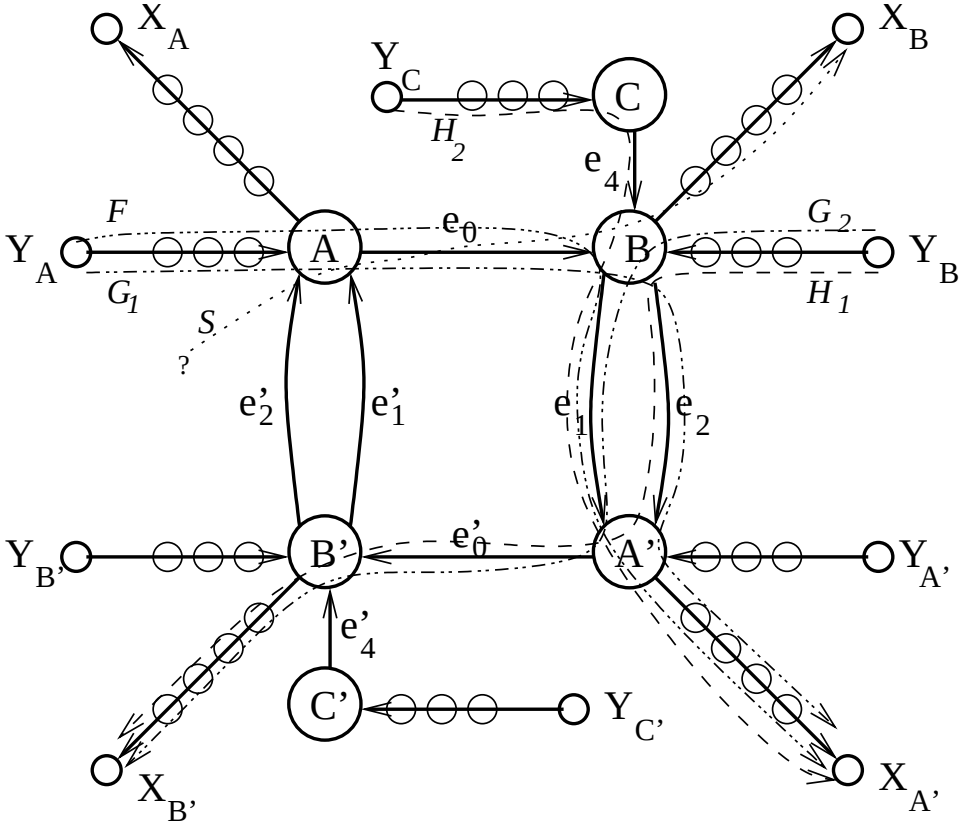


Abbildung 3.6: **Graph zum Nachweis der Instabilität von $\mathbf{S_f}$.** Es gibt *zentrale* Knoten A, B, C und A', B', C'. Jeder zentrale Knoten hat einen *Zubringerpfad*, der in einem Knoten $Y_A, \ldots, Y_{C'}$ beginnt. Jeder dieser Zubringerpfade besteht aus $x - 1$ Knoten und $x - 1$ Kanten. (Für $x = 1$ verschwinden die Pfade also und der Y-Knoten ist mit dem zugehörigen zentralen Knoten identisch). A, B, A' und B' haben außerdem einen *Abnehmerpfad*, der in einem Knoten $X_A, \ldots, X_{B'}$ endet. Jeder Abnehmerpfad hat Länge $y - x - 1$. Für $y = x + 1$ verschwinden diese Pfade und zentraler Knoten und zugehöriger X-Knoten fallen zusammen.

sind sie dort den S-Paketen nach Priorität unterlegen, da sie bis dahin $x-1$ Kanten überquert haben, während Pakete der Menge S nach Annahme x Kanten überquert haben, und $f(x,y) \geq f(x-1,y)$ nach Wahl von x und y gilt. Also überquert kein F Paket in Phase 1 die Kante e_0.

Nach Ablauf der Phase eins haben die s Pakete der Menge S die Kante e_0 überquert und sind entweder bereits in X_B angekommen oder auf dem Abnehmerpfad von Knoten B zu X_B, wo sie aber für unsere weiteren Betrachtungen keine Rolle mehr spielen.

In Phase zwei überqueren die F-Pakete die Kante e_0. Wir gehen davon aus, dass die Menge S und damit auch F so groß ist, dass im Strom der F-Pakete über e_0 keine Lücken entstehen: Man beachte, dass das letzte F-Paket zu Beginn der Phase zwei Knoten A noch nicht erreicht hat, sondern erst im letzten Schritt von Phase eins in den Zubringerpfad eingefügt worden ist. Wir gehen also davon aus, dass sich in A genügend F-Pakete aufgestaut haben, so dass das letzte F-Paket A erreicht bevor dieser Vorrat von Paketen in A abgeflossen ist.

Die Paketmenge G_1 wird in den Zubringerpfad von A eingefügt und trifft in A auf die angestauten Pakete aus F. Da beide Paketmengen in A genau $x-1$ Kanten überquert haben, kann sie die Queueing Strategie nicht unterscheiden und wir können festlegen, dass sich die G_1-Pakete hinter den F-Paketen einreihen, also kein Paket aus G_1 die Kante e_0 in Phase zwei überquert.

Die Menge G_2 wird gleichzeitig mit G_1 eingefügt, was legal ist, da die zugewiesenen Pfade beider Mengen kantendisjunkt sind. Die Pakete der Menge G_2 treffen in B auf die Pakete der Menge F und konkurrieren mit diesen um die Kante e_1. Bis zum Knoten B haben die F-Pakete jedoch x Kanten überquert, während die G_2-Pakete nur auf $x-1$ überquerte Kanten kommen. Der Strom aus F-Paketen setzt sich also ungehindert über e_1 fort, und die G_2 Pakete stauen sich in B vor e_1.

Nach Phase zwei hat das letzte Paket aus F die Kante e_1 überquert. Die F-Pakete sind entweder bereits in $X_{A'}$ absorbiert worden oder auf dem Abnehmerpfad von A', wo sie für uns keine Bedeutung mehr haben.

Die in Phase drei eingefügten Paketmengen H_1 und H_2 haben kantendisjunkte zugewiesene Pfade, können also parallel eingefügt werden. Verfolgen wir zunächst die Mengen G_1 und H_1. Die Pakete aus G_1 können nun Kante e_0 überqueren. Wir nehmen wieder an, dass die Menge so groß ist, dass die Nachzügler, die zu Beginn der Phase noch auf dem Zubringerpfad sind, dank des Staus in A aufschließen können und der Strom ab Kante e_0 lückenlos ist. In B angekommen setzen sich die G_1-Pakete gegen die H_1-Pakete durch, die ebenfalls über e_2 wollen. In B haben die G_1-Pakete bereits x Kanten überquert, die H_1-Pakete dagegen nur $x-1$. Also stauen sich die H_1-Pakete in B und kein H_1-Paket überquert e_2 in Phase drei.

Für G_2 und H_2 ist die Lage etwas unübersichtlicher. In den ersten x Schritten der Phase drei können x der in B wartenden G_2-Pakete Kante

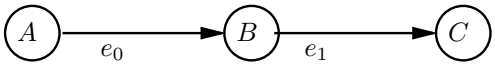


Abbildung 3.7: Graph zur Existenz gestrandeter Pakete bei NTS.

e_1 überqueren und im weiteren Verlauf der Phase drei ihren Zielort $X_{B'}$ erreichen. Nach x Schritten jedoch erreicht das erste H_2-Paket Knoten B. Die Pakete aus H_2 sind den G_2-Paketen in B überlegen, da sie bereits x Kanten überquert haben. Die H_2-Pakete laufen also völlig ungebremst durch den Graphen. Allerdings weist dieser Strom von Paketen für Raten $r < 1$ Löcher auf. Kante e_1 ist in $r \cdot (r^2 s - x)$ der $r^2 s - x$ verbleibenden Schritte der Phase 3 durch H_2-Pakete besetzt. Folglich können nur in $(1 - r) \cdot (r^2 s - x)$ Schritten Pakete aus G_2 die Kante e_1 überqueren. Die Zahl der G_2-Pakete, die am Ende der Phase drei Kante e_1 noch nicht überquert haben, ist also mindestens

$$r^2 s - x - (1 - r) \cdot (r^2 s - x) = r(r^2 s - x).$$

Diese restlichen Pakete aus G_2 und die komplette Menge H_1 werden, wenn keine weiteren Einfügungen folgen, nach dem Ende von Phase 3 Kante e'_0 überqueren und zu Knoten $X_{B'}$ wandern. Die Größe der Vereinigung dieser Mengen ist mindestens

$$r(r^2 s - x) + r^3 s = 2r^3 s - rx.$$

Für $r > \frac{1}{2}$ und eine hinreichend große Burstiness, mit der wir eine genügend große erste Menge S erzeugen können, hat sich die Zahl der Pakete um einen multiplikativen Faktor erhöht.

Da die obige Sequenz von Einfügungen beliebig häufig wiederholt werden kann, kann die Zahl der Pakete im System über jede Schranke getrieben werden. Instabilität folgt. □

3.5 Gestrandete Pakete

In (Ga03) wurde erstmals angemerkt, dass bei der kritischen Rate $r = 1$ die Stabilität keine beschränkte Transportzeit mehr impliziert (vergl. im Gegensatz dazu Lemma 3.1). Nicht einmal, dass jedes Paket in endlicher Zeit sein Ziel erreicht, wird durch 1-Stabilität sichergestellt. Ein einfaches Beispiel bietet die Strategie Nearest-To-Source (NTS) auf dem Graphen aus Abb. 3.7.

Man füge zum Zeitpunkt 1 ein Paket p mit Ziel C in A ein. Noch im selben Schritt erreicht es B. Fügt man nun ab Schritt 2 ununterbrochen Pakete mit Ziel C in B ein, so ist p in B gestrandet. In (RT04) ist der Begriff *eternal Packets* für solche Pakete eingeführt worden. Dort wird auch gezeigt,

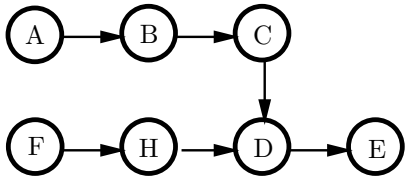


Abbildung 3.8: Der Graph G an dem gezeigt wird, dass jede WTS-Strategie bei $r = 1$ gestrandete Pakete zulässt.

dass keine Queueing Strategie 1-stabil sein kann und gleichzeitig gestrandete Pakete verhindern kann. Daraus folgt wiederum, dass es Queueing Strategien mit begrenzter Transportzeit für $r = 1$ nicht gibt.

Die Strategie Longest-In-System (LIS) ist universell stabil und verhindert, wie man sich rasch überlegt, das Stranden von Paketen. In diesem Sinne ist sie also bestmöglich. Es erscheint vernünftig anzunehmen, dass die Fähigkeit einer Strategie das Stranden von Paketen zu verhindern ein Indikator dafür ist, ob die Strategie die Queuegrößen klein halten kann.

Wir zeigen nun abschließend, dass die komplette Klasse der WTS-Strategien nicht in der Lage ist, bei $r = 1$ das Stranden von Paketen zu verhindern, selbst wenn man mit Burstiness 0 arbeitet und auf Worst-Case-Tiebreaking verzichtet.

Lemma 3.3 *Es gibt einen Graphen G auf dem jede WTS-Strategie gestrandete Pakete selbst bei Burstiness 0 zulässt.*

Beweis: Wir verwenden Graph G aus Abb. 3.8 und unterscheiden drei Fälle.

Fall 1: $\mathbf{f}(\mathbf{G}, (\mathbf{ABCD}), \mathbf{2}) \leq \mathbf{f}(\mathbf{G}, (\mathbf{CDE}), \mathbf{0})$.
Im ersten Schritt füge ein Paket p mit Pfad $(ABCD)$ ein. Es erreicht noch im ersten Schritt Knoten B und im zweiten Knoten C. Vom dritten Schritt an füge ununterbrochen Pakete in den Pfad (CDE) ein. Nach Fallannahme haben diese Pakete Vorfahrt und p ist in C gestrandet.

Fall 2: $\mathbf{f}(\mathbf{G}, (\mathbf{FHDE}), \mathbf{2}) \leq \mathbf{f}(\mathbf{G}, (\mathbf{CDE}), \mathbf{1})$.
Im ersten Schritt füge ein Paket p mit Pfad $(FHDE)$ ein. Es erreicht noch im ersten Schritt Knoten H. Von Schritt zwei an füge ununterbrochen Pakete in den Pfad (CDE) ein. Diese Pakete konkurrieren ab Schritt drei mit p um die Kante von D nach E. Nach Fallannahme haben sie Vorfahrt und p ist gestrandet.

Fall 3: Weder Fall 1 noch Fall 2.
Wir beschreiben eine Folge von Einfügungen von Paketen $a_1, a_2, p, b_1, b_2, \ldots$ und nutzen $f(G, (ABCD), 2) > f(G, (CDE), 0)$ und $f(G, (FHDE), 2) > f(G, (CDE), 1)$ aus. Paket p wird wieder das gestrandete sein.

Schritt	Einfügung (Pfad)	Bewegungen
1	a_1 : (ABCD)	a_1 erreicht B
2	a_2 : (ABCD)	a_2 erreicht B a_1 erreicht C
3	p : (CDE)	a_1 blockiert p in C und erreicht in D a_2 erreicht C
4	b_1 : (FHDE)	a_2 blockiert p in C und erreicht in D b_1 erreicht H
5	b_2 : (FHDE)	b_1 erreicht D p erreicht D b_2 erreicht H
6	b_3 : (FHDE)	b_1 blockiert p in D und erreicht in E b_2 erreicht D b_3 erreicht H
7	b_4 : (FHDE)	b_2 blockiert p in D und erreicht in E b_3 erreicht D b_4 erreicht H
$\vdots$	$\vdots$	$\vdots$ p ist gestrandet

Paket p hat in den Schritten 3 und 4 das Nachsehen, da Fall 1 nicht eintritt. Ab Schritt 6 benutzen wir, dass Fall 2 nicht vorliegt. Pro Schritt wird nur ein Paket eingefügt, damit ist die Rate 1 und die Burstiness 0. □

3.6 Zusammenfassung

In diesem Abschnitt haben wir die Klasse der Queueing-Strategien ohne Zeitnahme eingeführt und analysiert. Anders als bei der individuellen Analyse prominenter Queueing Strategien, ging es uns darum, unter Verwendung möglichst schwacher Annahmen über die Strategie möglichst starke Aussagen zu erzielen.

Die maximale Queuegröße einer Strategie ist eine Schlüsselgröße, und im Hauptresultat wurde bewiesen, dass sämtliche WTS-Strategien exponentielle Queuegrößen in Kauf nehmen müssen.

Das eingeführte Konzept der Aufschichtungskreise hat sich als nützlich zur Beurteilung der von Stabilität von WTS-Strategien erwiesen. Mit weiteren Argumenten ist es gelungen, eine komplette Klassifizierung distanzbasierter WTS-Strategien im Hinblick auf deren $1-$-Stabilität und universeller Stabilität vorzunehmen.

Abschließend wurde festgestellt, dass keine WTS-Strategien bei Rate $r = 1$ das Stranden von Paketen verhindern kann.

Ein Fazit der Ergebnisse dieses Abschnitts ist die Beobachtung, dass man auf direktem Weg wieder zu der Frage nach der Performance von Longest-In-System (LIS) geführt wird. Die Verwendung von Zeitstempeln scheint auf

Grund der Ergebnisse dieses Kapitels inhärent notwendig zu sein, um eine polynomielle Queuegrößen zu garantieren, und sicherlich ist LIS eine der einfachsten Strategien, die Zeitstempel benutzen.

Eine weitere Schlussfolgerung könnte sein, dass der Begriff der Stabilität als alleiniges Qualitätsmaß von Queueing-Strategien mit Fragezeichen zu versehen ist. Relativ obskure Strategien (vergleiche insbesondere Beobachtung 3) sind sogar 1-stabil, während sowohl die mit polynomiellen Queuegrößen auftrumpfende randomisierte Strategie aus (AAF$^+$01) und die einzige bekannte deterministische Strategie mit erwiesener polynomieller Transportzeit (AFGZ01) für $r = 1$ nicht definiert sind. LIS ist für $r = 1$ instabil, für $r < 1$ jedoch die einzige einfache Strategie, für die sogar polynomielle Transportzeit nicht ausgeschlossen ist.

Eine Eigenschaft von Queueing-Strategien, die neben dem Grad der Stabilität auch beachtet werden sollte, ist die Frage der Nachprüfbarkeit. Hier hat das nach wie vor populäre FIFO der Strategie LIS gegenüber einen entscheidenden Vorteil. FIFO basiert allein auf lokalen Informationen in dem Sinne, dass dem Paket keine Informationen angehängt werden, die Einfluß auf die Geschwindigkeit des Paketes im weiteren Verlauf haben. Ein Sender, der das Tempo seiner Pakete zu Lasten anderer erhöhen möchte, würde bei LIS unentdeckt bleiben, wenn er auf jedes seiner Pakete eine zu alte Einfügezeit einträgt.

Dasselbe Argument kann auch gegen die randomisierte Strategie aus (AAF$^+$01) und die derandomisierte Strategie in (AFGZ01) geltend gemacht werden. Im ersten Fall ist jeder Sender angehalten für die im Laufe einer Phase eingefügten Pakete eine gleichverteilte Anfangspriorität auszuwürfeln. Ein Konzept, das nur bei allseitig unterstellter Ehrlichkeit die gewünschte Performance nach sich zieht. Im zweiten Fall werden die Anfangsprioritäten aufwendig berechnet, aber auch hier müssen nachfolgende Router glauben, was die Sender als Priorität auf dem Paket vermerken.

Zusammenfassend können die Resultate über WTS-Strategien (deren Navigationsdaten wie Sender und Empfänger ja nicht ohne weiteres gefälscht werden können) auch als ein Schritt in Richtung eines Ergebnisses betrachtet werden, das besagt, dass man sich im Adversarial Queueing zwischen Warten und Vertrauen entscheiden muss. Ein solches Resultat sollte alle Strategien umfassen, die neben den Parametern der WTS-Strategien auch Zugriff auf lokale Zeiten (also aktuelle Wartezeit) und lokale Zufallsbits haben. Gelänge es auch für diese größere Klasse subexponentielle Queuegrößen auszuschließen, so wäre man der oben angedeuteten Notwendigkeit von Vertrauen oder Warten schon sehr nahe.

Literaturverzeichnis

[AAF+01] Andrews, M., Awerbuch, B., Fernández, A., Leighton, F. T., Liu, Z., und Kleinberg, J. M.: Universal-stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM.* 48(1):39–69. 2001.

[ACG+99] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., und Protasi, M.: *Complexity and Approximation.* Springer. 1999.

[AFGZ01] Andrews, M., Fernández, A., Goel, A., und Zhang, L.: Source routing and scheduling in packet networks. In: *42nd Annual Symposium on Foundations of Computer Science (FOCS).* S. 168–177. IEEE Computer Society. 2001.

[AR02] Adler, M. und Rosén, A.: Tight bounds for the performance of longest-in-system on dags. In: *19th Annual Symposium on Theoretical Aspects of Computer Science (STACS).* Bd. 2285 d. *Lecture Notes in Computer Science.* S. 88–99. Springer. 2002.

[AS95] Armen, C. und Stein, C.: Improved length bounds for the shortest superstring problem (extended abstract). In: *4th Workshop on Algorithms and Data Structures (WADS).* Bd. 955 d. *Lecture Notes in Computer Science.* S. 494–505. Springer. 1995.

[AS98] Armen, C. und Stein, C.: A 2 2/3 superstring approximation algorithm. *Discrete Applied Mathematics.* 88(1-3):29–57. 1998.

[AZ04] Andrews, M. und Zhang, L.: The effects of temporary sessions on network performance. *SIAM Journal on Computing.* 33(3):659–673. 2004.

[BEY98] Borodin, A. und El-Yaniv, R.: *Online Computation and Competitive Analysis.* Cambridge University Press. 1998.

[BG03] Bhattacharjee, R. und Goel, A.: Instability of fifo at arbitrarily low rates in the adversarial queueing model. In: *45th Annual*

Symposium on Foundations of Computer Science (FOCS). S. 160–167. IEEE Computer Society. 2003.

[BJJ97] Breslauer, D., Jiang, T., und Jiang, Z.: Rotations of periodic strings and short superstrings. *Journal of Algorithms*. 24(2):340–353. 1997.

[BKR+01] Borodin, A., Kleinberg, J. M., Raghavan, P., Sudan, M., und Williamson, D. P.: Adversarial queuing theory. *Journal of the ACM*. 48(1):13–38. 2001.

[BLTY94] Blum, A., Li, M., Tromp, J., und Yannakakis, M.: Linear approximation of shortest superstrings. *Journal of the ACM*. 41(4):630–647. 1994.

[BOR04] Borodin, A., Ostrovsky, R., und Rabani, Y.: Stability preserving transformations: Packet routing networks with edge capacities and speeds. *Journal of Interconnection Networks*. 5(1):1–12. 2004.

[CGPR97] Czumaj, A., Gasieniec, L., Piotrów, M., und Rytter, W.: Sequential and parallel approximation of shortest superstrings. *Journal of Algorithms*. 23(1):74–100. 1997.

[CLR90] Cormen, T. H., Leiserson, C. E., und Rivest, R. L.: *Introduction to Algorithms*. The MIT Engineering and Computer Science Series. MIT Press. 1990.

[Ga82] Gallant, J.: *String Compression Algorithms*. PhD thesis. Princeton University. 1982.

[Ga03] Gamarnik, D.: Stability of adaptive and nonadaptive packet routing policies in adversarial queueing networks. *SIAM Journal on Computing*. 32(2):371–385. 2003.

[GJ79] Garey, M. R. und Johnson, D. S.: *Computers and Intractability*. Freeman. 1979.

[GMS80] Gallant, J., Maier, D., und Storer, J. A.: On finding minimal length superstrings. *Journal of Computer and System Science*. 20(1):50–58. 1980.

[Gu97] Gusfield, D.: *Algorithms On Strings, Trees, and Sequences*. Cambridge University Press. 1997.

[Ho97] Hochbaum, D. S. (Hrsg.): *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company. 1997.

[Ka91] Karloff, H.: *Linear Programming*. Progress in Theoretical Computer Science. Birkhäuser. 1991.

[KMS03] Koukopoulos, D., Mavronicolas, M., und Spirakis, P.: Fifo is unstable at arbitrarily low rates (even in planar networks). *Electronic Colloquium on Computational Complexity*. April 2003.

[KPS94] Kosaraju, S. R., Park, J. K., und Stein, C.: Long tours and short superstrings (extended abstract). In: *35th Annual Symposium on Foundations of Computer Science (FOCS)*. S. 166–177. IEEE Computer Society. 1994.

[KS05] Kaplan, H. und Shafrir, N.: The greedy algorithm for shortest superstrings. *Information Processing Letters*. 93(1):13–17. 2005.

[La04] Laube, U.: SINDBAD (Dokumentation und Quellcode). intern. 2004.

[LW04] Laube, U. und Weinard, M.: Conditional inequalities and the shortest common superstring problem. In: *Prague Stringology Conference (PSC)*. S. 124–138. Vydavatelství, ČVUT. 2004.

[LW05] Laube, U. und Weinard, M.: Conditional inequalities and the shortest common superstring problem. *International Journal of Foundations of Computer Science*. 16(6). Dezember 2005.

[Mo81] Monge, G.: *Mémoire sur la théorie des déblais et des remblais*. S. 666–704. l'Academie Royale des Sciences. 1781.

[MS78] Maier, D. und Storer, J. A.: A note on the complexity of the superstring problem. In: *12th Annual Conference on Information Sciences and Systems (CISS)*. S. 52–56. 1978.

[Pa78] Paul, W. J.: *Komplexitätstheorie*. Teubner Verlag. 1978.

[RT04] Rosén, A. und Tsirkin, M. S.: On delivery times in packet networks under adversarial traffic. In: *16th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. S. 1–10. ACM. 2004.

[Sw99] Sweedyk, Z.: A $2\frac{1}{2}$-approximation algorithm for shortest superstring. *SIAM Journal on Computing*. 29(3):954–986. 1999.

[TU88] Tarhio, J. und Ukkonen, E.: A greedy approximation algorithm for constructing shortest common superstrings. *Theoretical Computer Science*. 57:131–145. 1988.

[TY97] Teng, S.-H. und Yao, F. F.: Approximating shortest superstrings. *SIAM Journal on Computing*. 26(2):410–417. 1997.

[Va01] Vazirani, V. V.: *Approximation Algorithms*. Springer Verlag. 2001.

[We05] Weinard, M.: The necessity of timekeeping in adversarial queueing. In: *4th International Workshop on Experimental and Efficient Algorithms (WEA)*. Bd. 3503 d. *Lecture Notes in Computer Science*. S. 440–451. Springer. 2005.

[WS03] Weinard, M. und Schnitger, G.: On the greedy superstring conjecture. In: *23rd Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*. Bd. 2914 d. *Lecture Notes in Computer Science*. S. 387–398. Springer. 2003.

[WS06] Weinard, M. und Schnitger, G.: On the greedy superstring conjecture (ext. version). *SIAM J. on Discrete Mathematics*. 2006.

Lebenslauf

Persönliche Daten

- Name : Maik Weinard
- Familienstand : ledig
- Geburtsdatum : 11.11.1972
- Nationalität : deutsch

Schulausbildung

- 1979 - 1983 Grundschulbesuch an der Saalburgschule in Bad Vilbel
- 1983 - 1985 Besuch der Förderstufe an der John F. Kennedy Schule Bad Vilbel
- 1985 - 1992 Besuch des Georg Büchner Gymnasiums in Bad Vilbel
  - Abschluß : Abitur (27.5.1992)
  - Note : 1,8
  - Leistungskurse : Mathematik und Physik
  - Fremdsprachen : englisch (9 Jahre), französisch (5 Jahre)

Wehrdienst: 1992 - 1993
Ableistung des Grundwehrdienstes; zuletzt am Fernmeldesektor H / Fermelderegiment 72 in Feuchtwangen (Bayern)

Studium: Oktober 1993 - Januar 2001
Studium im Diplomstudiengang der Informatik an der Johann Wolfgang Goethe-Universität in Frankfurt am Main

- Vordiplom
  - Abschluß : Vordiplom (19.3.1996)
  - Gesamtnote : sehr gut
  - Einzelprüfungen

        * Praktische Informatik : 1,0
        * Theoretische Informatik : 1,0
        * Mathematik A : 1,3
        * Mathematik B : 1,0
        * Nebenfach : 1,0
    - Nebenfach : Mathematik (Logik und Stochastik)

- Hauptstudium
    - Prüfungen
        * Praktisch/Technische Informatik : 1,3
          Entwurfsmethoden und Generierung von Testmustern
        * Vertiefungsfach : 1,7
          Effiziente Algorithmen und Kommunikationskomplexität
        * Nebenfach : 1,0
          Höhere Stochastik und Statistik
        * Theoretische Informatik : 1,0
          Analyseverfahren (I & II)
    - Diplomarbeit: „Komplexität der Schaltkreisoptimierung“: 1,0

Derzeitige Tätigkeit:
Seit Februar 2001 als Wissenschaftlicher Mitarbeiter angestellt am Lehrstuhl *Theoretische Informatik* bei Prof. Dr. G. Schnitger im Institut für Informatik der Johann Wolfgang Goethe-Universität Frankfurt am Main.

Erfahrungen in der Lehre

- Okt. 1996 - Jul. 2000: Tutorentätigkeit – Betreuung von Übungsgruppen zu den Veranstaltungen
    - Informatik 3; WS 96/97,
    - Informatik 4; SS 97,
    - Theoretische Informatik 1; WS 97/98, WS 98/99, WS 99/00,
    - Theoretische Informatik 2; SS 98,
    - Technische Informatik 2; SS 99,
    - Effiziente Algorithmen (Hauptstudium); SS 2000,
- seit Apr. 2001: Organisation und Betreuung von Übungsbetrieben als Wissenschaftlicher Mitarbeiter. Die Veranstaltungen:
    - Algorithmisches Lernen; SS 01
    - Theoretische Informatik 1; WS 01/02
    - Theoretische Informatik 2; SS 02

- Approximationsalgorithmen; WS 02/03
- Effiziente Algorithmen; SS 03
- Internet Algorithmen; WS 03/04, SS 05
- Komplexittstheorie; SS 04
- Parallele und Verteilte Algorithmen; WS 04/05, WS 05/06

- Parallel dazu: Betreuungen im Rahmen von Seminaren, Proseminaren und Diplomarbeiten.

Forschung

- Publikationen
  - mit Georg Schnitger: On the Greedy Superstring Conjecture – Foundations of Software Technology and Theoretical Computer Science (FSTTCS)), Dez. 2003
  - mit Uli Laube: Conditional Inequalities and the Shortest Common Superstring Problem – The Prague Stringology Conference (PSC), Aug. 2004
  - The Necessity of Timekeeping in Adversarial Queueing – Workshop on Efficient and Experiemental Algorithms (WEA), Mai 2005
  - mit Uli Laube: Conditional Inequalities and the Shortest Common Superstring Problem (Journal Version) – Journal on the Foundations of Computer Science (IJFCS),Dez. 2005
  - mit Georg Schnitger: On the Greedy Superstring Conjecture (ext. edition) – SIAM Journal on Discrete Mathematics, 2006 (accepted)

- Vorträge
  - 15.12.2003 - On the Greedy Superstring Conjecture (FSTTCS, Indian Institute of Technology, Mumbai, Indien)
  - 15/16.3.2004 - Das Studium der Informatik (Tage der Naturwissenschaften, Uni-Frankfurt)
  - 18.6.2004 - Kürzeste Superstrings (Schlusslehrgang der deutschen Endausscheidung zur internationalen Olympiade der Informatik (IOI), Arnoldshain)
  - 17.11.2004 - Adversarial Queueing Theorie (AG-Treffen)
  - 20.4.2005 - Die Grenzen von Queueing Strategien ohne Zeitnahme (AG-Treffen)
  - 13.5.2005 - Necessity of Timekeeping in Adversarial Queueing (Santorini, Griechenland)

- 13.7.2005 - Analyse von Heuristiken (Doktorandenkollquium des Instituts für Informatik)
- 29.11.2005 - FIFO-Stabilität von Netzwerken in polynomieller Zeit entscheiden (AG-Treffen)
- 16.1.2006 - Analyse von Heuristiken (Disputation)

Erfahrungen in der Selbstverwaltung (Bis Nov. 2000 ehrenamtlich als studentisches Mitglied, danach im Rahmen der WiMi-Tätigkeit)

- Fachbereichsrat bzw. Direktorium des Fachbereichs bzw. Instituts für Informatik
- Prüfungsausschuss
- Berufungskommissionen
- diverse Arbeitskreise