Ökonomische Analyse von Infrastrukturentscheidungen

Tim Stockheim
Institut für Wirtschaftsinformatik
Johann Wolfgang Goethe-Universität
Frankfurt am Main
stockheim@wiwi.uni-frankfurt.de

14. Juli 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Gegenstand der Untersuchung

In den letzten Jahrzehnten hat der Begriff des Netzwerkes bzw. der Netzwerk-Organisation zunehmend Eingang in die wissenschaftliche Literatur gefunden. Der Begriff des Netzwerkes hat sich als Bezeichnung für eine Koordinationsform, welche sich zwischen den Ausprägungen Markt für dezentralisierte Entscheidungsfindung und Hierarchie für zentrale Entscheidungsbefugnis befindet, etabliert[1].

Was unterscheidet nun das Netzwerk von einem Markt? In einem Markt, sei er nun zentral oder dezentral, sind die Beziehungen zwischen den Akteuren temporärer Natur, d.h. die Kosten für das Beenden oder den Neuaufbau einer Geschäftsbeziehung sind nicht die bestimmenden Größen. Kommt es zu Effekten, die die Wahlmöglichkeiten eines Marktteilnehmers stark einschränken, so kann die Funktion des Marktes erheblich gestört werden. Eine solche Störung wird auch als Marktversagen bezeichnet.

In Netzwerken bzw. Netzwerk-Organisationen kommt es zu festen Beziehungen, z.B. zur Spezialisierung mehrerer Partner, um ein Produkt herzustellen. Trotzdem, und hier findet die Abgrenzung zur Hierarchie statt, bleiben die Teilnehmer unabhängig in ihren Entscheidungen. Eine Übersicht und Strukturierung verschiedener Ausprägungen des Begriffes Netzwerk-Organisation findet sich in [Alstyne 1997], wobei abgegrenzt wird zwischen technischen, ökonomischen und sozialen Netzwerk-Organisationen, welche in sehr verschiedener Weise den Begriff des Netzwerkes adaptieren. Eine kurze Übersicht über die Ausprägungen der drei Arten von Netzwerken gibt die folgende Tabelle:

Was sind die Gemeinsamkeiten dieser Netzwerke? Die Lösung größerer komplexer Aufgaben ist in den meisten Fällen von Gruppen schneller und besser zu bewerkstelligen, als sie durch einen einzigen Prozessor, Akteur oder ein Individuum erreicht werden kann. In technischen Netzwerken hat sich beispielsweise der Forschungsbereich der Verteilten Künstlichen Intelligenz dem optimalen Aufbau und der Gestaltung der Beziehung von Gruppen von Akteuren bzw. Agenten angenommen. In ökonomischen Netzwerken und sozialen Forschungsbereichen werden ebenso große Anstrengungen unternommen, um solche Effekte zu erfassen und zu erklären. Beispielhaft seien hier aus der ökonomischen Perspektive die Theorie der Netzwerkexternalitäten und Stan-

[1] vgl. [Powell 1990]

technisch	ökonomisch	sozial
☐ Prozessabläufe und Parallelisierung	☐ Risikoverteilung und Informationsasymmetrie	☐ Granularität
☐ Spezialisierung oder Generalisierung	☐ Egoismus und Öffentliche Güter	☐ Identität und Integration
☐ Ausfallsicherheit	☐ Mechanismus Design	☐ Politik und Macht
☐ Kommunikations- und Koordinationskosten	☐ Transaktionskosten vs. Integration	☐ Vertrauen und Treue
☐ Verteilte Informationen bzw. Wissen	☐ Ressourcennutzung und Eigentumsrechte	☐ Rollen und Beschränkungen
☐ Modularität bzw. Kapselung	☐ Wertschöpfung und Nutzung	☐ Umgebung bzw. Umwelt

Tabelle 1.1: Netzwerktypen und Merkmale

dardisierung[2], sowie aus dem sozialen Kontext Gruppentheorien, Strukturtheorien und Theorien über Identität und Kontrolle genannt.

In einer weniger abstrakten Sichtweise spielen in all diesen Netzwerken Trade-Off Beziehungen zwischen Stabilität und Flexibilität eine Rolle. Deshalb ist es wichtig, zu erkennen, welche Faktoren bei der Etablierung von Netzwerken eine Rolle spielen, und welchen Einfluß diese dann auf die Struktur des Netzwerkes haben, bzw. wie stabil diese dann in Bezug auf sich ändernde Umweltbedingungen sind. Für eine solche Herangehensweise bedarf es die Realität möglichst gut abbildender Modelle[3], welche in dieser Arbeit um ein weiteres ergänzt werden sollen.

Unter dem technischen Gesichtspunkt erleben wir eine Zeit, welche durch massive und gezielte Investitionen in Infrastruktur geprägt ist[4]. Die richtige Investitionsentscheidung ist hier ein entscheidender Erfolgsfaktor für die Wirtschaft ganzer Gebiete. Die traditionellen Gleichgewichtsmodelle der Raumplanung wurden unter der Prämisse unbegrenzter Teilbarkeit der Ausprägung von Aktionsparametern aufgestellt[5] und sind den Bedürfnissen moderner Infrastrukturplanung nur eingeschränkt angepasst.

[2] vgl. [Farrell 1985] und [Buxmann 1996]

[3] Ein Modell, welches sich mit dem entstehen von Preisgleichgewichten in verteilten Märkten unter Berücksichtigung von Transportkosten beschäftigt, stammt von [Wellman 1993]. Ebenso wie eine Implementierung namens WALRAS in Common Lisp und dem Common Lisp Object System.

[4] Neben den alltäglichen Meldungen über den Ausbau der Kommunikationsnetzwerke insb. des Internets und der Mobilfunknetze, wird auch im Bereich der Wirtschaftsförderung zunehmend auf Infrastrukturausbau als Maßnahme der Förderung zurückgegriffen.

[5] vgl. [Wegener 1994] und [Lundqvist 1998]

1.2 Gang der Untersuchungen

Erkenntnismethode ist das Experiment. Ziel ist nicht hochspezialisierte Lösungsmethoden zu finden, sondern die Analyse des dezentralen Entscheidungsmodells. Getestet wird gegen verschiedene Parameterausprägungen, wie beispielsweise Größe, Transportmenge und Verteilung.

Zunächst wird in Kapitel 2 ein Modell vorgestellt, welches unter bestimmten Annahmen optimale Stukturen von Netzwerken ermitteln kann. Das kognitive Modell dieser Netzwerke kann sich sowohl im technischen, im ökonomischen als auch im sozialen Kontext wiederfinden, wobei bei letzterem eine geeignete Modellierung und Parametererhebung sehr fraglich erscheint. In Kapitel 3 werden geeignete Maße zur Analyse von Netzwerken vorgestellt, Kapitel 4 stellt die Lösungsansätze und Ergebnisse vor. Kapitel 5 gibt einen kurzen Einblick in die verwendete Software. Abgeschlossen wird die Diplomarbeit mit einer Zusammenfassung der Ergebnisse in Kapitel 6.

Kapitel 2

Problemformulierung

2.1 Transportproblem

2.1.1 Formale Beschreibung - das primale Problem

Das klassische Transportproblem beschreibt die Flüsse eines einzigen Produktes oder Rohstoffes von Anbietern zu Märkten. Bei der formalen Beschreibung bezeichnet der Index i einen Anbieter und der Index j einen Markt. Der Wert x_{ij} gibt die Menge an, welche von Anbieter i zu Markt j zu Kosten von c_{ij} pro Einheit transportiert wird. Das Gut wird als homogen und nicht beliebig teilbar in Bezug auf alle Anbieter und Nachfrager angenommen, wobei die kleinste Menge eine Einheit darstellt. Damit ergeben sich die Kosten von $c_{ij}x_{ij}$ für den Transport von i nach j. Die Menge des Gutes, welches Anbieter i liefern kann, wird mit S_i bezeichnet, die von Markt j benötigte Menge als D_j. Sollte die Angebotsmenge größer sein als die Nachfragemenge, so kann durch Einführung eines Dummy-Marktes, zu welchem alle Anbieter zum Preis Null liefern können, das Überangebot aufgenommen werden, ohne die Zielfunktion zu verändern. Sollte die Nachfrage höher sein, und es nur darum gehen, das vorhandene Angebot kostengünstigst zu verteilen, so lässt sich dieses durch Einführung eines Dummy-Anbieters erreichen.

Folgendes lineares Gleichungssystem beschreibt das Problem der Kostenminimierung für eine vorgegebene Transportmenge :

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}, \tag{2.1}$$

unter den Nebenbedingungen

$$\sum_{j=1}^{m} x_{ij} \leq S_i, \quad \forall_{\,0<i\leq n}, \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} \geq D_j, \quad \forall_{\,0<j\leq m}, \tag{2.3}$$

und die Nichtnegativitätsbedingungen (NNB)

$$x_{ij} \geq 0, \quad \forall_{\,0<i\leq n}; \quad \forall_{\,0<j\leq m}. \tag{2.4}$$

Die Zielfunktion wird in Gleichung (2.1) angegeben, Restriktion (2.2) begrenzt für jeden Anbieter die Angebotsmenge. Restriktion (2.3) legt die Menge des Gutes fest, welches jeder Markt mindestens erhalten muss. Schließlich sorgt die Nichtnegativitätsbedingung in Ungleichung (2.4) dafür, dass keine Lösungen mit negativen Transportmengen ermittelt werden.

Aufgrund der Unteilbarkeit der einzelnen Einheiten des zu transportierenden Gutes gilt für die Aktionsvariablen die Ganzzahligkeitsbedingung, also $x_{ij} \in \mathbb{N}$. Auf diese Bedingung kann jedoch verzichtet werden, da die Lösungen eines solchen linearen Programms immer ganzzahlig sind. Für jedes primale Problem existiert auch ein duales Problem, welches benötigt wird, um den Transportalgorithmus zu erklären.

2.1.2 Das duale Problem

Das duale Gleichungssystem ergibt sich wie folgt:

$$max \quad Z' = \sum_{j=1}^{m} D_j v_j \sum_{i=1}^{n} S_i u_i, \tag{2.5}$$

unter den Nebenbedingungen

$$v_j - u_i \leq c_{ij}, \quad \forall\, {}_{0<i\leq n}; \quad \forall\, {}_{0<j\leq m}, \tag{2.6}$$

und die NNB

$$u_i, v_j \leq 0, \quad \forall\, {}_{0<i\leq n}; \quad \forall\, {}_{0<j\leq m}. \tag{2.7}$$

Wobei u_i und v_j Hilfsvariablen sind, deren ökonomische Interpretation in Abschnitt 2.1.3 erläutert wird. Zwischen dem primalen und einem dualen Gleichungssystem existiert folgender Zusammenhang:

- Wenn das primale Problem ein Ressourcenproblem ist, dann ist das duale Problem immer ein Wertproblem.
- Ist das primale Problem ein Minimierungsproblem, so ist das duale Problem ein Maximierungsproblem und vice versa.
- Die duale Zielfunktion hat eine Variable für jede Restriktion des primalen Problems.
- Es gibt für jede Variable der primalen Zielfunktion eine Restriktion im dualen Gleichungssystem.
- Die Koeffizienten (c_{ij}) in der primalen Zielfunktion werden zu Konstanten in den Restriktionen des dualen Gleichungssystems; die Konstanten (S_i, D_j) der primalen Restriktionen werden zu Koeffizienten der dualen Zielfunktion.
- Das Relationssymbol der Ungleichungen im dualen Gleichungssystem (2.6) entspricht dem inversen Relationssymbol der Ungleichungen im primalen Gleichungssystem (2.2) und (2.3). Das Relationssymbol für die Nichtnegativitätsbedingungen bleibt erhalten. Die primalen Bedingungen in Ungleichung (2.3) stellt eine kleine Schwierigkeit dar, welche jedoch durch Multiplikation mit dem Faktor -1 und damit Vereinheitlichung des Relationssymbols gelöst wird.

2.1.3 Anwendung des dualen Problems

Zunächst einmal kann das duale Problem mittels linearer Programmierung für sich gelöst werden. Die Optimallösung, welche sich für das duale Problem ergibt, ist auch die Optimallösung des primalen Problems. Aus dem Tableau des primalen und des dualen Problems kann zudem der sog. Transportalgorithmus abgeleitet werden, welcher ein sehr effizientes Verfahren zur Lösung von Transportproblemen darstellt.

Die Variablen u_i und v_j werden in der ökonomischen Literatur als Schattenpreise der Angebotskapazität in i bzw. der Nachfragemenge in j interpretiert; das bedeutet, u_i ist der Wert des Gutes am Angebotsort und v_i der Wert am Nachfrageort. Die duale Zielfunktion versucht den Wertzuwachs des Systems als ganzes zu maximieren, wobei zur selben Zeit die Restriktionen dafür sorgen, dass die Wertzunahme nicht die Transportkosten c_{ij} übersteigen kann. Haggett (vgl. [Haggett 1977] Seite 495) zeigt, dass v_j als Nachfragepreis im Gleichgewicht am Knoten j interpretiert werden kann, wobei sich die Angebotspreise u_i dann in Verbindung mit den Transportkosten $u_i = v_j - c_{ij}$ ergeben. Existiert an jedem Knoten ein einheitlicher Herstellungspreis und wird davon ausgegangen, dass an einem Nachfrageknoten für ein Produkt immer derselbe Preis bezahlt wird, so ergeben sich für alle zuliefernden Knoten außer dem entferntesten Zulieferer eine Marge aufgrund ihrer Lage. So kann über die Schattenpreise auch die Vorteilhaftigkeit einer bestimmten geographischen Region bestimmt werden.

2.2 Das Beckmann-Marschak Problem

Das mit dem Transportproblem beschriebene ökonomische Problem ist in seiner Struktur sehr einfach. Ein realistischeres Szenario könnte beispielsweise einen Produktionsprozeß bzw. eine Transformation beinhalten. Also:

a) Eine Menge von n Produktionsstätten liefert ein Vorprodukt zu einer Menge von Transformationszentren. Die Erstellungskosten an der Produktionsstätte i werden mit e_i bezeichnet.

b) Die Einheiten x_{ij}, welche von Produktionsstätte i zum Transformationszentrum j transportiert werden, verursachen Transportkosten in Höhe von c_{ij} pro Einheit.

c) Es gibt t Transformationszentren, wobei am j-ten Transformationszentrum Prozeßkosten in Höhe von jeweils e_j^* pro Einheit anfallen.

d) Zum k-ten Markt werden x_{jk}^* Einheiten des Produktes von Transformationszentrum j zu Transportkosten in Höhe von c_{jk}^* geliefert. Es gibt m Märkte.

Das lineare Programm zum Beckmann-Marschak Problem:

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{t}(e_i + c_{ij})x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{t} e_j^* x_{ij} + \sum_{j=1}^{t}\sum_{k=1}^{m} c_{jk}^* x_{jk}^*, \tag{2.8}$$

unter den Nebenbedingungen

$$\sum_{j=1}^{t} x_{ij} \leq S_i, \quad \forall\, {}_{0<i\leq n}, \tag{2.9}$$

$$\sum_{j=1}^{t} x_{jk}^{*} \geq D_k, \quad \forall_{\,0<k\leq m}, \tag{2.10}$$

$$a_j \sum_{i=1}^{n} x_{ij} - \sum_{k=1}^{m} x_{jk}^{*} \geq 0, \quad \forall_{\,0<j\leq t}, \tag{2.11}$$

$$\sum_{i=1}^{n} x_{ij} \leq K_j, \quad \forall_{\,0<j\leq t}, \tag{2.12}$$

und die NNB

$$x_{ij} \geq 0, \quad \forall_{\,0<i\leq n}, \quad \forall_{\,0<j\leq t}, \tag{2.13}$$

$$x_{jk}^{*} \geq 0, \quad \forall_{\,0<j\leq t}, \quad \forall_{\,0<k\leq m}. \tag{2.14}$$

In den Ungleichungen (2.9) bis (2.12) werden das Angebot S_i und die Nachfrage D_k angegeben, wobei jedoch die Nachfrage nicht, wie beim Transportproblem, direkt von den Produktionsstätten befriedigt wird, sondern über das Transformationszentrum. Zusätzlich gibt K_j die Kapazität und a_j den Input-Output Koeffizient am Transformationszentrum j an. Der Input-Output Koeffizient gibt die Menge Output an, welche pro Menge Input am jeweiligen Transformationszentrum erzeugt werden kann.

Die einzelnen Elemente der Zielfunktion geben von links nach rechts die Erstellungskosten des Vorproduktes, die Transportkosten zu den Transformationszentren, die Transformationskosten und die Transportkosten zu den Märkten an. Ungleichung (2.9) und (2.10) sind Beschränkungen für Angebot und Nachfrage analog zum Transportproblem. Relation (2.11) sichert, dass die Menge Output, welche ein Transformationszentrum verlässt, nicht größer ist als die Menge, die dort mit dem angelieferten Input erzeugt werden kann. Die Transformation wird zudem durch die Kapazitätsbeschränkung (2.12) eingeschränkt.

2.2.1 Das Transshipment-Problem

Die Transformationszentren des Beckmann-Marschak Problems werden im Transshipment-Problem als reine Umschlagpunkte behandelt. Damit ist das Gleichungssystem des Transshipment-Problems als Spezialfall des Beckmann-Marschak Problems darstellbar, wobei der mittlere Teil der Zielfunktion $(\sum_{i=1}^{n}\sum_{j=1}^{t} e_j^{*} x_{ij})$ und Restriktion (2.12) entfällt, sowie in Ungleichung (2.11) der Koeffizient a_j gleich 1 ist. Die Möglichkeit der Umleitung von Transportgütern wird im weiteren noch näher betrachtet.

Zur Lösung von Transshipment-Problemen ohne Beschränkung der Transportkapazitäten bzw. der Umschlagmengen wird in der Operations-Research-Literatur eine Lösung mittels Algorithmen zur Berechnung des kürzesten Pfades vorgeschlagen. Insbesondere der Tripelalgorithmus, welcher simultan alle kürzesten Pfade eines Netzwerkes ermittelt, ist der Abbildung in ein lineares Gleichungssystem und dessen Lösung mittels des Simplex-Algorithmus deutlich überlegen.

2.2.2 Der Tripelalgorithmus

Der Tripelalgorithmus ermittelt die kürzeste Entfernung von jedem Knoten eines Netzwerkes zu jedem anderen Knoten eines Netzwerkes, mit dem erweiterten Tripel-

algorithmus können darüber hinaus auch die kürzesten Wege ermittelt werden. Durch Anwendung des Enumerationskriteriums der dynamischen Optimierung wird der Algorithmus, obwohl er eine implizite Enumeration darstellt, numerisch sehr effizient.

Ablauf

Für das hier abgebildete Netzwerk werden nachfolgend zur Illustration des Tripelalgorithmus die kürzesten Wege berechnet.

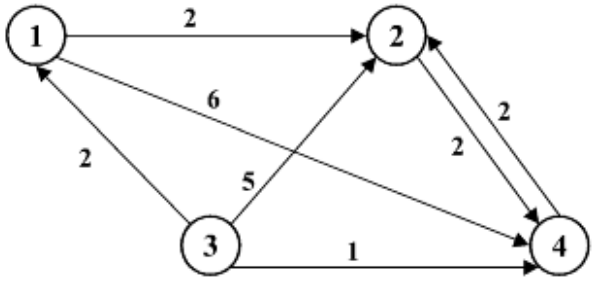

Abbildung 2.1: Einfacher 4-Knoten-Graph

Zur Ermittlung aller kürzesten Pfade in einem n-Knoten Netzwerk wird zunächst eine nxn Matrix erstellt, bei welcher zweckmäßigerweise die Zeilen und Spalten von 1 beginnend aufsteigend nummeriert werden. In diese Matrix werden für sämtliche Kanten des Netzwerkes die Entfernungen[1] vom Knoten i zum Knoten j in das Kreuzungsfeld der i-ten Zeile mit der j-ten Spalte eingetragen. Nachdem in alle Felder der Hauptdiagonalen der Wert Null eingesetzt wurde, wird in alle Felder, die noch nicht besetzt sind, der Wert ∞ eingetragen.

		nach			
	No	1	2	3	4
von	1	0	2	∞	6
	2	∞	0	∞	2
	3	2	5	0	1
	4	∞	2	∞	0

		nach			
	No	1	2	3	4
von	1	**0**	**2**	∞	**6**
	2	∞	0	∞	2
	3	**2**	4	0	1
	4	∞	2	∞	0

Tabelle 2.1: Initialisierungsmatrix und Matrix nach der 1. Iteration

Damit ist die Initialisierung der Matrix abgeschlossen, die Laufvariable (i) wird auf 0 gesetzt, und es kann mit der ersten Iteration begonnen werden:

Iterationsschritt 1: Es wird eine neue leere nxn-Matrix aufgestellt, in welcher die Werte der i-ten Spalte und der i-ten Zeile der vorhergehenden Matrix übernommen werden.

Iterationsschritt 2: Ist die Matrix vollständig gefüllt, erhöhe i um 1 und fahre fort mit **Iterationsschritt 1**.

[1] Im Falle der Transportkostenoptimierung werden statt der Entfernungen die Transportkosten auf den Kanten eingetragen.

Iterationsschritt 3: Ein noch freies Feld wird vorübergehend als Operationsfeld betrachtet. Der Index seiner Zeile sei z, der Index seiner Spalte sei s. Ermittle aus der vorhergehenden Matrix, deren Felder vorübergehend mit a_{ij} bezeichnet werden: $X = a_{zs}$ und $Y = a_{zi} + a_{is}$

Iterationsschritt 4: Trage das Minimum aus X und Y in das Operationsfeld der neuen Matrix ein, und fahre fort mit **Iterationsschritt 2**.

Ende: Die letzte Matrix enthält die kürzesten Wege für jede Verbindung.

		nach			
	No	1	2	3	4
von	1	0	2	∞	6
	2	∞	0	∞	2
	3	2	5	0	1
	4	∞	2	∞	0

		nach			
	No	1	2	3	4
von	1	**0**	**2**	∞	**6**
	2	∞	0	∞	2
	3	**2**	4	0	1
	4	∞	2	∞	0

Tabelle 2.2: Matrix nach der 2. Iteration und Optimaltableau

Steht in einem Feld der i-ten Zeile und der j-ten Spalte der Wert ∞, so existiert weder eine direkte noch eine indirekte Verbindung von Knoten i nach Knoten j. Sollten negative Kanten erlaubt sein, kann über die Felder auf der Hauptdiagonalen überprüft werden, ob es negative Zyklen gibt. Sobald in einem dieser Felder ein negativer Wert auftritt, existiert ein solcher Zyklus.

Um zugleich die Wege aufzuzeichnen, bedarf es einer weiteren (Wege-)Matrix, welche parallel zum einfachen Tripelalgorithmus mitgerechnet wird. Initialisiert wird diese mit den entsprechenden Knotennummern, welche die Kanten begrenzen, also beispielsweise in das Feld der ersten Zeile und der dritten Spalte 1 3. Die Felder der Hauptdiagonalen können leer bleiben oder mit Nullen gefüllt werden. Wird in der zur Rechnung benutzten Matrix der Wert in einem Feld durch die Summe aus zwei anderen Feldern ersetzt, so wird in der Wegematrix das entsprechende Feld durch Aneinanderreihen der Zahlen bzw. Zahlenfolgen aus den entsprechenden Feldern der Wegematrix ersetzt. Dabei wird der Inhalt des Feldes der entsprechenden Zeile vor den Inhalt des Feldes aus der entsprechenden Spalte gestellt. Von den Zahlen aus dem Feld derselben Spalte wird die letzte Zahl nicht übertragen[2].

		nach			
	No	1	2	3	4
von	1	0	1 2	1 3	1 4
	2	2 1	0	2 3	2 4
	3	3 1	3 2	0	3 4
	4	4 1	4 2	4 3	0

		nach			
	No	1	2	3	4
von	1	0	1 2	1 3	1 4
	2	2 1	0	2 3	2 4
	3	3 1	**3 1 2**	0	3 4
	4	4 1	4 2	4 3	0

Tabelle 2.3: Wegematrix zu Beginn und nach der 1. Iteration

[2] Sollten die Knoten mit Buchstaben bezeichnet werden, können diese natürlich auch im erweiterten Tripelalgorithmus verwendet werden.

Im Beispiel ist zu sehen, dass in der ersten Iteration der Weg von Knoten 3 nach Knoten 2 durch die Strecke Knoten 3 - Knoten 1 - Knoten 2 ersetzt wird. Führt man diesen Prozeß bis zur letzten Iteration weiter, so erhält man alle kürzesten Pfade.

2.3 Kapazitierte Kanten

2.3.1 Grundmodell

Transportprobleme mit kapazitierten Kanten werden durch dieselben Gleichungen wie das einfache Transportproblem beschrieben mit der zusätzlichen Einschränkung, dass

$$x_{ij} \leq C_{ij}, \quad \forall_{\,0<i\leq n}, \quad \forall_{\,0<j\leq m}, \tag{2.15}$$

wobei C_{ij} die Transportkapazität der Kante von Punkt i nach j ist. Soll eine optimale Erweiterung der Kantenkapazitäten mit einem festgesetzten Budget B erfolgen, läßt sich durch Einführung einer Hilfsvariablen das einfache Transportmodell erweitern, indem die einfache Kapazitätsrestriktion (2.15) durch die Ungleichung

$$x_{ij} - y_{ij} \leq C_{ij}, \quad oder \quad x_{ij} \leq C_{ij} + y_{ij} \tag{2.16}$$

ersetzt wird und die Budgetbeschränkung

$$\sum_{i=1}^{n}\sum_{j=1}^{m} b_{ij}y_{ij} \leq B, \tag{2.17}$$

sowie die Nichtnegativitätsbedingungen $y_{ij} \geq 0$ für alle $i = 1, 2, ..., n$ und $j = 1, 2, ..., m$ eingeführt werden. Die bekannten Variablen werden wie bisher verwendet, neu ist die Variable y_{ij}, welche die zusätzliche Kapazität angibt, um die x_{ij} durch die Investition erhöht wird. In Ungleichung (2.17) wird des weiteren durch b_{ij} die Höhe der Kosten angegeben, welche für die Erhöhung der Kapazität anfallen.

Gibt es keine Budgetrestriktion, sollte die Investition in einer Höhe getätigt werden, in der die Kosten gleich der Ersparnis sind. Nimmt man also die Kosten in die Zielfunktion, sieht das Gleichungssystem wie folgt aus:

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{m} b_{ij}y_{ij}, \tag{2.18}$$

unter den Nebenbedingungen

$$\sum_{j=1}^{m} x_{ij} \leq S_i, \quad \forall_{\,0<i\leq n}, \tag{2.19}$$

$$\sum_{i=1}^{n} x_{ij} \geq D_j, \quad \forall_{\,0<j\leq m}, \tag{2.20}$$

$$x_{ij} - y_{ij} \leq C_{ij}, \qquad oder \qquad x_{ij} \leq C_{ij} + y_{ij} \tag{2.21}$$

und den NNB

$$x_{ij}, y_{ij} \geq 0, \quad \forall_{\,0<i\leq n}, \quad \forall_{\,0<j\leq m}. \tag{2.22}$$

2.3.2 Diskrete Investitionen

Es kann die Situation auftreten, dass Investitionen nur im Ganzen getätigt werden können. Um dieses Problem zu lösen, werden binäre Variablen $w_{ij} \in \{0;1\}$ benutzt, welche auch verwendet werden, um die entstehenden Kosten F_{ij} in die Zielfunktion einzubeziehen. Die Zielfunktion lautet dann wie folgt:

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}F_{ij}. \tag{2.23}$$

Das Gleichungssystem besteht zudem aus den Ungleichungen (2.19) bis (2.22) des Transportproblems mit kapazitierten Kanten sowie der Restriktion:

$$y_{ij} \leq w_{ij}\frac{F_{ij}}{b_{ij}}, \quad \forall_{\,0<i\leq n}, \quad \forall_{\,0<j\leq m}. \tag{2.24}$$

Wenn mehr als die bisher mögliche Transportmenge über die Kante von i nach j transportiert werden soll, wird aufgrund der Ungleichung (2.24) $w_{ij} = 1$, wobei $\frac{F_{ij}}{b_{ij}}$ die maximale Steigerung der Kapazität ausdrückt. Wird y größer als 0, so geht F_{ij} komplett in die Zielfunktion ein. Gleichungssysteme mit binären Variablen sind jedoch nicht mehr linear und damit auch nicht mehr garantiert in Polynomialzeit lösbar. Ein Ausweg bietet mitunter die Relaxation, statt w_{ij} wird die Variable w^*_{ij} mit $0 \leq w^*_{ij} \leq 1$ verwendet, jedoch gibt es nach Wissen des Autors gelungene Transformationen nur für Probleme, bei welchen die Kanten entweder benutzt oder unbenutzt sind, beispielsweise das Matchingproblem oder das Standardisierungsproblem. Wobei z.B. das Travelling-Salesman-Problem sich durch einen hohen Anstieg der Anzahl der benötigten Restriktionen, welche zum Sperren von Zyklen eingeführt werden müssen, diesem Lösungsverfahren entzieht. Jedoch kann man mittels Relaxation die endogene Optimalität konstituieren, also ohne das Problem erneut zu berechnen oder den kompletten Lösungsweg zu kennen, die Optimalität einer Lösung aus sich selbst heraus bestätigen.

2.4 Variable Transportkosten

2.4.1 Grundmodell

Besteht die Möglichkeit der Senkung der Transportkosten durch Ausbau der Kanten bzw. Investition in die bestehende Infrastruktur, lässt sich zwar noch ein Gleichungssystem aufstellen, jedoch ist die Zielfunktion nicht mehr linear:

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{m} (c_{ij} - r_{ij})x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{m} R_{ij}. \tag{2.25}$$

Das Gleichungssystem besteht des weiteren aus den Ungleichungen (2.19) und (2.20), sowie einer Gleichung, welche den Zusammenhang zwischen der Investition R_{ij} und den eingesparten variablen Kosten r_{ij} angibt:

$$r_{ij}k_{ij} \leq R_{ij}, \quad \forall_{\,0<i\leq n}, \quad \forall_{\,0<j\leq m}. \tag{2.26}$$

Wobei k_{ij} die Ersparnis für den Transport einer Einheit auf einer Kante im Verhältnis zu den dafür in diese zu investierenden Kosten ist.

2.4.2 Diskrete Investitionen

Lassen sich diese Investitionen in die Infrastruktur nur stückweise vornehmen, entsteht also durch eine Investition von F_{ij} ein neuer Transportpreis c^*_{ij} auf der Kante von i nach j, so läßt sich ein lineares Gleichungssystem mit binären Variablen aufstellen. Dieses kann dann, wie in Abschnitt 2.3.2 erwähnt, auf seine Eigenschaften bezüglich der Relaxation überprüft werden. Die Zielfunktion lautet:

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{m} c^*_{ij}x^*_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}F_{ij}, \tag{2.27}$$

unter den Nebenbedingungen:

$$\sum_{j=1}^{m} x_{ij} + x^*_{ij} \leq S_i, \quad \forall_{\,0<i\leq n}, \tag{2.28}$$

$$\sum_{i=1}^{n} x_{ij} + x^*_{ij} \geq D_j, \quad \forall_{\,0<j\leq m}, \tag{2.29}$$

$$\frac{x^*_{ij}}{M} \leq w_{ij}, \tag{2.30}$$

wobei M eine sehr große (konstante) Zahl ist, und die NNB

$$x_{ij} \geq 0, \quad x^*_{ij} \geq 0, \quad i = 1, 2, ..., n, \; j = 1, 2, ..., m, \tag{2.31}$$

sowie die Definition der binären Variablen:

$$w_{ij} \in \{0; 1\}. \tag{2.32}$$

Im Gleichungssystem gibt $x_{ij} + x^*_{ij}$ den Transport von Quelle i nach Senke j in Abhängigkeit des Status der Kante an, wobei x_{ij} den Transport zu normalen Kosten c_{ij} und x^*_{ij} den Transport zu reduzierten Kosten c^*_{ij} darstellt. Erfolgte keine Investition in die Infrastruktur, so ist $w_{ij} = 0$ und damit nach Restriktion (2.30) $x^*_{ij} \leq 0$, also ist kein Transport zu reduzierten Kosten möglich. Ist dieser Transport möglich, so wird, da keine Kapazitätsgrenze angegeben ist, der komplette Transport zu den geringeren Kosten stattfinden.

2.5 Erweiterung auf verschiedene Transportgüter

Um das Modell eines Netzes darzustellen, wird jeder Knoten eines Netzes als Umladepunkt betrachtet. Es kann also ein Routing über mehrere verschiedene Knoten erfolgen, wenn dadurch eine Senkung der Transportkosten erreicht werden kann. Um die Transportkosten auf den Kanten zu senken, ist es möglich, die oben erwähnten diskreten Investitionen auf den Kanten vorzunehmen.

Des weiteren unterscheiden sich in diesem Modell die Produkte der Anbieter, d.h. im Grunde bietet jeder Anbieter genau ein Produkt an, welches von mehreren Marktteilnehmern nachgefragt wird. Eine solche Unterscheidung ist im einfachen Transportmodell nicht sinnvoll, da bei konstanten Kantenkosten der Transport eines jeden Gutes für sich optimiert werden kann.

Formal dargestellt ergibt sich für dieses Problem folgende Zielfunktion:

$$\min Z = \sum_{g=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(c_{ij}^{1}x_{gij}^{a} + c_{ij}^{2}x_{gij}^{b}\right) + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}F_{ij}, \tag{2.33}$$

wobei zwischen dem Transport eines Gutes über Strecken ohne Infrastruktur und über Strecken mit Infrastruktur unterschieden wird. Die Variable x_{gij}^{a} gibt also die Menge des Gutes g an, welche von Knoten i nach Knoten j zu Normalkosten transportiert wird, während x_{gij}^{b} den Transport über eine 'ausgebaute' Strecke angibt. Entsprechend werden auch die variablen Kosten c_{ij}^{1} (einfache Kosten) und c_{ij}^{2} (reduzierte Kosten) angesetzt. Zusätzlich gehen die Kosten für die Erhaltung der Infrastruktur in Form von F_{ij} in die Zielfunktion ein.

Als eine Nebenbedingung für die Konsistenz der Flüsse muss gelten, dass die Menge der Güter, welche zu einem Knoten transportiert werden, abzüglich der weitergeleiteten Menge, dem Bedarf bzw. der Nachfrage entspricht. Die Zielfunktion hat damit folgende Nebenbedingungen:

$$\sum_{i=1}^{n}\left(x_{gik}^{a} + x_{gik}^{b}\right) - \sum_{j=1}^{n}\left(x_{gkj}^{a} + x_{gkj}^{b}\right) = D_{k}^{g}, \quad \forall\, {}_{0<g\leq n}, \forall\, {}_{0<k\leq n}, \tag{2.34}$$

$$\frac{x_{gij}^{b}}{M} \leq w_{ij}, \tag{2.35}$$

wobei M eine sehr große (konstante) Zahl ist. Sollte der Knoten ein Anbieter sein, wobei davon ausgegangen wird, dass Knoten g der einzige Anbieter für Gut g ist, so gilt $D_{i}^{g} < 0$ für $i = g$. Die Konsistenzbedingung $\sum_{i=1}^{n} D_{i}^{g} = 0$ mit $D_{i}^{g} \geq 0$ für alle $0 < g \leq n, i \neq g$ muss für die exakte Lösung ebenfalls eingehalten werden.

Des weiteren gelten die NNB:

$$x_{gij}^{a} \geq 0, \quad x_{gij}^{b} \geq 0, \quad \{g,i,j\} = 1,2,...,n, \tag{2.36}$$

sowie die Definition der binären Variablen $w_{ij} \in \{0;1\}$.

Es handelt sich also um ein Transportproblem, bei welchem die Auswahl zwischen zwei alternativen Kostenfunktionen besteht. Aggregiert man diese so, dass man sich immer auf dem effizienten Rand befindet, also dort wo die Transportkosten geringer sind, entsteht die in Abbildung 2.2 dargestellte Funktion für die Gesamtkosten.

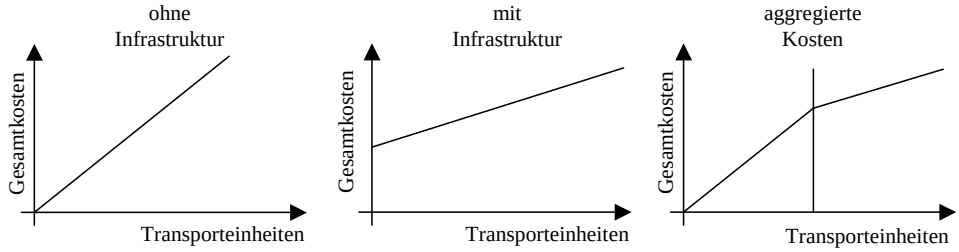

Abbildung 2.2: Transportkosten für das Infrastrukturproblem

Abbildung 2.3 zeigt die daraus resultierende degressive Durchschnittskostenfunktion der Transportmenge. Da jedoch nur Transportprobleme mit linearen oder progres-

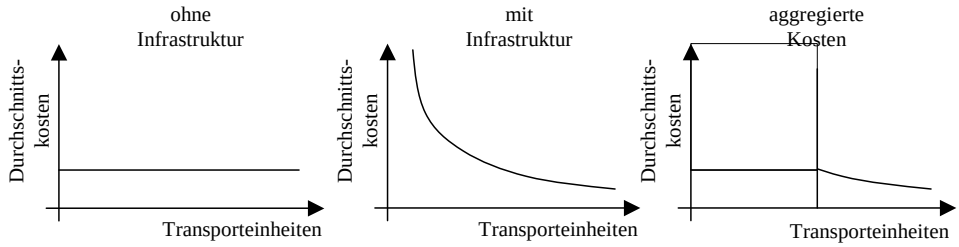

Abbildung 2.3: Durchschnittliche Kosten für das Infrastrukturproblem

siven Kostenfunktionen mit dem klassischen Transportalgorithmus oder Modifikationen davon lösbar sind, muss man das in dieser Form erweiterte Transportproblem mit anderen geeigneten Methoden lösen.

2.6 Eigenschaften

Zur Abgrenzung verschiedener Infrastrukturprobleme voneinander kann man mehrere Eigenschaften isolieren. Die Eigenschaften können sich auf die Transportkostenmatrix mit oder ohne Infrastruktur beziehen.

2.6.1 Symmetrie

Zunächst einmal kann anhand der Kostenmatrix des zugrunde liegenden Graphen in symmetrische und asymmetrische Probleme unterschieden werden. Für ein symmetrisches Problem muß also gelten:

$$c_{ij} = c_{ji}, \quad \forall_i, j \in \{1, .., n\}. \tag{2.37}$$

Gilt diese Gleichung für mindestens ein Paar (ij) nicht, sind also die Kosten von i nach j und die Kosten von j nach i verschieden, so liegt ein asymmetrisches Infrastrukturproblem vor. Offen ist jedoch, ob diese Bedingung auch für die Kostenmatrix der Infrastrukturverbindungen gelten muß. Naheliegend ist, zwischen komplett symmetrischen und teilweise symmetrischen Problemen zu unterscheiden, abhängig davon, ob die Eigenschaft der Symmetrie nur für die einfachen Kanten oder auch für die Infrastrukturkanten erfüllt ist, also $c^*_{ij} = c^*_{ji}$ ebenfalls gilt.

2.6.2 Dreiecksungleichung

Die Dreiecksungleichung besagt, dass für jedes Dreieck die Summe zweier beliebiger Kanten immer größer als die dritte Kante ist. Für die Kostenmatrix muß also gelten:

$$c_{ij} \leq c_{ik} + c_{kj}, \quad \forall_i, j, k \in \{1, .., n\}. \tag{2.38}$$

Auch hier stellt sich die Frage, wie diese Eigenschaft in Bezug auf die Infrastrukturkanten zu bewerten ist. Wie auch bei der Symmetrie scheint es sinnvoll, Graphen, welche die Dreiecksungleichung komplett erfüllen und solche, bei denen dies nur für die normalen Kanten gilt, zu unterscheiden. Für viele Probleme der Praxis ist es jedoch wahrscheinlich, dass sie der Dreiecksungleichung genügen.

2.6.3 Euklidische Distanzen

Gilt für ein Problem die Eigenschaft der Symmetrie und die Dreiecksungleichung, so erfüllen abstrakte Probleme meist noch eine weitere Eigenschaft. Jeder Knoten kann durch seine Position im n-dimensionalen, euklidischen Vektorraum repräsentiert werden. Beschränkt man sich auf den 2-dimensionalen Raum, so muß gelten:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \quad \forall_i, j \in \{1, .., n\}, \tag{2.39}$$

wobei x_i und y_i die Koordinaten der Stadt i bezeichnen sollen.

Die Probleme, welche im Rahmen dieser Arbeit untersucht wurden, erfüllen alle drei Eigenschaften, wobei die ersten beiden aus der dritten ableitbar sind.

2.7 Ähnliche Probleme

Probleme, welche eine ähnliche Struktur aufweisen, jedoch im Rahmen dieser Arbeit nicht ausführlich behandelt werden können, sollen an dieser Stelle kurz angeschnitten werden.

2.7.1 Das Standardisierungsproblem

Basierend auf Überlegungen zu Netzeffekten wurde das Standardisierungsproblem in [Buxmann 1996] formuliert. Grundlage ist ein Netzwerk aus Knoten und Kanten, wobei die Knoten -wie im Infrastrukturmodell- die Akteure eines Netzwerkes darstellen. Über die Kanten findet eine Kommunikation bzw. ein Transport statt, für welchen Kosten anfallen. Im Gegensatz zum vorgestellten Modell können diese Kosten jedoch nicht durch Investitionen in eine Kante gesenkt werden, sondern es muss eine 'homogene' Investition in den beiden die Kante begrenzenden Knoten stattfinden. Diese Investition wird auch als Einführung eines Standards bezeichnet. Diese sind nicht spezifisch für jeden Knoten, sondern ein dritter Knoten könnte auch diesen Standard einführen und auf diese Art, falls vorhanden, seine Kommunikationskosten mit den Knoten, welche auch diese Investition getätigt haben, senken. Als mathematische Formulierung des Modells ergibt sich für das einstufige Problem:

$$\min Z = \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} e_{ij} y_{ij} \tag{2.40}$$

unter den Nebenbedingungen

$$y_{ij} \leq x_i + x_j - 2 \quad \forall_i, j \in \{1, .., n\}, \tag{2.41}$$

$$x_i \in \{0, 1\}, \qquad y_{ij} \in \{0, 1\} \tag{2.42}$$

Wobei a_i die Kosten für die Investitionsentscheidung an Knoten i sind, und e_{ij} die Kosten, welche durch Investition in Knoten i und Knoten j eingespart werden können. Zwar würden die Einsparungen über mehrere Perioden anfallen, die abdiskontierte Summe ist jedoch eine Konstante und kann deshalb in e_{ij} zusammengefasst werden. Die Aufhebung der Ganzzahligkeitsbedingung ist für das einfache Modell möglich,

soll jedoch an dieser Stelle bestehen bleiben, da die grundsätzliche Struktur des Modells für Standardisierungsentscheidungen veranschaulicht werden soll.

Das Standardisierungsmodell kann nahezu beliebig erweitert werden, beispielsweise kann man die Auswahlentscheidung zwischen mehreren Standards mit oder ohne Intermediäre hinzunehmen, Unsicherheit bezüglich der Kosten oder Perioden einführen. Prinzipiell handelt es sich jedoch immer um direkte Netzeffekte.

2.7.2 Standortplanung

Eine auf die Anordnung von Elementen abzielende Problemstellung ist die Standort- bzw. Layoutplanung. Voraussetzung für das hier vorgestellte Grundmodell ist, dass die Kosten für die Neuordnung aufgrund der hohen operativen Transportkosten vernachlässigt werden können. Die anzuordnenden Elemente können Maschinen, Arbeitsplätze, Werkstätten oder Unternehmen bzw. Unternehmensbereiche sein[3], in jedem Fall kann man davon ausgehen, dass die Lage der Organisationseinheiten keinen Einfluss auf die Höhe der Erlöse hat. Das erklärte Ziel der Standortplanung ist die Minimierung der Summe aller anfallenden standortbedingten Transport-, Lager- und Produktionskosten. Dieses Ziel ist jedoch nicht operational.

Die Grundversion des quadratischen Zuordnungsproblems (QAP)[4] geht von folgenden Annahmen aus:

- Es existieren n (potentielle) Standorte zur Anordnung der Objekte.
- Auf jedem Platz kann genau ein Objekt angeordnet werden.
- Der kürzeste (nutzbare) Weg d_{jk} ist (für alle $j, k \in \{1, .., n\}$) Standorte bekannt.
- Zwischen den Maschinen h und i findet ein Materialaustausch der Menge t_{hi} Einheiten pro Periode statt.[5]
- Die Kosten des Transportes zwischen zwei Standorten sind proportional zu der zu transportierenden Menge und der zurückgelegten Entfernung. Ohne Beschränkung der Allgemeinheit kann immer der Proportionalitätsfaktor 1 verwendet werden.

Zur mathematischen Formulierung des Modells wird noch die binäre Entscheidungsvariable x_{hi} benötigt, welche wie folgt definiert ist:

$$x_{hi} = \begin{cases} 1 & falls\ Objekt\ h\ auf\ Platz\ i\ anzuordnen\ ist, \\ 0 & sonst \end{cases} \tag{2.43}$$

Die Zielfunktion des Modells, welche die Transportkosten in Abhängigkeit vom Standort eines jeden Objektes berechnet, lautet wie folgt:

$$\min Z = \sum_{h=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} t_{hi}\, d_{jk}\, x_{hj}\, x_{ik}\ , \tag{2.44}$$

[3] vgl. Domschke, Standortplanung, innerbetriebliche S. 3950-3962 in [EdB93]

[4] Allgemein wird auch der Anglizismus Quadratic Assignment Problem verwendet, weshalb auch hier das Kürzel QAP gewählt wurde.

[5] Die Menge t_{hi} wird auch als Transportintensität bezeichnet.

unter den Nebenbedingungen

$$\sum_{j=1}^{n} x_{hj} = 1 \quad \forall_{h\,=\,1,\ldots,n}, \tag{2.45}$$

$$\sum_{h=1}^{n} x_{hj} = 1 \quad \forall_{j\,=\,1,\ldots,n}, \tag{2.46}$$

$$x_{ij} \in \{0,1\} \quad \forall_{h,j\,=\,1,\ldots n}. \tag{2.47}$$

Da QAPs trotz der restriktiven Annahmen NP-vollständig sind[6], spielen exakte Lösungen, welche auch nur für relativ kleine Probleme existieren, in der Praxis kaum eine Rolle. Bei den heuristischen Verfahren gibt es eine Reihe Konstruktionsverfahren, welche auch in Kombination mit Verbesserungsverfahren zur Anwendung kommen. In jüngerer Zeit wurden auch mit dem später vorgestellten Verfahren Simulated Annealing gute Ergebnisse erzielt[7].

2.7.3 Das File-Allocation-Problem

Das Problem der optimalen Verteilung von Informationen, welche unter zu vernachlässigenden Kosten beliebig vervielfältigt werden kann, gewinnt durch den heutigen Einsatz von komplexen Computersystemen verbunden zu Netzwerken verstärkt an Bedeutung. Ziel ist es, durch die Verteilung von Kopien eines Datums an verschiedenen Orten, die operativen Kosten eines Systems zu minimieren. Modelliert wurde dieses Problem erstmals von Chu [Chu 69] unter dem Namen File-Allocation-Problem (FAP). In seinem Modell werden explizit Speicherobergrenzen der Knoten[8] und maximale Wartezeiten berücksichtigt. Eine einfachere Version, welche diese Einschränkungen nicht berücksichtigt, wurde 1972 von Casey [Casey 72] entwickelt, und soll hier beispielhaft dargestellt werden.

Folgende Annahmen werden getroffen:

- Jeder Transaktionsprozess ist eindeutig identifiziert und verursacht Kosten proportional zu Umfang und Kommunikationskosten auf den benutzten Verbindungen.
- Es werden ausschließlich Transaktionen von Daten berücksichtigt, da davon ausgegangen werden kann, dass Programme zum Zugriff auf Daten relativ klein sind und selten aktualisiert werden müssen.
- Abfrageoperationen müssen auf genau eine Kopie bzw. das Original eines Datums zugreifen.
- Aktualisierungsoperationen[9] müssen zur Aufrechterhaltung der Integrität und Konsistenz des Systems auf alle Kopien eines Datums zugreifen.

[6] vgl. [Sahni et al. 1976]

[7] vgl. [Domschke 1993]

[8] Die Knoten werden im Kontext der Informatik oft auch als Prozessoren bezeichnet, um keine zusätzliche Verwirrung zu stiften, wird auch hier der Begriff Knoten verwendet.

[9] Einigen werden die Begriffe Query- und Update Transaktionen geläufiger sein, jedoch wurde zur Unterstützung der allgemeinen Verständlichkeit auf die deutschen Begriffe zurückgegriffen.

- Die Daten werden immer auf dem kürzesten Weg versandt, d.h. Routing-Strategien, welche zu schlechteren Szenarien führen, werden ausgeschlossen.
- Die Kosten, welche für Lagerung und Pflege an den Knoten anfallen, sind abhängig von dem an einem Knoten verfügbaren Umfang an Daten.

Insofern keine zeitliche Reihenfolge der Abfragen berücksichtigt wird, also ein garantiert aktuelles Datum nur bei den Knoten abgefragt werden kann, welche durch Aktualisierungsoperationen erreicht werden, ergibt sich folgendes Modell:

Q_i^d	=	die Menge an Daten, welche durch Nachfrage von Knoten i an dDatum transportiert werden müssen.
U_j^d	=	die Menge an Daten, welche durch Aktualisierungen von Knoten j an Datum d transportiert werden müssen.
C_j^d	=	die Speicher- und Pflegekosten für Datum d auf Knoten j
t_{ij}	=	die Transportkosten zwischen Knoten i und Knoten j.
S^d	=	die Größe von Datum d.
N	=	die Anzahl der Knoten.
F	=	die Anzahl der Daten.

Als Entscheidungsvariablen gehen X_j^d und y_{ij}^d in das Modell ein, wobei $X_j^d = 1$, wenn Datum d auf Knoten j gespeichert wird und $y_{ij}^d = 1$ falls Abfragen für Datum d von Knoten i an Knoten j geleitet werden.

Speicherkosten

$$Z1 = \sum_{j=1}^{N} \sum_{d=1}^{F} c_j^d x_j^d. \tag{2.48}$$

Kommunikationskosten durch Abfrageoperationen

$$Z2 = \sum_{i=1}^{N} \sum_{j=1}^{N} t_{ij} \left(\sum_{d=1}^{F} y_{ij}^d Q_i^d \right). \tag{2.49}$$

Kommunikationskosten durch Aktualisierungsoperationen

$$Z3 = \sum_{i=1}^{N} \sum_{d=1}^{F} U_i^d \left(\sum_{j=1}^{N} X_j^d t_{ij} \right). \tag{2.50}$$

Das Optimierungsproblem, welches das FAP bildet, ist damit also:

$$min\, Z = Z1 + Z2 + Z3 \tag{2.51}$$

unter den Nebenbedingungen:

$$\sum_{j=1}^{N} y_{ij}^d = 1 \quad \forall_{i,d}, \tag{2.52}$$

$$X_j^d - y_{ij}^d \geq 0 \quad \forall_{i,j,d}. \tag{2.53}$$

Restriktion 2.52 sorgt dafür, dass die Nachfrage von Knoten i nach Datum d befriedigt wird, Restriktion 2.53 sorgt dafür, dass Abfragekommunikation von Knoten i nach Datum d über Knoten j geleitet werden kann, wenn eine Kopie des Datums d vorrätig ist. Zusätzlich können Restriktionen zur Einhaltung von Kapazitäten an den Knoten, Beschränkungen der Kanten oder Zeitgrenzen hinzugefügt werden, wenn das Problem dadurch besser erfaßt wird.

Durch seine Anwendung im technischen Umfeld sind bereits eine Reihe hoch spezialisierter Lösungsansätze für das FAP entstanden. Einige stützen sich auch auf die starke Strukturähnlichkeit zum Knapsack-Problem, worauf jedoch hier nicht weiter eingegangen werden kann. Forschungsbedarf besteht insbesondere in Hinsicht auf die Reaktion solcher Verfahren in Bezug auf Unsicherheit bezüglich der Verfügbarkeit der Knoten.

2.7.4 Abgrenzung

Das Infrastrukturproblem wurde in dieser Form noch nicht in der Literatur behandelt. Prinzipiell decken all diese Probleme bestimmte Strukturen ab, welche in Netzwerken auftreten können. Aus der Kenntnis der einzelnen Modelle und Verfahren zur Optimierung, welche für die einzelnen Ausprägungen geeignet erscheinen, sollte man den richtigen Weg finden, um bestimmte praktische Probleme zu adressieren.

Kapitel 3

Struktur von Netzwerken - Graphentheorie

Um Zusammenhänge zwischen bestimmten Netzwerkstrukturen und den Eigenschaften, welche diese Strukturen in Bezug auf bestimmte Prozesse ausüben, feststellen zu können, bedarf es quantitativer Maße, mit welchen diese Strukturmerkmale erfassbar werden. Zunächst werden zwei einfache Maße beschrieben, welche ganzzahlig sind, und deren Aussage nicht ins Verhältnis zu einem anderen Maß des Graphen oder eines Subgraphen gesetzt sind. Die darauf folgenden Maße geben relationale Maßzahlen an.

3.1 Nicht-relationale Maße

3.1.1 Bettie Zahl oder Zyklomatische Nummer

Die Zyklomatische Nummer (μ) eines Graphen (G), eine der fundamentalen Maßzahlen der Graphentheorie, geht zurück auf C. Berge[1]. Sie lässt sich einfach als:

$$\mu(G) = e - v + p \tag{3.1}$$

schreiben, wobei e die Anzahl der Kanten, v die der Knoten und p die Anzahl der unabhängigen Subgraphen angibt.

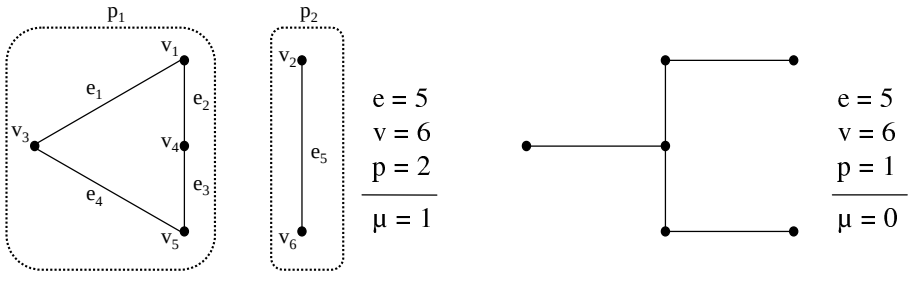

Abbildung 3.1: Nichtverbundener Graph (links) und Baumstruktur (rechts)

[1] vgl. [Berge 1958]

Dieser Index hat zwei wichtige Eigenschaften: (1) er gibt für einen linearen Graph die Anzahl der unabhängigen Zyklen a, (2) für einen verbundenen Graphen gibt er die maximale Zahl an fundamentalen Zyklen an. Augenscheinlich haben Bäume damit immer eine Zyklomatische Nummer von Null, während stark vernetzte Graphen hohe Werte aufweisen.

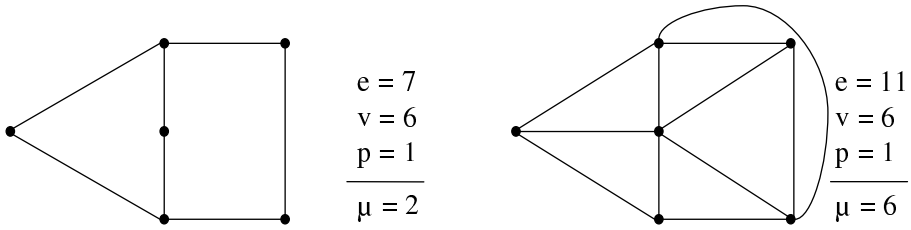

Abbildung 3.2: Weitere verbundene Graphen

Einschränkend gilt allerdings, daß ohne Kenntnis des Netzwerkes die Aussagekraft der Zyklomatischen Nummer recht gering ist. Hätte beispielsweise das Internet eine Zyklomatische Nummer von 100, so wäre es wohl kein Problem, ein Land zu isolieren. Für ein unternehmensinternes Netzwerk würde dieser Wert wahrscheinlich als übertrieben hoch gelten.

3.1.2 Durchmesser

Der Durchmesser (δ) ist die maximale Anzahl von Kanten auf den kürzesten Pfaden zwischen jedem Knotenpaar in einem Netzwerk. Per Definition ist der Durchmesser in einem verbundenen Graphen (G) der längste (zyklenfreie) Weg. Man kann schreiben:

$$\delta(G) = \max d(e_i, e_j) \quad \forall i, j, \tag{3.2}$$

wobei $d(x, y)$ die minimale Kantenzahl zwischen Knoten x und Knoten y angibt.

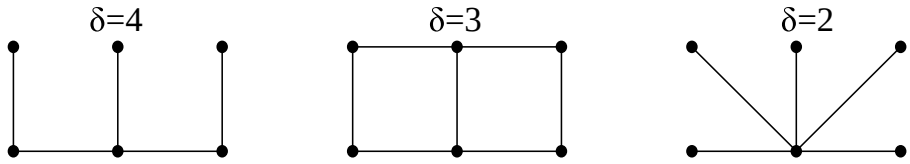

Abbildung 3.3: Durchmesser als Maß für die Vernetzung

Allgemein gilt: Gibt es nur wenige Kanten, so steigt dieser Index mit wachsender Größe des Netzwerkes. Wächst jedoch die Anzahl der Kanten bei konstanter Knotenzahl[2], dann sinkt der Durchmesser. Damit kann der Durchmesser als ein topologisches Maß für den Grad der Vernetzung benutzt werden. Die Aussagekraft dieses Maßes für sich ist jedoch noch gering, kann aber für die Erzeugung aussagekräftigerer relativer Maße verwendet werden.

Zur Berechnung des Durchmessers eignet sich der Tripelalgorithmus[3], dahingehend modifiziert, das alle bestehenden Verbindungen mit der Länge 1 angegeben werden. Der dann berechnete längste kürzeste Pfad entspricht dem Durchmesser.

[2] Dies kann als steigende Vernetzung angesehen werden.

[3] siehe Abschnitt 2.2.2

3.2 Relationale Maße

3.2.1 Alpha

Das Maß α wird direkt aus der Zyklomatischen Nummer abgeleitet, und kann als Verhältnis der Anzahl der elementaren Zyklen zu der maximal möglichen Anzahl bei gegebener Knotenzahl verstanden werden. Die maximale Zahl der Zyklen μ_{max} kann über die Zyklomatische Nummer für den maximal verbundenen Graphen[4] errechnet werden, also:

$$\mu_{max} = e_{max} - v + 1 \quad mit \quad e_{max} = \tfrac{v(v-1)}{2} \tag{3.3}$$

$$\mu_{max} = \frac{(v-1)(v-2)}{2}, \tag{3.4}$$

wobei v die Anzahl der Knoten angibt. Für α erhält man also:

$$\alpha = \frac{\mu}{\mu_{max}} = \frac{2 * \mu}{(v-1)(v-2)} \quad bzw. \quad \alpha = \frac{2 * (e - v + p)}{(v-1)(v-2)}. \tag{3.5}$$

Durch Verwendung dieses Index kann man Graphen eine Art Grad der Vernetztheit, welcher sich zwischen 0 und 1 befindet, zuweisen. Für vollständig vernetzte Graphen ergibt sich immer ein α-Wert von 1 (obere Grenze), während verbundene oder unverbundene Bäume einen α-Wert von 0 haben. Somit kann der α-Wert als Erreichungsgrad der maximalen Vernetzung interpretiert werden.

3.2.2 Beta

Der β-Index ist ein sehr einfaches Maß, welches die Anzahl der Kanten und Knoten ins Verhältnis setzt. Die Formel lautet:

$$\beta = \frac{e}{v}, \tag{3.6}$$

wobei e die Anzahl der Kanten und v die Anzahl der Knoten angibt. Die Aussage in Bezug auf Transportnetzwerke ist ähnlich der von α und μ; besitzt das Netzwerk eine komplizierte Struktur, so hat es einen hohen β-Wert, Netzwerke mit einfacher Struktur besitzen dagegen einen niedrigen β-Wert. Normiert man den β-Wert auf einen Wert zwischen 0 und 1, so ist er gleich dem im nächsten Abschnitt beschriebenen γ-Wert, weshalb es selten sinnvoll ist, beide Werte anzugeben.

3.2.3 Gamma

Der γ-Wert ist ein Verhältnismaß zwischen Knoten und Kanten eines gegebenen Graphen. Für einen nicht-planaren Graphen hat er folgende Formel:

$$\gamma = \frac{e}{\frac{v(v-1)}{2}}. \tag{3.7}$$

[4] Die maximale Zahl elementarer Zyklen in einem nicht-planaren Graphen ist auch gleich der Anzahl der Kanten im maximal verbundenen Graphen $\frac{v(v-1)}{2}$ minus der Anzahl der Kanten auf dem vollständigen Baum $(v-1)$. Für einen planaren Graphen ergibt sich $\mu_{max} = 2v - 5$.

Damit ist der γ-Wert der Quotient der beobachteten Anzahl der Kanten und der maximalen Anzahl der Kanten. Für einen planaren Graphen schreibt sich die Formel dann wie folgt:

$$\gamma = \frac{e}{3(v-2)}. \tag{3.8}$$

In beiden Fällen ist der γ-Wert im Bereich zwischen 0 und 1[5], und kann ähnlich dem α-Wert als Grad der Verbundenheit interpretiert werden. Für ein Transportnetzwerk wäre die Annahme der planaren Struktur naheliegend. Werden jedoch beispielsweise die Direktverbindungen als Kanten betrachtet, so würde diese Annahme nicht mehr sinnvoll erscheinen.

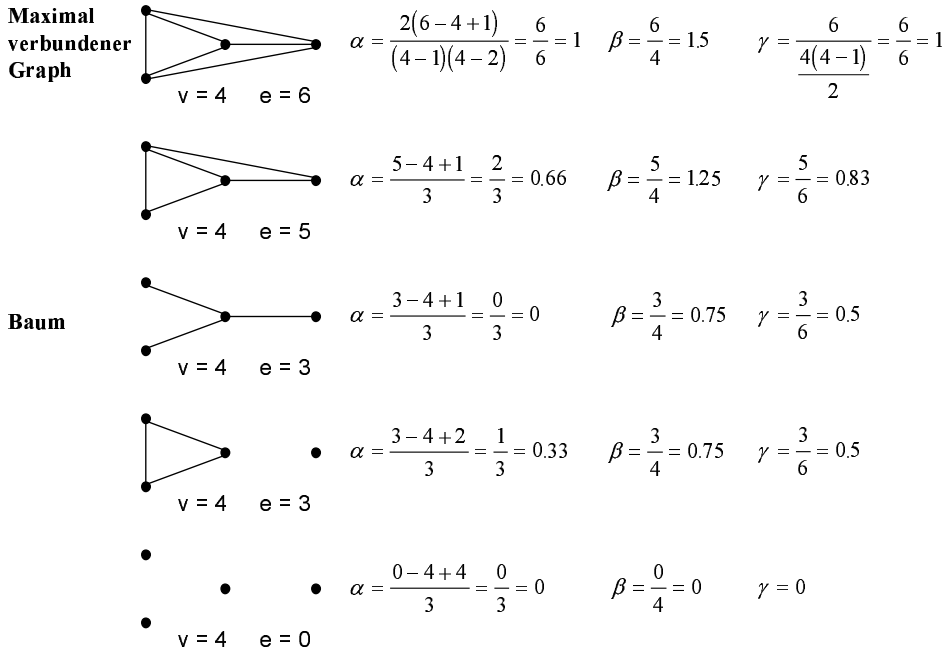

Abbildung 3.4: Anwendungen des α-, β- und γ-Wertes

3.2.4 Eta

Der η-Wert errechnet sich aus der Länge des Netzwerkes geteilt durch die Anzahl der Kanten. Die für die Berechnung verwendete Formel lautet:

$$\eta = \frac{M}{e}, \tag{3.9}$$

Er gibt also lediglich die durchschnittliche Länge der Kanten an, und soll hier nur der Vollständigkeit halber erwähnt werden.

[5] Die maximale Zahl der Kanten in einem planaren Graphen ist $3v - 6$.

3.2.5 Pi

Wie der η-Wert gibt der π-Wert ein Verhältnis zwischen Ausdehnung des Netzes und Dichte an. Die Bezeichnung π-Index weist auf die logische Ähnlichkeit mit der Flächenberechnung des Kreises hin, in welcher die irrationale Zahl π eine wichtige Rolle spielt.[6]

Errechnet man in Analogie dazu den π-Wert eines Netzwerkes, so ergibt sich folgende Formel:

$$\pi = \frac{C}{d}, \tag{3.10}$$

wobei C die Summe der Kantenlängen und d den Durchmesser, also die maximale Ausdehnung in Längeneinheiten gemessen, angibt. Wie in Abschnitt 3.1.2 angegeben, bezeichnet man den längsten kürzesten Weg durch das Netzwerk als Durchmesser. Wird der π-Wert für einen nicht-zusammenhängenden Graphen berechnet, so gibt man das arithmetische Mittel der π-Werte der Subgraphen an.

3.2.6 Theta

Ein Maß, welches den gesamten Fluss oder die zurückgelegten Strecken der Flüsse eines Netzwerkes ins Verhältnis zu seiner Knotenzahl setzt, ist der θ-Index. Die Formeln sind entsprechend:

$$\theta = \frac{T}{v} \quad oder \quad \theta = \frac{M}{v}, \tag{3.11}$$

wo v die Anzahl der Knoten des Netzwerkes angibt, und die Variable T den totalen Fluss bzw. M die zurückgelegte Strecke der einzelnen Flüsse im Netzwerk angibt.

3.3 Zusammenfassung

Die hier angeführten Maße aus der Graphentheorie sollen in der Evaluationsphase im nächsten Kapitel dazu dienen, Zusammenhänge zwischen Strukturen und Effekten, welche während des Prozesses oder im Gleichgewicht auftreten, aufzudecken und vorhersagen zu können.

[6] Die Formel zur Berechnung des Flächeninhaltes eines Kreises lautet $A = \frac{1}{4}\ \pi * d^2$, wobei π ungefähr 3.14159 ist und d den Durchmesser des Kreises angibt.

Kapitel 4

Lösungsverfahren

4.1 Klassifizierung

Die hier vorgestellten Lösungsansätze für das Infrastrukturproblem lassen sich in zentrale Verfahren, also Lösungsverfahren, bei welchen ein Zentralplaner über alle entscheidungsrelevanten Informationen verfügt, und dezentrale Lösungsverfahren, unter die auch die marktlichen fallen, unterteilen. Der Unterschied besteht in der Zielfunktion. Während für die Zentralplanung die Wohlfahrt bzw. der Gesamtnutzen der Akteure optimiert wird, optimieren bei der dezentralen Planung die einzelnen Akteure nur ihren Nutzen. Ist dieser transferierbar und bestehen keine eine optimale Allokation behindernden Einflüsse[1], so sollte sich bei beiden Koordinationsformen die gleiche pareto-effiziente Lösung einstellen[2].

Bei einem dezentralen Lösungsansatz unterscheidet man des weiteren, ob es sich um ein einperiodiges und damit einstufiges oder ein mehrstufiges bzw. mehrperiodiges Entscheidungsmodell handelt. Während bei einer einmaligen simultanen Entscheidung mehrerer Akteure nur ex ante Erwartungswerte gebildet werden können und aufgrund dieser entschieden wird, können bei mehrperiodigen Verfahren Entscheidungen getestet und verworfen werden. Eine nähere Analyse und der Aufbau eines Entscheidungsmodells findet sich in den folgenden Abschnitten.

Die durch zentrale Verfahren ermittelten Optimallösungen bzw. sehr guten Lösungen[3], welche die pareto-effizienten Lösungen dieses Koordinationsproblems darstellen, werden als Benchmark für dezentrale Entscheidungsmodelle verwendet. Zur Ermittlung dieser Lösungen dient das naturanaloge Verfahren Simulated Annealing.

Wenn marktliche Koordination als Alternative zur zentralen Koordination für ein solches Problem angesehen wird, wäre es wünschenswert, wenn das Ergebnis eines einfachen Optimierungsverfahrens im allgemeinen von den dezentralen Verfahren erreicht oder überboten wird. Im folgenden Abschnitt wird deshalb ein reines Verbesserungsverfahren[4] kurz dargestellt.

[1] Ein solcher Einfluss ist jedoch die Unmöglichkeit, durch die einschränkten Informationen, über welche die einzelnen Akteure verfügen, die optimale Verteilung zu kennen.

[2] Es wird jedoch gezeigt, dass dies hier nicht der Fall ist.

[3] Das zur Ermittlung der optimaler Lösungen implementierte Verfahren der vollständigen Enumeration sprengt für größere Probleme den Zeitrahmen dieser Diplomarbeit

[4] Gebräuchlich ist auch der Begriff Bergsteigerverfahren, welcher an das Bild eines Lösungsgebirges anlehnt.

4.2 Zentrale Ansätze

Die Zielfunktion, welche für die zentralen Ansätze gilt, wurde in Kapitel 2 aufgestellt, und hat folgende Form:

$$\min Z = \sum_{g=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(c_{ij}^{1}x_{gij}^{a} + c_{ij}^{2}x_{gij}^{b}\right) + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}F_{ij}. \tag{4.1}$$

Sie ist auch die Kosten- bzw Energiefunktion $f : I_n \to \mathbb{R}$, welche verwendet wird, um erzeugte Netze zu bewerten.

Für die im weiteren beschriebenen Verfahren muss zudem noch eine Nachbarschaft S festgelegt werden. Diese Nachbarschaft definiert das Erzeugen einer neuen Lösung I_{i+1} aus einer bekannten Lösung I_i für das zu optimierende Problem, sozusagen einen Schritt im Lösungsraum. Wobei jede Lösung eine in diesem Fall durch die Zielfunktion feststehende Qualität hat. Bewegt man sich nun über diese Nachbarschaft von Lösung zu Lösung, so entsteht eine Art Gebirge, in welchem es gilt, den höchsten Punkt, also die Lösung, welche durch die Zielfunktion am höchsten bewertet wird, zu finden. Zur Veranschaulichung soll zunächst ein einfaches 4 Städte Infrastrukturproblem dienen.

Abbildung 4.1: Beispielnetz (links) und vereinfachte Darstellung (rechts)

Die vier Verbindungen sind im linken Netzwerk explizit dargestellt, würden jedoch schnell dazu beitragen, selbst in einfachen Netzen die Übersicht zu verlieren. Deshalb werden im weiteren Verlauf der Arbeit gegenläufige Infrastrukturverbindungen wie in der rechten Abbildung dargestellt. Definiert man als Nachbarschaft das Hinzufügen oder Entfernen einer Kante, so ergibt sich das in Abbildung 4.2 dargestellte Bild.

Wie leicht zu erkennen ist, können für ein n-Knoten Problem $n(n-1)$ verschiedene gerichtete Kanten auftreten. Also existieren $2^{n(n-1)}$-Lösungen und für die beschriebene Nachbarschaft $n(n-1)$ Nachbarn.

4.2.1 Iterative Improvement

Ein einfaches Verfahren könnte nun zufällig eine Lösung I_0 erzeugen, und alle Nachbarn daraufhin untersuchen, welcher die beste Ausprägung hinsichtlich der Zielfunktion hat. Ist dessen Zielerreichungsgrad zudem noch höher als der der vorher betrachteten Lösung, so wird die Suche von dort aus fortgesetzt. Findet sich kein besserer Nachbar, ist die Suche abgeschlossen[5].

[5]Wiederholt man den gesamten Vorgang, besteht die Möglichkeit, eine bessere Lösung zu finden. Für unendlich viele Durchläufe ist die Konvergenz des Verfahrens ins Optimum offensichtlich.

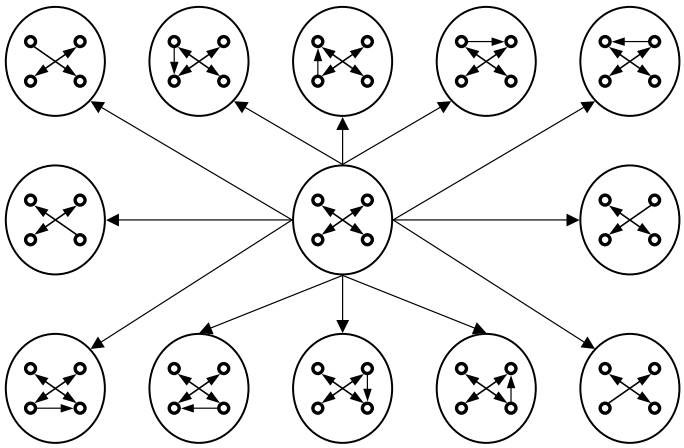

Abbildung 4.2: Nachbarschaft für eine Lösung

procedure Iterative Improvement
generate initial solution $I_0 \in S$
initialize neighborhood relation $N \subseteq S$
set variable $i := 0$
do
$i := i + 1$
$I_i = min\{f(N(I_{i-1}))\}$
while $C(I_i) < C(I_{i-1})$
$I_{opt} = I_{i-1}$
end.

4.2.2 Simulated Annealing

Die Grundidee des Simulated Annealing (SA), welche auf zeitgleichen Arbeiten von Kirkpatrick [Kirkpatrik 1983] und Cerny [Cerny 1985] basiert, entstammt dem physikalischen Vorgang der Abkühlung von Metallen. In Analogie dazu sucht man mittels SA den Lösungsraum, welcher durch ein Problem und seine Nachbarschaft dargestellt wird, ab. Ziel ist es, dass der Prozess möglichst gute Lösungen in möglichst geringer Zeit findet. Dabei handelt es sich um eine approximative Lösung des Problems, da SA nur unter extremen und in der Regel nicht gegebenen Rahmenbedingungen mit Sicherheit die optimale Lösung eines kombinatorischen Optimierungsproblems findet[6]. Die Idee des SA soll im folgenden erläutert werden[7].

Grundlagen des Simulated Annealing

Der Begriff des Ausglühens beschreibt in der Festkörperphysik einen Prozess, in welchem sich Teilchen, ausgehend von einem Zustand hoher Temperatur (Gas), beim Ab-

[6] Vgl. [Hajek 1988]
[7] Eine ausführliche Beschreibung findet sich in [Laarhoven 1988].

senken dieser zufällig anordnen (Flüssigkeit) und bei weiterem langsamen Abkühlen in die energetisch günstige Anordnung eines Gitters finden (Festkörper). Ein langsames Abkühlen ist gewährleistet, wenn bei jeder Temperatur ein thermisches Gleichgewicht erreicht wird[8] und damit eine Verteilung der Teilchen auf die energetisch günstigsten Positionen erfolgt. Findet die Abkühlung zu schnell statt, können Defekte 'eingefroren' werden, wodurch das Gitter gestört wird und weniger stabile Strukturen entstehen.

Um diesen Prozess auf eine Simulation zu übertragen, wird eine Monte Carlo Methode verwendet, welche eine Sequenz von Zuständen folgendermaßen erzeugt. Ausgehend von einem Zustand I_n wird durch eine minimale zufällige Veränderung ein Zustand I_{new} erreicht. Wenn die Differenz der Energieniveaus von I_n und I_{new} negativ ist, also ein energetisch günstigerer Zustand erreicht wird, so wird der Prozess von I_{new} aus fortgeführt. Wird ein energetisch höherer Zustand erreicht, so wird der Prozess mit einer Wahrscheinlichkeit von $exp\,(-\triangle E/(k_B T))$, wobei T die Temperatur und k_B die Boltzmann-Konstante ist, an dieser Stelle fortgeführt. Diese Regel wird als Metropoliskriterium bezeichnet und führt für hinreichend viele Schritte in ein thermisches Gleichgewicht[9].

Dieser Prozess, auch als Metropolis-Algorithmus bekannt, kann auch auf kombinatorische Optimierungsprobleme übertragen werden. Dafür muss eine Regel spezifiziert werden, so dass für eine gegebene Lösung i eine neue Lösung j aus deren Nachbarschaft zufällig ausgewählt werden kann. Wenn $C : S \rightarrow \mathbb{R}$ die zu minimierende Kostenfunktion ist, wobei S den Lösungsraum darstellt, so kann die Akzeptanzwahrscheinlichkeit für Lösung j wie folgt definiert werden:

$$Pr_{ij}(c) = min\left\{1, exp\left(\frac{C(j) - C(i)}{T}\right)\right\}. \tag{4.2}$$

Im folgenden wird der Algorithmus schematisch dargestellt. Als Regel, um die Erreichung des thermischen Gleichgewichts hinreichend genau festzustellen, wird eine Anzahl an Iterationen gefordert, während welcher keine Verbesserung der Lösungsqualität auftreten darf.

---

[8] Dieses Gleichgewicht wird über die Wahrscheinlichkeit, mit welcher sich ein Teilchen auf einem bestimmten Energieniveau befindet, die Boltzmann Verteilung, beschrieben. Die Formel dafür lautet $Pr\{\mathbf{E} = \mathbf{E}\} = \frac{\mathbf{1}}{\mathbf{Z(T)}}\,\mathbf{exp}\left(-\frac{\mathbf{E}}{\mathbf{k_B T}}\right)$, wobei $Z(T)$ der Normalierungsfaktor, T die Temperatur und k_B die Boltzmann-Konstante ist.

[9] Für eine ausführliche Darstellung der Metropolis Wahrscheinlichkeit sowie eine interessante Erweiterung des SA-Ansatzes siehe [Wendt 1995].

procedure Simulated Annealing
  generate initial solution $I_0 \in S$
  initialize neighborhood relation $N \subseteq S$
  set control sequence length for thermodynamic equilibrium $l \in \mathbb{N}$
  set number of transitions $K := n * l$ with $n \in \mathbb{N}$
  set temperature $T \in \mathbb{R}_+$ and cooling factor $\alpha \in]0;1[\subset \mathbb{R}$
  set diff $:= 0$
  for $i := 0$ **to** K **do**
    generate $I_{new} \in N(I_i)$
    if $random(0,1) \leq min\left\{1,\ exp\left(\frac{C(j)-C(i)}{T}\right)\right\}$
    then $I_{i+1} := I_{new}$
    else $I_{i+1} := I_i$
    diff $+=\ C(I_{i+1})\ -\ C(I_i)$
    if $i\ mod\ l = 0$ **then**
      if $diff \geq 0$ **then** $T := T * \alpha$
      diff $:= 0$
  next i
end.

Simulated Annealing für Infrastrukturprobleme

Wie in Kapitel 1 bereits beschrieben, kann das Infrastrukturproblem in zwei Teile zerlegt werden. Ein 'kürzeste Wege Problem' für ein bestehendes Netz, und das Problem, das ideale Netz zu finden. Das Problem der Ermittlung der kürzesten Wege kann sehr effizient mit dem Tripelalgorithmus gelöst werden. Eine Transition ist also der Auf- bzw. Abbau einer Infrastrukturverbindung.

Wird jede Transition akzeptiert, ergab sich bei verschiedenen Infrastrukturproblemen (20 Städte, zwischen 10 Pakete und 20 Pakete an je 5 andere Knoten) eine durchschittliche Abweichung pro Transition zwischen 0,44 und 0,46 Prozent.

4.3 Dezentrale Ansätze

4.3.1 Spieltheoretische Grundlagen

Die Spieltheorie untersucht Interdependenzen, welche bei der Auswahl einer Handlungsalternative in einem Umfeld mehrerer autonomer Akteure auftreten. Kooperative Spiele zeichnen sich durch die Möglichkeit verbindlicher Abmachungen, d.h. Kommunikation und exogener Durchsetzung aus [Holler 96], andernfalls spricht man von nicht-kooperativen Spielen. Dabei spielen Anreizprobleme und das Einbeziehen des Entscheidungskalküls der anderen, am Ausgang des Prozesses beteiligten Parteien eine wichtige Rolle.

Spieltheoretische Ansätze der Koordination

Die in der Spieltheorie benutzten Modelle helfen bei der Analyse, als auch bei der praktischen Umsetzung von Beziehungen zwischen Mensch und Softwareagent, als auch bei der Verhandlung bzw. Allokation von Ressourcen unter mehreren Softwareagenten. Die gebräuchlichste Darstellung von Spielen ist die Normalform, bei welcher es neben einer Menge von Agenten eine Menge von Strategien gibt, welche den Agenten zugeordnet sind. Zusätzlich sind die Resultate für jede Strategiekombination in einer Matrix angegeben, welche den Agenten bekannt ist. Ein Hauptanliegen der Spieltheorie ist die Analyse von strategischen Entscheidungen, bei welchen das Resultat von den Handlungen anderer Personen (-gruppen) abhängig ist. Um dieses Ziel zu erreichen, wird eine Entscheidungssituation zunächst formal dargestellt. Eine solche Darstellung wird als Spielsituation bezeichnet, folgende Anforderungen bestehen an die formale Darstellung als Spiel:

- An der Spielsituation nehmen n Personen teil, welche ferner als Spieler bezeichnet werden.
- Jeder Spieler wählt seine Strategie aus einer Menge von verschiedenen Strategien.
- Die Auszahlung, welche sich aus einer bestimmten Strategiekombination ergibt, ist ex ante bekannt.
- Alle Spieler verfügen über vollständige Kenntnis der Regeln des Spiels.
- Strategische Überlegungen über die Entscheidung der anderen Spieler werden von jedem Spieler durchgeführt.

Offen ist die Frage, ob es die Möglichkeit für Absprachen gibt, sowie ob eine Institution, durch welche sich ein Spieler glaubhaft an seine Versprechen binden kann, existiert. Wenn eine solche Möglichkeit zur Durchsetzung von Absprachen besteht, spricht man von kooperativen Spielen. Besteht keine Möglichkeit zur Absprache oder nur die Möglichkeit zur Absprache, jedoch kein Zwang, sich daran zu halten, würden sich rationale Agenten nicht an ihre Absprache halten, wenn sie daraus einen Vorteil ziehen könnten. Deshalb werden auch diese Spiele als nicht-kooperativ bezeichnet.

4.3.2 Lösungskonzepte

Basierend auf einer formalen Darstellung können Anforderungen an eine Lösung bzw. Konzepte für die Lösung von Konflikten formuliert werden.

Dominanz

Als einfachste Auswahlregeln für eine Strategie gilt es herauszufinden, ob es eine alternative Strategie gibt, welche einer anderen in jedem möglichen zukünftigen Zustand der Welt überlegen ist. Gibt es eine solche, kann die erste Strategie aus der Entscheidungsmatrix entfernt werden.

Nash-Gleichgewicht

Unter einem Nash-Gleichgewicht versteht man eine Situation, in der kein Spieler seinen Nutzen durch abweichendes Verhalten erhöhen kann, sofern die anderen Spieler bei ihrem Verhalten bleiben [Fees 1997].

4.3.3 Strategien zur Verteilung von Koalitionsgewinnen

Liegen kooperative Spiele mit transferierbarem Nutzen vor, spielen neben Konflikten auch Koalitionen eine entscheidende Rolle. Wird diese Möglichkeit in einem Multi Agenten System abgebildet, besteht ein Hauptproblem in der Verteilung des Gewinns unter den Koalitionspartnern [Owen 1982]. Beispielsweise können Spieler Strategien wählen, welche zwar individuell nicht rational wären, aber den Nutzen eines anderen Spielers erhöht. Der dadurch erzielte höhere Gewinn fließt dann zum Teil an den Koalitionspartner zurück. Das Hauptproblem, welches beim Entwurf eines Mechanismus für Koalitionsbildung in Spielen auftritt, ist die Verteilung des Gewinns unter den Koalitionspartnern. Im folgenden werden zwei Konzepte zur Allokation von Koalitionsgewinnen dargestellt:

Anforderungen

Für alle eine Lösung darstellenden Verteilungen gelten zwei Einschränkungen, mit welchen die Menge aller möglichen Verteilungen reduziert werden kann. Zum einen individuelle Rationalität, welche besagt, dass ein Spieler nicht an einer Koalition teilnimmt, wenn er nicht mindestens so viel erhält, wie er ohne Teilnahme an der Koalition erhalten kann, und zum anderen Pareto-Effizienz. Kann kein Teilnehmer besser gestellt werden, ohne den Nutzen eines anderen Teilnehmers zu reduzieren, spricht man von Pareto-Effizienz. Aus dieser Bedingung folgt, dass der gesamte Gewinn, welchen eine Koalition erwirtschaftet, auf die Teilnehmer verteilt wird. Diese beiden Einschränkungen reduzieren den Möglichkeitsraum jedoch noch nicht hinreichend.

Kernel eines Spiels

Das Prinzip der Rationalität kann nicht nur auf Individuen, sondern auch auf Teilgruppen der Koalition, also zur kollektiven Rationalität erweitert werden. Es muss gelten, dass es keine Gruppe innerhalb der Koalition geben darf, welche für sich eine größere

Auszahlung erzielen könnte, als diese Teilgruppe innerhalb der Koalition erzielt. Unglücklicherweise gibt es oft keine Lösung, die dieses Kriterium erfüllt, weshalb der Kernel eines Spiels nicht als genereller Lösungsweg benutzt werden kann.

Shapley Value

Für den Fall, dass ein Spiel keinen Kernel besitzt, wird es für jede Koalition eine Teilmenge von Spielern geben, welche mit der Verteilung nicht zufrieden sind, da sie zusammen mehr Nutzen erhalten könnten, wenn sie eine eigene Koalition formen. Ein Vorschlag zur Verteilung des Nutzens innerhalb einer Koalition ist der Shapley Value, welches natürlich keine Stabilität gewährleisten kann, aber eine Basis für eine faire Gewinnverteilung bietet. Die Ermittlung der Gewinnanteile ergibt sich beim Shapley Value aus dem Nutzen, welche eine Koalition aus dem Hinzukommen eines Mitgliedes erhalten würde. Da dieser davon abhängt, wer schon in der Koalition enthalten ist, müssen alle Permutationen, in welcher die Mitglieder einer Koalition angeordnet werden können, berücksichtigt werden. Das Shapley Value kann für jede Koalition errechnet werden, ist eindeutig und erfüllt immer die Forderung nach individueller Rationalität und Pareto-Effizienz.

Allgemein

Für Spiele, welche in Normalform formuliert werden können, sind die formulierten Anforderungen, also Dominanz und Nash Gleichgewicht, immer anwendbar, wenngleich ein Nash Gleichgewicht nicht immer existieren muss. Kernel und Shapley Value sind nicht anwendbar in Spielen, in welchen der Nutzen einer Koalition auch vom Verhalten der nicht an der Koalition teilnehmenden Spieler abhängt, da es sich nicht um Lösungskonzepte für Spielstrategien sondern um Konzepte zur Nutzenverteilung innerhalb von Koalitionen handelt.

4.3.4 Ergebnismatrix für das Infrastrukturproblem

Gegeben sei ein 3 Knoten Netzwerk, welches zunächst aus Perspektive von Knoten 1 untersucht wird. Knoten 1 hat 1 Paket an Knoten 2 und 1 Paket an Knoten 3 auszuliefern. Die Transportkosten auf einer Kante betragen 5 GE pro Einheit, wird jedoch in eine Kante ein fester Betrag von 6 GE investiert, so sinken die Transportkosten auf 1 GE pro Mengeneinheit. Die Transportmengen zwischen Knoten 2 und Knoten 3 seien vorerst nicht bekannt. Eine graphische Darstellung dieses Netzwerkes findet sich in Abbildung 4.3.

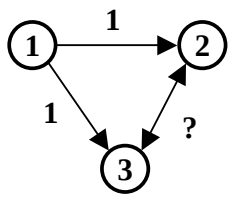

Abbildung 4.3: Einfaches Netz

In Tabelle 4.1 werden die Handlungsalternativen für Knoten 1 dargestellt. Dabei gibt P_2 bzw. P_3 den Preis an, welchen Knoten 2/3 für den Transport einer Mengeneinheit über 'seine' Infrastruktur verlangt.

vorhandene Infrastruktur	Infrastruktur nach			
	keine	Knoten 2	Knoten 3	Kn. 2 und 3
$2 \rightarrow 3$	10	$8+P_2$	12	14
$2 \leftarrow 3$	10	12	$8+P_3$	14
$2 \leftrightarrow 3$	10	$8+P_2$	$8+P_3$	14

Tabelle 4.1: Kostenmatrix

Wie sich an diesem einfachen Beispiel ablesen lässt, gibt es keine dominante Strategie, wenn die Preise kleiner als 2 GE pro Transporteinheit sind. Angenommen, der Transport zwischen Knoten 2 und Knoten 3 kostet ebenfalls 1 GE, so liegt die sinnvolle Preisspanne, welche für den Transport verlangt werden kann zwischen einer GE und zwei Geldeinheiten.

Zudem verdeutlicht dieses Beispiel auch den Anreiz zu Absprachen. Beispielsweise wäre es für diese 3 Akteure sinnvoll, sich abzusprechen, ob Infrastruktur von 1 nach 2 nach 3 nach 1 oder von 1 nach 3 nach 2 nach 1 eingerichtet wird. Beide Varianten stellen effiziente Lösungen dar, und ein Verlassen dieser Lösung ist nur unter erheblichem Koordinationsaufwand möglich. Dieses Phänomen bezeichnet man auch als Lock-In-Effekt.

4.3.5 Statischer Optimierungsansatz

Als erster Ansatz zur dezentralen Optimierung soll hier ein einfaches Verfahren gewählt werden, bei welchem die Akteure den Versuch unternehmen die abzuwickelnde Transportmenge einzuschätzen. Ausgehend von diesem Schätzwert wird die Entscheidung getroffen, ob eine Investition in Infrastruktur lohnt. Nachdem diese Entscheidung von allen Akteuren getroffen ist, soll der gesamte Nutzen berechnet werden. Im folgenden wird der Ansatz vorgestellt:

Zunächst werden Annahmen über das Wissen, über welches ein jeder Akteur verfügt, getroffen. Jeder Akteur kennt

- die **Positionen** aller anderen Akteure und
- die **Transportmengen** aller Akteure untereinander.

Da die Entscheidung jedoch simultan getroffen werden muss, schätzt jeder Agent die Menge der Einheiten, welche über die einzelnen Kanten geleitet werden. Diese Menge setzt sich zusammen aus dem eigenen Transportbedarf, welcher direkt oder indirekt sein kann, sowie die geschätzte Transportnachfrage anderer Knoten. An den folgenden zwei Beispielen in Abbildung 4.4 sollen kurz die Kalkulation von Akteur A erläutert werden.

Die Variable x_{AC} gibt den direkten Transportbedarf, also die Menge, welche von Akteur A zu Akteur C transportiert werden soll, an. Die Einsparung, welche durch den Bau einer Infrastrukturverbindung auf der Strecke von A nach C erfolgen kann, ist

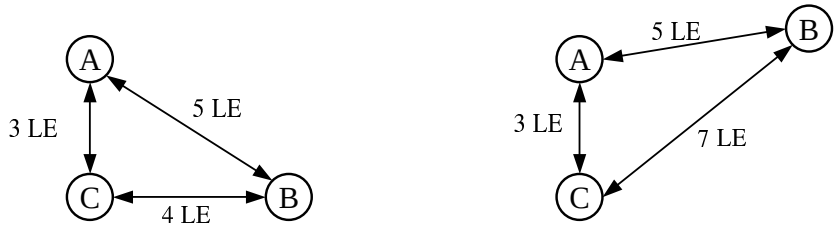

Abbildung 4.4: A abseits von B und C (links), A zwischen B und C (rechts)

also zunächst $x_{AC}(k^N_{AC} - K^I_{AC})$, wobei k^N_{AC} die direkten Kosten angibt, welche pro Einheit anfallen, wenn keine ausgebaute Verbindung besteht bzw. k^I_{AC} die Kosten pro Einheit bei Bestehen einer Infrastrukturverbindung.

Die Wahrscheinlichkeit für den Transport von Akteur A zu B über C ist bedingt von den Positionen der Akteure bzw. den sich daraus ergebenden Transportkosten k^N_{XY} auf den die Akteure verbindenden Kanten.[10] Bei der linken Abbildung besteht offensichtlich eine größere Chance, dass ein Transport über A stattfindet, als bei einer Anordnung wie in der rechten Abbildung. Dieser Zusammenhang soll mit dem Parameter p_{ACB} erfasst werden, welcher wie folgt errechnet wird:

$$p_{ACB} = \left(\frac{k^N_{AB}}{k^N_{AC} + k^N_{CB}} \right)^2 \tag{4.3}$$

Hinzu kommt noch die Wahrscheinlichkeit für eine Transportnachfrage anderer Akteure, welche analog zur indirekten Transportnachfrage berechnet wird.

$$p_{BAC} = \left(\frac{k^N_{BC}}{k^N_{BA} + k^N_{AC}} \right)^2 \tag{4.4}$$

Multipliziert man die jeweilige Anzahl der Pakete mit der entsprechenden potentiellen Kosteneinsparung und der Eintrittswahrscheinlichkeit, so erhält man die Kostenersparnis, welche durch die Einrichtung einer Infrastrukturverbindung realisiert werden könnte. Demgegenüber stehen die Kosten für die Einrichtung, woraus sich folgende Bedingung ergibt, dass die Infrastruktur $\overline{AC}$ eingerichtet wird, wenn:

$$K_{AC} \leq \left(x_{AC} + \sum_{kn \in Knoten} p_{knAC}\ x_{knC} + p_{ACkn}\ x_{Akn} \right) (k^N_{AC} - k^I_{AC}) \tag{4.5}$$

Ergebnisse

Simulationsläufe des statischen Modells haben die, in den folgenden Tabellen dargestellten, Ergebnisse erbracht. Zum Vergleich wurden die selben Infrastrukturnetzwerkprobleme auch mit Simulated Annealing optimiert. Eine graphische Veranschaulichung der Ergebnisse erfolgt in Abbildung 4.5.

Die Kosteneinsparungen, welche durch die Einrichtung der Infrastruktur erreicht werden, steigen zwar absolut mit wachsender Knotenzahl, gehen jedoch im Verhältnis zu den Gesamtkosten zurück.

[10] Werden im Modell Kosten verwendet, welche proportional zur euklidischen Distanz sind, so können die Entfernungsangaben auch mit den Kosten ersetzt werden.

Knoten	10	15	20	25
Kosten	40931	50088	52456	65300
optimal	22387.75	31481.5	30065.75	41412.25
Einsparung	45.3%	37.1%	42.7%	36.6%
statisch	33418.5	41896.75	41149.75	52225
Einsparung	18.4%	16.4%	21.6%	20.0%

Knoten	30	35	40	45
Kosten	61947	67698	71812	74285
optimal	38763.75	45271	45548.25	47972.5
Einsparung	37.4%	33.1%	36.6%	35.4%
statisch	51870.75	59123	59223.5	62997.5
Einsparung	16.3%	12.7%	17.5%	15.2%

Tabelle 4.2: Vergleich bei verschiedenen Knotenzahlen

4.3.6 Dynamischer Optimierungsansatz

Im Gegensatz zum statischen Ansatz soll der dynamische Ansatz die Möglichkeit bieten, Entscheidungen zu evaluieren, d.h. es kann eine Infrastruktur gebaut werden, um zu testen, wieviel zusätzlicher Transport durch das preiswertere Angebot der Dienstleistung stattfindet. Da die Berücksichtigung von Transportzeit in diesem einfachen Modell zunächst keine Rolle spielt, könnte dieser Aufbau auch nur ein Test sein, bei welchem ein Preis benutzt wird, welcher erst durch den Aufbau einer Infrastruktur langfristig zu realisieren wäre.

Des weiteren soll das Modell dahingehend optimistisch sein, dass auf jenen Strecken, auf welchen keine Infrastrukturinvestitionen stattfinden, keine Gewinnmaximierung betrieben wird, sondern Transport zum minimalen Preis, also den Selbstkosten, angeboten wird. Im folgenden wird der Preisbildungsprozeß für Transportdienstleistungen über Verbindungen mit Infrastruktur untersucht. Alle im folgenden verwendeten Variablen gelten für einen Akteur:

> x_s *sei der eigene Transport, welcher über eine bestimmte Verbindung transportiert wird,*
>
> x_v *sei der nicht eigene Transport, welcher über diese Verbindung transportiert wird,*
>
> k_N *seien die minimalen Kosten, welche pro Einheit anfallen, wenn keine Infrastrukturverbindung besteht, dabei muss es sich keineswegs um die direkte Verbindung handeln.*
>
> k_I *seien die Kosten, welche pro Einheit anfallen, wenn eine Infrastrukturverbindung besteht,*
>
> p_G *sei der Preis, welcher pro Einheit für den Transport fremder Einheiten verlangt wird,*

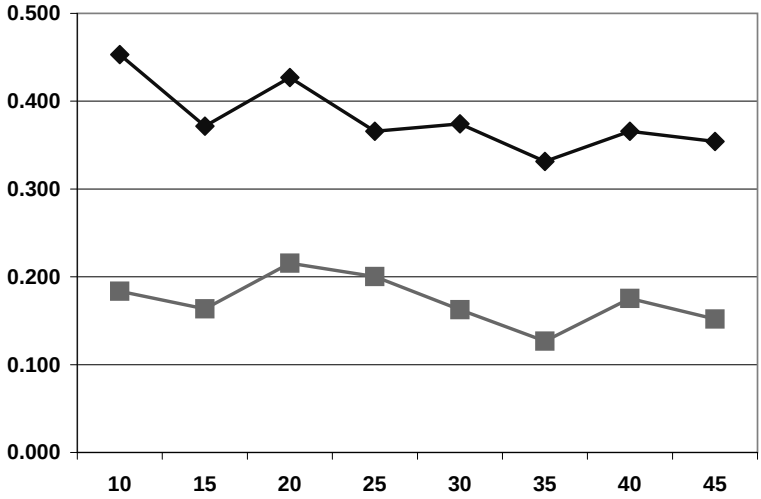

Abbildung 4.5: Kosteneinsparungen nach Knotenzahl

K_I seien die Kosten, welche für die Einrichtung der Infrastrukturverbindung anfallen.

Wenn gilt $(k_N - k_I)x_s \geq K_I$ wird die Infrastruktur in jedem Fall eingerichtet. Ein Entscheidungsproblem tritt auf, wenn gilt, dass die Einsparung $(p_N - p_I)x_s$ kleiner als die Kosten ist, jedoch durch den Verkauf von Transportleistungen doch wieder eine positive Bilanz erzielt werden könnte. Ist der Angebotspreis höher als die Transportkosten ohne Infrastruktur, gilt also $p_G > k_N$, ergibt sich folgende Bedingung:

$$k_N x_s + k_N x_v - p_G x_v > K_I + k_I x_s + k_I x_v - p_G x_v \tag{4.6}$$

$$\Rightarrow \quad x_v > \frac{K_I}{p_N - p_I} - x_s. \tag{4.7}$$

Da die Folge einer Erhöhung des Angebotspreises keine Steigerung der Menge x_v sein kann, wird die Erhöhung des Preises über die Kosten k_N hinaus also nicht zu einer Erfüllung der für die Investition in eine Infrastruktur notwendigen Bedingung führen. Der Preis muss zwischen den eigenen Kosten für den Transport mit bzw. ohne vorhandener Infrastrukturverbindung liegen. Aus der Bedingung, dass es zu einer Kostenersparnis kommen muss (also $k_N x_s > K_I + k_I x_s + k_I x_v - p_G x_v$), lässt sich eine Angebotskurve ableiten (Abbildung 4.6).

Der Schnittpunkt mit der Ordinate zeigt die Menge, welche mindestens verkauft werden muss, damit die Möglichkeit besteht, dass die Infrastruktur selbsttragend finanziert werden kann. Ceteris paribus müsste jeder Preis evaluiert werden, um festzustellen, mit welchem Preis der Gewinn maximiert werden kann. Da es jedoch weder möglich ist, alle Preise zu evaluieren, noch der Preis die einzige beeinflussende Größe ist, muss eine Strategie gefunden werden, welche auch mit begrenzter Zeit zu einem guten Ergebnis führt.

Ein ähnliches Problem stellt sich auch auf jedem Markt, auf welchem Güter geund verkauft werden. Es können lediglich Punkte auf der Angebotskurve ermittelt werden, woraus dann mit einer Vielzahl von Werkzeugen optimale

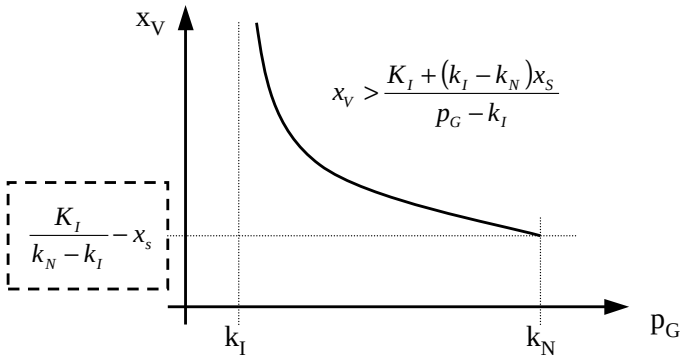

Abbildung 4.6: Angebotskurve

Züge ermittelt werden können, um in möglichst hohe Gewinne zu generieren. Der hier gewählte Ansatz ist jedoch ein anderer. Zunächst einmal verfolgen die Akteure nur Satisfizierungsziele, d.h. wenn eine Kostendeckung erreicht ist, wird der Evaluati-

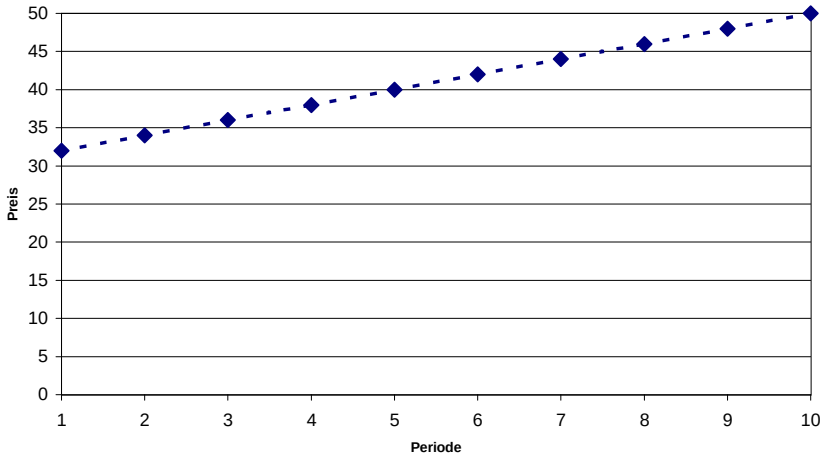

Abbildung 4.7: Reale Angebotskurve über die Zeit

Die Genauigkeit, mit welcher ein solcher Preis ermittelt wird, und auch die Vorgehensweise des Akteurs werden ex ante dadurch vorgegeben, dass der Akteur festlegt wieviele Versuche er unternimmt, durch einen neuen Transportpreis mindestens Kostendeckung für eine eingerichtete Infrastrukturverbindung zu erreichen. Die einfachste Angebotspreisfunktion ist die lineare Verbindung zwischen dem Selbstkostenpreis und dem Preis ohne Infrastruktur[11] (Abbildung 4.7). Eine progressive Angebotspreisfunktionen könnte beispielsweise durch die quadratische Funktion $p(t) = \frac{1-t}{T}^2$ abgebildet werden. Durch freies Setzen des Exponenten n in der Funktion $p(t) = \frac{1-t}{T}^n$ können sowohl progressive $(n > 1)$ als auch degressive $(n < 1)$ Angebotsfunktion

[11] Auf der Ordinate wird die Zeit bzw. die Anzahl der Perioden in welchen bereits ein Angebot stattfand angegeben.

abgebildet werden. Dabei gibt T die Anzahl der Perioden an, über welche sich die Angebotsfunktion maximal erstreckt und t die aktuelle Periode.

Ergebnisse

Einige beispielhafte Simulationen sollen zeigen, wie sich die Einsparungen durch den Aufbau von Infrastruktur bei zunehmender durchschnittlicher Transportmenge verhält. Die Standardabweichung für alle Transportmengen ist 1 und der Erwartungswert wird ausgehend vom Wert 1 um je 1 Einheit erhöht.

Pakete	1	2	3	4	5
Kosten	23316	37640	55482	74996	88832
SA	21953.25	30515.25	37311.00	40729.00	43291.50
Einsparung	5.84%	18.93%	32.75%	45.69%	51.27%
dynamisch	21756.50	30789.25	39146.50	42678.00	45404.00
	22146.00	30806.00	39784.00	42036.00	44593.00
	22071.50	31196.50	39651.50	43312.25	44925.00
	21911.25	32649.00	39331.75	42901.50	44327.00
	22279.25	31146.75	38380.75	43659.50	44209.50
Ø	22032.90	31317.50	39258.90	42917.45	44691.70
Einsparung	5.5%	16.8%	29.2%	42.8%	49.7%
statisch	23114.25	36439.00	48444.25	55046.50	65950.25
Einsparung	0.9%	3.2%	12.7%	26.6%	25.8%

Pakete	6	7	8	9	10
Kosten	110238	133090	152502	180603	198405
SA	49836.00	62372.50	65026.25	73994.00	73055.75
Einsparung	54.79%	53.14%	57.36%	59.03%	63.18%
dynamisch	53214.75	63411.00	67402.75	76231.75	81509.50
	54570.50	64467.00	65427.50	79185.75	80935.50
	50650.50	64185.00	66693.50	78737.25	81317.50
	52004.25	62757.75	66074.25	77100.00	80565.00
	53407.25	64264.00	65670.00	77633.50	81221.75
Ø	52769.45	63816.95	66253.60	77777.65	81109.85
Einsparung	52.1%	52.0%	56.6%	56.9%	59.1%
statisch	73604.50	80845.00	86193.00	95245.00	99098.75
Einsparung	33.2%	39.3%	43.5%	47.3%	50.1%

Pakete	11	12	15	20	25	50
Kosten	209699	221312	301189	360837	499036	973022
SA	84494.25	83653.00	108234.75	119902	162453.00	279699.75
Einsp.	59.71%	62.20%	64.06%	66.77%	67.45%	71.25%
dyn.	85767.25	84384.50	109432.50	120014.50	162448.00	279669.25
	85088.50	86155.25	108834.00	120402.50	162531.75	279669.25
	84802.75	85732.75	108644.50	120824.00	162488.00	279669.25
	85837.00	86036.00	109037.50	120120.25	162531.75	280684.75
	85421.25	84244.25	109432.50	119877	162448	279669.25
Ø	85383.35	85310.55	109076.20	120247.65	162489.50	279872.35
Einsp.	59.3%	61.5%	63.8%	66.7%	67.4%	71.2%
stat.	100332.25	101525.50	125664.75	135206.75	174414.00	291930.00
Einsp.	52.2%	54.1%	58.3%	62.5%	65.0%	70.0%

Tabelle 4.3: Vergleich bei verschiedenen Transportmengen

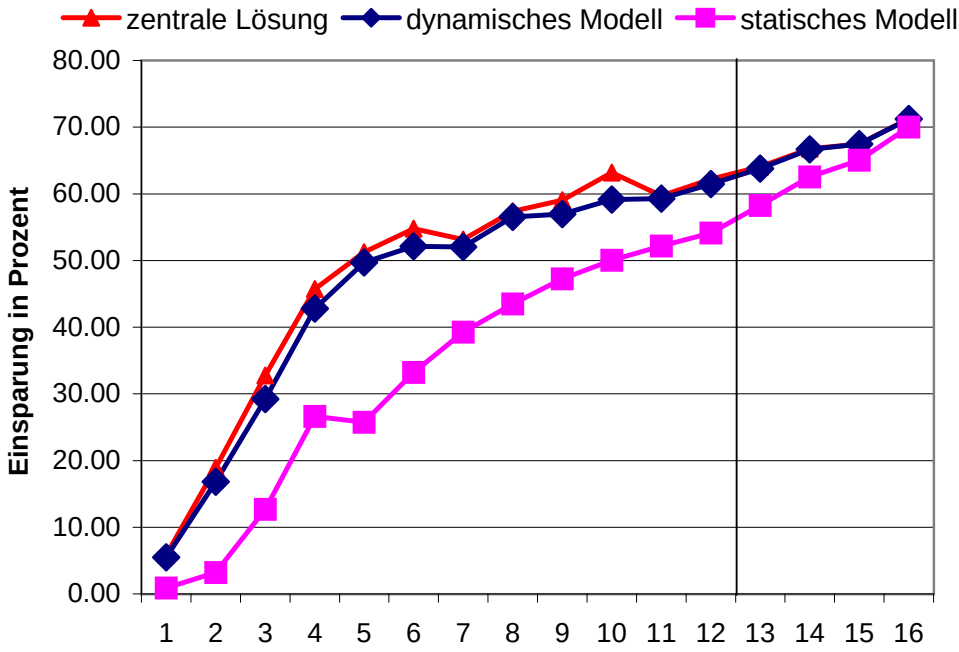

Abbildung 4.8: Graphischer Vergleich bei verschiedenen Transportmengen

Wie man sieht, steigt die relative Ersparnis mit zunehmender Transportmenge bei beiden Verfahren. Die Ersparnis der am Anfang sehr schlecht abschneidenden statischen Optimierung wächst jedoch schneller als jene beim dynamischen Modell. Beide Graphen konvergieren bei zunehmender Transportmenge gegen $1 - \frac{Transportkosten\ mit\ Infrastruktur}{Transportkosten\ ohne\ Infrastruktur}$ in diesem Fall $1 - \frac{25}{100} = 75\%$.

In der nächsten Anordung wurden die Transportmenge konstant gehalten. Variiert wurde nur die mittlere Abweichung von der erwarteten Transportmenge. Die Ergebnisse sind in Tabelle 4.4 zu finden.

Pakete	12	12	12	12	12	12
Abweichung	0	1	2	3	4	5
Kosten	215052	221312	180420	185800	172530	230406
dyn.	75899.50	84244.25	67757.00	71236.25	65956.25	85950.75
Einsp.	65%	62%	62%	62%	62%	63%
statisch	98570.50	101525.50	84487.50	84370.00	79209.75	99336.00
Einsparung	54%	54%	53%	55%	54%	57%
SA	75803.00	83653.00	66894.25	71021.25	64944.00	84876.50
Einsparung	65%	62%	63%	62%	62%	63%

Tabelle 4.4: Vergleich bei gleichen Transportmengen

Die Berechnung der Abweichung von der erwarteten Transportmenge erfolgt über eine Normalverteilung, wobei in der Tabelle die Standardabweichung angegeben ist. Aus den ermittelten Daten lässt sich die Aussage ableiten, dass die Verteilung der Transportmengen keinen Einfluss auf die Einsparungen haben. Eine graphische Veranschaulichung findet sich in Abbildung 4.9.

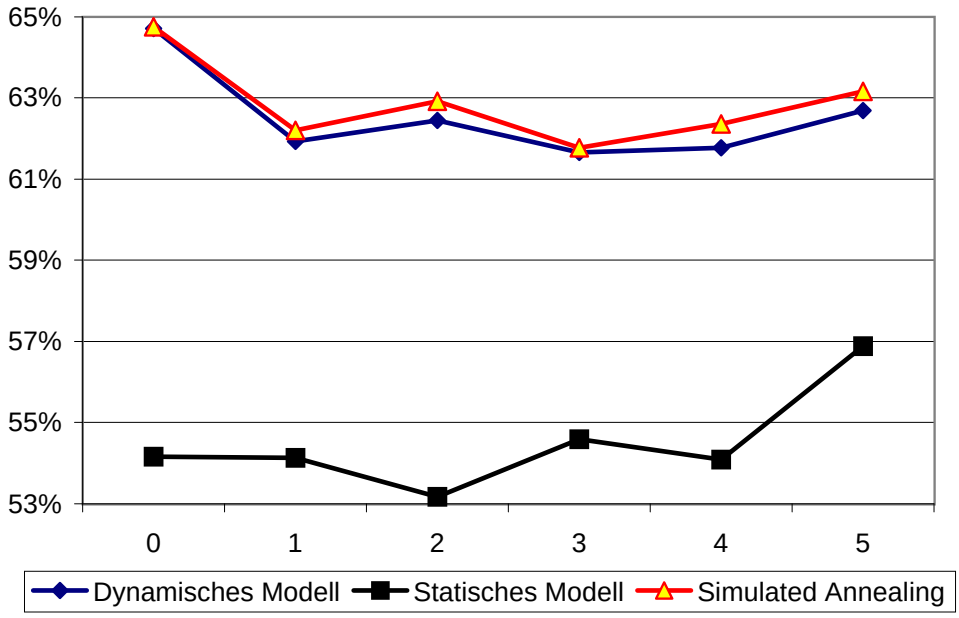

Abbildung 4.9: Graphischer Vergleich bei gleichen Transportmengen

4.4 Zusammenfassung

Es wurden vier verschiedene Ansätze zur Lösung des Infrastrukturproblems implementiert und miteinander verglichen. Wie erwartet waren die erreichten Einsparungen mit dem zentralen Lösungsansatz am höchsten, gefolgt von dem reinen Verbesserungsverfahren. Der Abstand zwischen dem als nächstes folgendem dynamischen Schätzer und der mit einer zentralen Lösung erreichten Einsparung war nicht so groß wie erwartet. Der Grund dafür liegt sicher in der Struktur des Problem, worauf auch das gute Abschneiden der reinen Verbesserungsverfahrens hindeutet. Wie erwartet schnitt der einfache statische Schätzer am schlechtesten ab.

Kapitel 5

Das Programm

Im Rahmen dieser Diplomarbeit entstand ein Programm, welches die Infrastrukturentscheidung entsprechend dieses Modelles simuliert bzw. errechnet. Die verwendete Programmiersprache ist JAVA^{TM} 1.1[1], zur Unterstützung wurde das Entwicklungswerkzeug IBM Visual Age 3.0 verwendet[2]. Neben der Implementierung der Modelle wurde eine graphische Benutzerschnittstelle, welche unter der Adresse http://www.wiwi.uni-frankfurt.de/∼stockhei/network/index.html zur Verfügung steht, entwickelt. Eine kurze Beschreibung sowie Veranschaulichung findet sich in den nächsten Kapiteln.

5.1 Das Simulationstool

Aufgrund der Verwendung von JAVA^{TM} kann das Programm auf jeder Rechnerplattform, auf welcher eine entsprechende Virtual Machine zur Verfügung steht, benutzt werden. Der Aufruf erfolgt mit `java Network.Workbench`. Werden keine weiteren Parameter angegeben, so wird eine kurze Hilfe ausgegeben, in welcher die verfügbaren Parameter erläutert werden[3].

```
Networksimulation (c) Tim Stockheim 2000 Germany
------------------------------------------------
Available options :
/g or
/graphic                        - show GUI
/l or /load name.map                  - load a prebuilt map
/s or /save name.map                  - build and save a map
/o or /output [Name of output file] - save output to a file
/m or /method \{d/sa/cosa/enum\}      - set the method to ...
/w or /width number     - set width of map
/h or /height number - set height of map
/Infrastructure float   - set relative infrastructure cost
```

[1] Näheres zu dieser Programmiersprache unter http://www.javasoft.com.

[2] Diese Entwicklungsumgebung ist in der Einsteigerversion (Entry Version) frei verfügbar. Ein geeigneter Einstiegspunkt findet sich unter http://www.software.ibm.com/vadd.

[3] Unter neueren Microsoft Windows^{TM} Umgebungen kann auch der Befehl `jview` verwendt werden.

```
/Transport float        - set relative transport cost
(without infrastructure)
/reducedTransport float - set relative transport cost
(with infrastructure)
If output file is not specified the program will write to
the standard output. This programm will simulate a network
where each player has the choice between a infrastructure
connection (high fixed cost) and volume dependent cost
(high variable cost). For a clearer understanding read the
manual ;-)}
```

Die Standardeinstellungen sind in der Datei `market.properties` angegeben. Dadurch ist gewährleistet, dass einfache Einstellungen auch ohne Änderungen am Sourcecode angepasst werden können. Die Ausgabe von aktuellen Daten während der Simulation in eine Datei ist mit der Option `/output` oder `/o` möglich, aber noch nicht frei einstellbar.

Dieser Teil des Programms ist im Paket Network abgelegt, die Klassen und Methoden sind im Anhang oder über die oben angegebene URL und den Verweis javadoc einsehbar.

5.2 Die graphische Benutzeroberfläche

Neben der Erreichbarkeit über die Webpage kann man die graphische Oberfläche auch mittels der Option `/g` oder `/graphic` aufrufen.

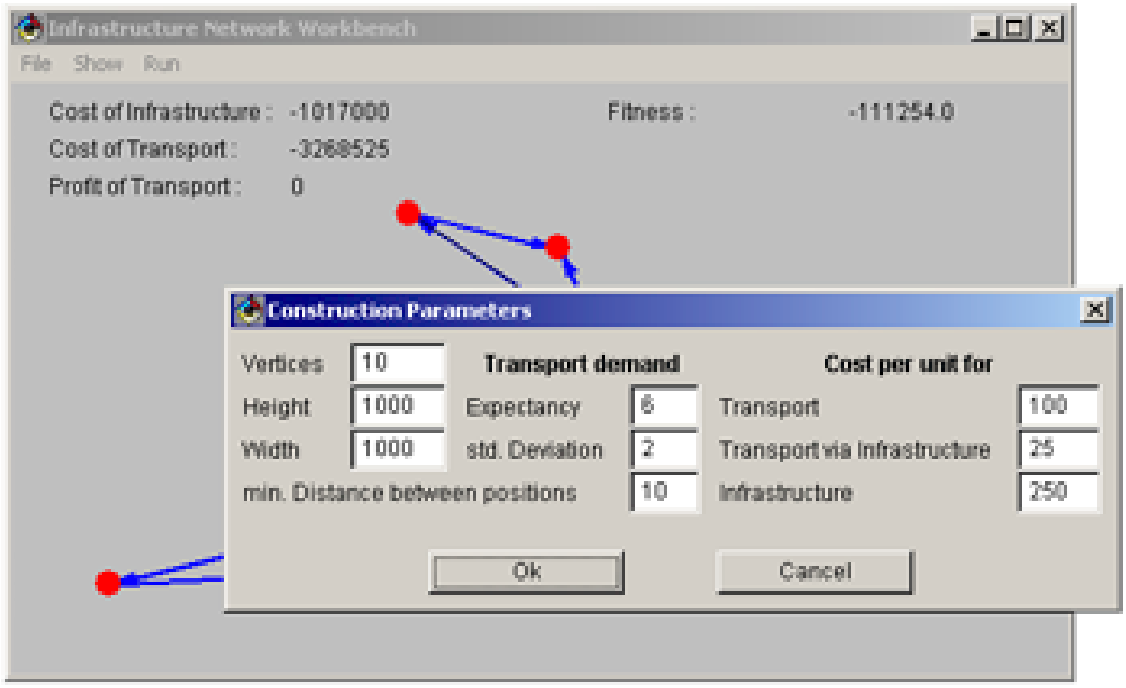

Abbildung 5.1: Graphische Benutzerschnittstelle

Wie zu sehen ist, können die Optionen für das Erzeugen eines Netzwerkes auch hier eingegeben werden. Es ist zudem möglich, vorbereitete Beispieldateien zu importieren. Die Funktion zur Speicherung von Netzwerken funktioniert nur, wenn das Programm als Applikation, also nicht aus dem Internet aufgerufen wird.

Startet man eine Simulation, so werden die dafür notwendigen Parameter über seperate Menüs abgefragt. Eine laufende Simulation kann nach Belieben unterbrochen werden. Ein interaktives Eingreifen ist jedoch nicht möglich.

Das Programm ist durch die graphische Oberfläche sowohl für das Veranschaulichen der Ergebnisse, als auch für die Simulation auf einzelnen oder mehreren Rechnern geeignet. Eine Erweiterung um viele Aspekte, welche teilweise auch aus dieser Arbeit hervorgehen, ist angestrebt.

Die Berechnungen aus den vorangegangenen Kapiteln wurden zum Großteil im PC-Pool der Uni Frankfurt und vielen anderen, in diesem Umfeld zur Verfügung stehenden Rechnern, erzielt.

Kapitel 6

Zusammenfassung

Der Ansatz des Infrastruktur-Netzwerkes bietet eine Möglichkeit, das Entstehen von Netzwerken und die Abhängigkeit von bestimmten Einflüssen auf die Struktur nachzuvollziehen. Trotz der komplexen Struktur scheinen dezentrale Ansätze ihre Berechtigung zu haben. Das deutlich schlechtere Abschneiden des statischen Ansatzes zeigt, wie wichtig Kommunikation zwischen den Akteuren ist. Da die umfangreiche Kommunikation trotz des recht einfachen Entscheidungsverhaltens gute Ergebnisse erzielt, wäre zu untersuchen, wie weit man bei Einschränkung der Kommunikation durch bessere Entscheidungsregeln ähnlich gute Ergebnisse erzielen kann. Die Frage, wie gut ein statischer Ansatz sein kann, ist sicher nicht endgültig beantwortet. Ob jedoch rein statische Ansätze ein realistisches Szenario darstellen und untersucht werden sollten, ist eine Frage, die hier nicht beantwortet werden kann.

Das sehr gute Abschneiden von reinen Verbesserungsverfahren lässt auf eine relativ homogene Nachbarschaftsstruktur schließen. Des weiteren war zu beobachten, dass in größeren Netzen der relative Verlust durch Fehlentscheidungen tendenziell geringer ausfällt. Ein Ursache könnte sein, dass die zunehmende Dichte von Netzwerken zu Bündelungseffekten bezüglich der Transportleistungen führen.

In einem nächsten Schritt wäre es interessant, intelligentere Entscheidungsregeln von Agenten in einem solchen Umfeld zu testen. Durch geeigneten Einsatz dieser 'internen Intelligenz' bei eingeschränkter Kommunikation, ist es vielleicht möglich, zu schnelleren und effizienteren Lösungen zu finden, als es mit diesem Ansatz möglich war.

Anhang A

Source Code

```
package Network;

import java.lang.Math;
import java.util.Vector;
import java.awt.Point;

public class Agent {
    private Market market;

    private int number;
    private int networkSize;
    public Point position;
    protected Vector network        = new Vector();
    private Vector secondbestFlows  = new Vector();
    private Vector flows            = new Vector();
    protected Vector newFlows       = new Vector();
    protected Vector soldFlows      = new Vector();
    protected Vector purchasedFlows = new Vector();

    public double temperature = 0;
    private long oldProfit = 0;

/*****************************************************************************
 * Constructor of Agent, where
 *    - m ... is the market where the agent can offer his services
 *    - n ... identifies the agent at the market
 *    - x and y are the coordinates of the agent
 *****************************************************************************/
public Agent ( int n, int x, int y, Market m) {
    this.market = m;
    this.number = n;
    this.position = new Point( x, y);
}
/*****************************************************************************
 * Constructor of Agent, where
 *    - m ... is the market where the agent can offer his services
 *    - n ... identifies the agent at the market
 *    - pos are the coordinates of the agent given as java.awt.Point
 *****************************************************************************/
public Agent ( int n, java.awt.Point pos, Market m) {
    this.number = n;
    this.position = pos;
    this.market = m;
}
private final boolean accept (double deltaFitness, double temp) {
   return ( (deltaFitness > 0) ||
            (Math.random() < (Math.exp(Math.max((deltaFitness/temp),-100)))));
}
/*****************************************************************************
 * Every AgentModel represents a market participant to this Agent. So after
 * all agents are created every agent gets a representation of each other via
 * this method.
 *****************************************************************************/
public void addAgent ( AgentModel a ) {
    long dist = Vec.distance ( this.position, a.position);
    a.resetParameters ( dist, market);
    network.addElement ( a );
    networkSize = network.size();
}
public boolean checkDiff() {
    long profit = getProfit();
    if ( temperature>market.stopTemperature) {
        if ( oldProfit>profit) temperature*=market.cRate;
        oldProfit = profit;
        return false;
    }
```

```
    oldProfit = profit;
    return true;
}
/*************************************************************************
 * The agent tries to find out if there is a better route to deliver his
 * packet. If so he or she will use this connection
 *************************************************************************/

public boolean checkForBetterPrices () {
    boolean change = false;
    for ( int i=0; i<networkSize; i++) if (i!=number) {
        long oldPrice = ((Flow)flows.elementAt(i)).getCost();
        long min = ((AgentModel)network.elementAt(i)).getTransportCost(1);
        ((Flow)newFlows.elementAt(i)).setCost(min);
        ((Flow)newFlows.elementAt(i)).setTarget(i);

        // checkout the best route to the target i
        for ( int j=0; j<networkSize; j++) if ( (j!=number) && (j!=i) )
        if ( min > ( ((AgentModel)network.elementAt(j)).getTransportCost(1)
             + market.getAgent(j).getPrice(i) ) ) {
            ((Flow)newFlows.elementAt(i)).set(j,
            ((AgentModel)network.elementAt(j)).getTransportCost(1)
            + market.getAgent(j).getPrice(i) );
            min = ((Flow)newFlows.elementAt(i)).getCost();
        }
        int target = ((Flow)newFlows.elementAt(i)).getTarget();
        boolean infrastructure =
            ((AgentModel)network.elementAt(target)).isInfrastructure();

        ((Flow)newFlows.elementAt(i)).setMargin(
        infrastructure?(((AgentModel)network.elementAt(target)).getMargin()):0 );

        long sb = ((AgentModel)network.elementAt(i)).getTransportCost( false, 1);
        ((Flow)secondbestFlows.elementAt(i)).setCost(sb);
        ((Flow)secondbestFlows.elementAt(i)).setTarget(i);

        for ( int j=0; j<networkSize; j++) if ( (j!=number) && (j!=target) )
        if ( sb > ( ((AgentModel)network.elementAt(j)).getTransportCost(1)
             + ((j!=i)?market.getAgent(j).getPrice(i):0) ) ) {
            ((Flow)secondbestFlows.elementAt(i)).set( j,
            ((AgentModel)network.elementAt(j)).getTransportCost(1)
            + ((j!=i)?market.getAgent(j).getPrice(i):0) );
            sb = ((Flow)secondbestFlows.elementAt(i)).getCost();
        }

        change |= (oldPrice!=min);
    }
    return change;
}
/******************
 * Try to guess
 ******************/

public void deleteAllFlows () {
    soldFlows.removeAllElements();
    purchasedFlows.removeAllElements();
}
public AgentModel getAgentModel( int i) {
    return ((AgentModel)network.elementAt(i));
}
public boolean getInfrastructure ( int n ) {
    return ((AgentModel)network.elementAt(n)).isInfrastructure();
}
public long getInfrastructureCost() {
    long p=0;
    for ( int i=0; i<networkSize; i++) if ( i!=number)
        p -= ((AgentModel)network.elementAt(i)).getInfrastructureCost();
    return p;
}
public long getNewPrice(int i) {
    return ((Flow)newFlows.elementAt(i)).getPrice();
}
public Point getPosition() {
    return position;
}
/************************************
 * Price to deliver to i (cost + margin)
 ************************************/
public long getPrice(int i) {
    return ((Flow)flows.elementAt(i)).getPrice();
}
public long getProfit() {
    long p = 0;
    int soldFlowsSize = soldFlows.size();
    if ( soldFlowsSize>0) for ( int i=0; i<soldFlowsSize; i++) {
        p += ((Flow)soldFlows.elementAt(i)).getProfit();
    }
    int purchasedFlowsSize = purchasedFlows.size();
    if ( purchasedFlowsSize>0) for ( int i=0; i<purchasedFlowsSize; i++)
        if (((Flow)purchasedFlows.elementAt(i)).getSource() == this.number) {
```

```
            p -= ((Flow)purchasedFlows.elementAt(i)).getCost();
        }
    for ( int i=0; i<networkSize; i++) if ( i!=number) {
        p -= ((AgentModel)network.elementAt(i)).getInfrastructureCost();
        p -= ((AgentModel)network.elementAt(i)).sendData
            * ((Flow)newFlows.elementAt(i)).getCost();
    }
    return p;
}
public long getTransportCost() {
    long p = 0;
    int purchasedFlowsSize = purchasedFlows.size();
    if ( purchasedFlowsSize>0) for ( int i=0; i<purchasedFlowsSize; i++)
        if (((Flow)purchasedFlows.elementAt(i)).getSource() == this.number) {
            p -= ((Flow)purchasedFlows.elementAt(i)).getCost();
        }
    for ( int i=0; i<networkSize; i++) if ( i!=number) {
        p -= ((AgentModel)network.elementAt(i)).sendData
            * ((Flow)newFlows.elementAt(i)).getCost();
    }
    return p;
}
public long getTransportProfit() {
    long p = 0;
    int soldFlowsSize = soldFlows.size();
    if ( soldFlowsSize>0) for ( int i=0; i<soldFlowsSize; i++) {
        p += ((Flow)soldFlows.elementAt(i)).getProfit();
    }
    return p;
}
/*****************************************************************************
 * This method sets all delivery prices to direct delivery costs plus some margin.
 *****************************************************************************/

public void initPrices () {
    for (int i=0; i<networkSize; i++) {
        secondbestFlows.addElement( new Flow(number, i,(i!=number)?
            ((AgentModel)network.elementAt(i)).getTransportCost(1):0,
            (i!=number)?((AgentModel)network.elementAt(i)).getMargin():0, 0));
        flows.addElement( new Flow(number, i,(i!=number)?
            ((AgentModel)network.elementAt(i)).getTransportCost(1):0,
            (i!=number)?((AgentModel)network.elementAt(i)).getMargin():0, 0 ));
        newFlows.addElement( new Flow(number, i,(i!=number)?
            ((AgentModel)network.elementAt(i)).getTransportCost(1):0,
            (i!=number)?((AgentModel)network.elementAt(i)).getMargin():0, 0));
    }
    secondbestFlows.trimToSize();
    flows.trimToSize();
    newFlows.trimToSize();
    network.trimToSize();
}
/***************************************
 * Add a new route to the purchased Routes
 ***************************************/
private void newPurchasedFlow( int from, int to, int s) {
    purchasedFlows.addElement( new Flow( from, to, s*market.getAgent(from).getNewPrice(to), 0, s));
}
/***************************************
 * Add a new route to the sold Routes
 ***************************************/
private void newSoldFlow ( int from, int to, int s) {
    int target = ((Flow)newFlows.elementAt(to)).getTarget();

    soldFlows.addElement( new Flow( from, target, to, s*((Flow)newFlows.elementAt(to)).getCost(),
        s*((Flow)newFlows.elementAt(to)).getMargin(), s ));
    if ( target != to) {
        market.getAgent(target).newSoldFlow ( number, to, s);
        newPurchasedFlow ( target, to, s);
    }
}
/*****************************************************************************
 * Wenn eine Infrastruktur besteht, wird die Einsparung errechnet und kalkuliert,
 * ob Einsparung + verkaufte Päckchen größer ist als die Kosten
 *****************************************************************************/
public boolean probablisticTestRoute ( int a) {
    AgentModel amodel = ((AgentModel)network.elementAt(a));
    boolean b = amodel.isInfrastructure();
    long cost = 0;
    if ( b) {
        cost += amodel.getInfrastructureCost(true);
        for ( int i=0; i<networkSize; i++) if (i!=number) {
            int s = ((AgentModel)network.elementAt(i)).sendData;
            if ( (s>0)&&(((Flow)newFlows.elementAt(i)).getTarget()==a) )
            {
                cost += s*((Flow)newFlows.elementAt(i)).getCost();
                cost -= s*((Flow)secondbestFlows.elementAt(i)).getCost();
            }
        }
        long soldFlowRealProfit = 0;
        long soldFlowMaxProfit = 0;
```

```
        for ( int i=0; i<soldFlows.size();i++)
            if (((Flow)soldFlows.elementAt(i)).getTarget()==a) {
                soldFlowRealProfit += ((Flow)soldFlows.elementAt(i)).getProfit();
                soldFlowMaxProfit += ((Flow)soldFlows.elementAt(i)).sendData
                    * amodel.maxMargin;
            }
        if ( soldFlowRealProfit < cost ) {
            if ( soldFlowMaxProfit < cost ) {
                setInfrastructure(a);
            } else amodel.increaseMargin(market);
            return true;
        } else return false;
    } else {
        cost -= amodel.getInfrastructureCost(true);
        for ( int i=0; i<networkSize; i++) if (i!=number) {
            int s = ((AgentModel)network.elementAt(i)).sendData;
            if (s>0) {
                long profit = ((Flow)newFlows.elementAt(i)).getCost();
                profit -= (a==i?0:market.getAgent(a).getNewPrice(i));
                profit -= ((AgentModel)network.elementAt(a)).getTransportCost(true,1);
                if (profit>0) cost += s*profit;
            }
        }
        for ( int i=0; i<soldFlows.size();i++)
            if ( ((Flow)soldFlows.elementAt(i)).getTarget()==a ) {
                int s = ((Flow)soldFlows.elementAt(i)).sendData;
                long profit = ((Flow)newFlows.elementAt(a)).getCost();
                profit -= ((AgentModel)network.elementAt(a)).getTransportCost(true, 1);
                cost += (profit>0)?(s*profit):0;
            }
        if ( accept( cost ,temperature)) {
            setInfrastructure(a);
            ((AgentModel)network.elementAt(a)).resetMargin();
            return true;
        } else return false;
    }
}
public long purchased ( ) {
    long p = 0;
    if ( purchasedFlows.size()>0) for ( int i=0; i<purchasedFlows.size(); i++)
        p -= ((Flow)purchasedFlows.elementAt(i)).getCost();
    return p;
}
public void resetMargin ( int a) {
    ((AgentModel)network.elementAt(a)).resetMargin();
}
public void resetNetworkParameters () {
    for (int i=0; i<networkSize; i++)
        ((AgentModel)network.elementAt(i)).resetParameters(market);
}
public void setFlows( ) {
    for ( int i=0; i<networkSize; i++) if (i!=number) {
        int newTarget = ((Flow)newFlows.elementAt(i)).getTarget();
        int send =((AgentModel)network.elementAt(i)).sendData;
        if (( newTarget!=i)&&( send>0)) {
            market.getAgent(newTarget ).newSoldFlow( number, i, send);
            newPurchasedFlow( newTarget, i, send);
        }
    }
}
public void setInfrastructure ( int n ) {
    ((AgentModel)network.elementAt(n)).setInfrastructure(
        !((AgentModel)network.elementAt(n)).isInfrastructure());
}
/*******************************************
 * Set the infrastucture to n the state of b
 *******************************************/
public void setInfrastructure ( int n, boolean b ) {
    ((AgentModel)network.elementAt(n)).setInfrastructure(b);
}
public void setTemperature() {
    temperature = market.startTemperature;
}
/*****************************************************************************
 * Returns the value of profit generated by transport services for other agents
 *****************************************************************************/
public long sold() {
    long p = 0;

    if ( soldFlows.size()>0) for ( int i=0; i<soldFlows.size(); i++)
    p += ((Flow)soldFlows.elementAt(i)).getProfit();

    return p;
}
/*****************************************************************************
 * After all optimization is made we overwrite the old routes with the new ones
 *****************************************************************************/
public void syncRoutes () {
```

```
    for ( int i=0; i<networkSize; i++) {
        ((Flow)flows.elementAt(i)).set(((Flow)newFlows.elementAt(i)).getTarget(),
        ((Flow)newFlows.elementAt(i)).getCost(),
                ((Flow)newFlows.elementAt(i)).getMargin());
    }
}
public String toString() {
    return "Profit : " + getProfit();
}
}
```

```
package Network;

import java.lang.Double;
import java.awt.Point;

public class AgentModel {
    private int number;
    public Point position;
    private boolean infrastructure = false;
    public int sendData = 0;
    private int t;
    private long infrastructureCost;
    private long transportCost;
    private long reducedTransportCost;
    public long maxMargin;
    private long margin = 0;
    public long distance;
    private double potence = 1;
/**************************************************************************
 * Representation of the market participants our agent can route through
 *************************************************************************/

AgentModel ( int n, int x, int y, int s) {
    this.number = n;
    this.position = new Point (x, y);
    this.sendData = s;
    this.t = 0;
}
AgentModel ( int n, java.awt.Point p, int s) {
    this.number = n;
    this.position = new Point(p);
    this.sendData = s;
    this.t = 0;
}
AgentModel ( int n, Point p, int s, long d, Market m) {
    this.number = n;
    this.position = p;
    this.sendData = s;
    this.t = 0;
    this.distance = d;
    this.maxMargin = (long) ( (m.initMargin * distance *
        (m.variableTransportCost-m.variableInfrastructureTransportCost))/100 );
}
AgentModel ( int n, java.awt.Point p, int s, boolean b) {
    this.number = n;
    this.position = new Point (p);
    this.sendData = s;
    this.t = 0;
    infrastructure = b;
}
public boolean decreaseMargin (Market m) {
    if ( m.n == 1) {
        t++;
        if ( t < m.periods)
            margin =(long) ( maxMargin * ( 1 - t / m.periods));
        else {
            margin = 0;
            infrastructure = false;
            return true;
        }
    } else if ( m.n == 0 ) {

    } else {
        t++;
        if ( t < m.periods)
            margin = new Double( maxMargin * Math.pow( (double)1 - (double)t
                / (double)m.periods, m.n)).longValue();
        else {
            margin = 0;
            infrastructure = false;
            return true;
        }
    }
    return false;
}
/**************************************************************************
 * If there is an infrastructure connection (connected) we have fixed costs
 *************************************************************************/
```

```
public long getInfrastructureCost   () {
    if (infrastructure) return infrastructureCost;
    else return 0;
}
/**********************************
 * Simulate the infrastructure cost
 **********************************/

public long getInfrastructureCost   ( boolean c) {
    if (c)  return infrastructureCost;
    else return 0;
}
/**********************************
 * I dont like to comment this
 **********************************/
protected long getMargin () {
    return margin;
}
/**********************************
 * Hell what was that ?
 **********************************/
public final Point getPosition() {
    return position;
}
/*************************************************************************
 * Returns the direct transport cost to the agent represented by this
 * instance for the packets from this Agent
 *************************************************************************/
public long getTransportCost () {
    if (infrastructure) return reducedTransportCost*sendData;
    else return transportCost*sendData;
}
/*************************************************************************
 * Returns the direct transport cost to the agent represented by this
 * instance for s packets
 *************************************************************************/
public long getTransportCost ( int s) {
        if (infrastructure) return reducedTransportCost*s;
        else return transportCost*s;
}
/*************************************************************************
 * Simulates the direct transport cost to the agent represented by this
 * instance for the packets from this Agent. If c is true we simulate a
 * infrastructure connection
 *************************************************************************/
public long getTransportCost ( boolean c) {
    if (c) return reducedTransportCost*sendData;
    else return transportCost*sendData;
}
/*****************************************************************************
 * Simulates the direct transport cost to the agent represented by this
 * instance for s packets. If c is true we simulate a infrastructure connection
 *****************************************************************************/
public long getTransportCost ( boolean c, int s) {
        if (c) return reducedTransportCost*s;
        else return transportCost*s;
}
public boolean increaseMargin (Market m) {
    if ( m.n == 1) {
        t++;
        if ( t < m.periods)
            margin = (long) ( maxMargin * ( (double)t / m.periods ));
        else {
            margin = 0;
            infrastructure = false;
            return true;
        }
    } else if ( m.n == 0 ) {

    } else {
        t++;
        if ( t < m.periods)
            margin = new Double( maxMargin *
                Math.pow( (double)t / (double)m.periods, m.n)).longValue();
        else {
            margin = 0;
            infrastructure = false;
            return true;
        }
    }
    return false;
}
/*************************************************************************
 * Return true if there is a infrastructure connection from the Agent this
 * instance of AgentBasic is part of
 *************************************************************************/
public boolean isInfrastructure () {
    return infrastructure;
}
public void resetMargin () {
    this.margin = 0;
    t = 0;
```

```
}
public void resetParameters ( long d, Market market) {
    this.distance = d;
    this.infrastructureCost = (long) distance* market.fixedInfrastructureCost;
    this.transportCost = (long) distance* market.variableTransportCost;
    this.reducedTransportCost = (long) distance
        * market.variableInfrastructureTransportCost;
    this.maxMargin = (long) ( (market.initMargin * distance
        * (market.variableTransportCost
        - market.variableInfrastructureTransportCost))/100 );
}
public void resetParameters (Market market) {
    this.infrastructureCost = (long) distance* market.fixedInfrastructureCost;
    this.transportCost = (long) distance* market.variableTransportCost;
    this.reducedTransportCost = (long) distance
                * market.variableInfrastructureTransportCost;
    this.maxMargin = (long)((market.initMargin*distance*(market.variableTransportCost
                - market.variableInfrastructureTransportCost))/100 );
}
protected void setCost( int ic, int tc, int rtc) {
        this.infrastructureCost = ic;
        this.transportCost  = tc;
        this.reducedTransportCost = rtc;
}
/*****************************************
 * This method sets the infrastructure
 *****************************************/
public void setInfrastructure ( boolean i) {
    infrastructure = i;
}
protected void setMargin (int m) {
    this.margin = m;
}
public String toString() {
    java.lang.StringBuffer strb = new java.lang.StringBuffer();
    strb.append( number+" - ");

    strb.append(infrastructure?"yes":"no");
    strb.append("; Packets : ");
    strb.append(sendData);
    return strb.toString();
}
}

package Network;

import java.io.*;
import java.util.Vector;
import java.util.StringTokenizer;
import java.util.Properties;
import java.lang.*;
import java.io.PrintWriter;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.awt.event.*;
import java.awt.Point;

public class Market implements Runnable, Individual {
    boolean safeOptimizedMap = true;
    boolean closeTopo = false;
    int runs = 5;
    public String name = "";

    java.util.Random rand;
    Thread t = null;
    public static boolean isApplication = false;
    public Vector workers = null;
    private int elementNumber;
    private Vector test = new Vector();

    private Thread mainThread;
    private PrintWriter out
        = new java.io.PrintWriter(new java.io.OutputStreamWriter(System.out),true);
    private static java.util.ResourceBundle resmarket
        = java.util.ResourceBundle.getBundle("market");

    private int control = 100;
    private int m=0;
    private boolean b[][];
    private int x=0;

    public double fitness;
    public int maxtrans = -1;

    private Properties properties;

    private Vector actionListeners = new Vector( 0, 1);

    public int height = -1;
    public int width  = -1;
```

```
    public long variableTransportCost;
    public long variableInfrastructureTransportCost;
    public long fixedInfrastructureCost;
    public long initMargin;

    private double expectancyTransport;
    private double standardDeviation;
    private double netProb;
    private int minDist;

    protected int periods = 0;
    protected double n = 0;

    protected double startTemperature =
        Double.valueOf( resmarket.getString("startTemperature") ).doubleValue();
    protected double stopTemperature =
        Double.valueOf( resmarket.getString("stopTemperature") ).doubleValue();
    protected double cRate =
        Double.valueOf( resmarket.getString("cRate") ).doubleValue();

public Market () {
    rand = new java.util.Random();

    this.elementNumber = Integer.parseInt(resmarket.getString("Vertices"));
    this.height = Integer.parseInt(resmarket.getString("Height"));
    this.width  = Integer.parseInt(resmarket.getString("Width"));
    this.minDist = Integer.parseInt(resmarket.getString("minDist"));;
    this.expectancyTransport =
        Double.valueOf(resmarket.getString("expectancyTransport")).doubleValue();
    this.standardDeviation =
        Double.valueOf(resmarket.getString("standardDeviation")).doubleValue();
    this.netProb = Double.valueOf(resmarket.getString("netProb")).doubleValue();

    this.variableTransportCost = Long.parseLong(resmarket.getString("Transport"));
    this.variableInfrastructureTransportCost =
        Long.parseLong(resmarket.getString("RedTransport"));
    this.fixedInfrastructureCost =
        Long.parseLong(resmarket.getString("Infrastructure"));
    this.initMargin = Long.parseLong(resmarket.getString("initMargin"));
}
public Market ( int height, int width, int minDist, int elements, long
        variableTransportCost, long variableInfrastructureTransportCost, long
        fixedInfrastructureCost, long initMargin, double netProb ) {

    rand = new java.util.Random();
    this.height = height;
    this.width  = width;
    this.elementNumber = elements;
    this.variableTransportCost = variableTransportCost;
    this.variableInfrastructureTransportCost = variableInfrastructureTransportCost;
    this.fixedInfrastructureCost = fixedInfrastructureCost;
    this.initMargin = initMargin;
    this.netProb = netProb;
}
private final boolean accept (double deltaFitness, double temp) {
    return ( (deltaFitness > 0) ||
        (rand.nextFloat() < (Math.exp(Math.max((deltaFitness/temp),-100)))));
}
public void addActionListener( ActionListener a) {
    if ( !actionListeners.contains(a) ) actionListeners.addElement(a);
}
public void changeMarketParameters() {
    for ( int i=0; i<elementNumber; i++)
        ((Agent)workers.elementAt(i)).resetNetworkParameters();
}
/*****************************************
 * This method returns a copy of this market *
 *****************************************/
public Object clone() {
    Market mnew = new Market( height, width, minDist, elementNumber,
            variableTransportCost, variableInfrastructureTransportCost,
            fixedInfrastructureCost, initMargin, netProb);

    mnew.workers = new Vector();
    mnew.name = name;

    // create the agentmodels in the copied agents
    for ( int i=0; i<elementNumber; i++) {
        mnew.workers.addElement(
            new Agent( i, ((Agent)workers.elementAt(i)).getPosition(), mnew));
        for ( int j=0; j<elementNumber; j++)
            ((Agent)mnew.workers.elementAt(i)).addAgent(
                new AgentModel( j, ((Agent)workers.elementAt(j)).getPosition(),
            ((AgentModel)((Agent)workers.elementAt(i)).network.elementAt(j)).sendData,
            ((AgentModel)((Agent)workers.elementAt(i)).network.elementAt(j)).distance,
             this));
    }
    mnew.randomizeNet();
    mnew.init();
    mnew.fitness();
```

```
    return (Object)mnew;
}
public double comutate(Individual x, double temp) {
    int a = ((int)(rand.nextFloat()*elementNumber))%elementNumber;
    int b;
    do { b = ((int)(rand.nextFloat()*elementNumber))%elementNumber; } while ( a==b);
    double diff = fitness;
    boolean changed;

    if ( x == null) {
        ((Agent)workers.elementAt(a)).setInfrastructure(b);
        double newfitness = fitness();
        if ( !accept( (newfitness-fitness), temp))
            ((Agent)workers.elementAt(a)).setInfrastructure(b);
        else fitness = newfitness;
    } else {
        boolean is = ((Agent)workers.elementAt(a)).getInfrastructure(b);
        if ( is == ((Market)x).getAgent(a).getInfrastructure(b) )
            do { b = ((int)(rand.nextFloat()*elementNumber))%elementNumber; }
            while ( a==b);
        if ( b>=0) if ( !((Agent)workers.elementAt(a)).getInfrastructure(b))
            do { b = ((int)(rand.nextFloat()*elementNumber))%elementNumber; }
            while ( a==b);
        ((Agent)workers.elementAt(a)).setInfrastructure(b);
        double newfitness = fitness(); //if diff is < 0 cost gettin smaller
        if ( !accept( (newfitness-fitness), temp))
            ((Agent)workers.elementAt(a)).setInfrastructure(b);
        else fitness = newfitness;
    }
    return (diff-fitness);
}
public double comutate(Individual x, double temp, float coopt) {
    return comutate(x, temp);
}
/*****************************************************************************
 * We need a problem .. well we will design it here.
 * 20 nodes or agents, the probability for a necessary information transport
 * is 10 % (0.1) and the probability for a connection beetween two agents is
 * 5 % ( 0.05)
 *****************************************************************************/
public void create () {
    if ( workers!=null ) workers.removeAllElements();
    Point[] v = randomizeSet( );
    workers  = randomizeInfo ( v);
    randomizeNet ( );
    init();
    notifyActionEvent();
}
public double fitness() {
    double profit = 0;
    boolean changed = true;
    do {
        changed = false;
        for (int i=0; i<elementNumber; i++)
            changed|=(((Agent)workers.elementAt(i)).checkForBetterPrices());
        for (int i=0; i<elementNumber; i++) ((Agent)workers.elementAt(i)).syncRoutes();
    } while (changed);
    for (int i=0; i<elementNumber; i++) ((Agent)workers.elementAt(i)).deleteAllFlows();
    for (int i=0; i<elementNumber; i++) ((Agent)workers.elementAt(i)).setFlows();
    for (int p=0; p<elementNumber; p++)
            profit += ((Agent)workers.elementAt(p)).getProfit();

    return (profit/100);
}
/*****************************************************
 * Returns the agent i. The agents are in a fixed order.
 *****************************************************/
public Agent getAgent(int i) {
    if ( i < elementNumber ) return (Agent)workers.elementAt(i);
    return null;
}
public long getInfrastructureCost() {
    long cost = 0;
    for ( int p=0; p<elementNumber; p++)
        cost += ((Agent)workers.elementAt(p)).getInfrastructureCost();
    return cost;
}
public long getTransportCost() {
    long cost = 0;
    for ( int p=0; p<elementNumber; p++)
        cost += ((Agent)workers.elementAt(p)).getTransportCost();
    return cost;
}
public long getTransportProfit() {
    long cost = 0;
    for ( int p=0; p<elementNumber; p++)
        cost += ((Agent)workers.elementAt(p)).getTransportProfit();
```

```
    return cost;
}
public void init() {
    boolean changed;

    for (int i=0; i<elementNumber; i++) {
        ((Agent)workers.elementAt(i)).initPrices();
    }
    do {
        changed = false;
        for (int i=0; i<elementNumber; i++)
            changed|=(((Agent)workers.elementAt(i)).checkForBetterPrices());
        for (int i=0; i<elementNumber; i++)((Agent)workers.elementAt(i)).syncRoutes();
    } while (changed);

    for (int i=0; i<elementNumber; i++)((Agent)workers.elementAt(i)).deleteAllFlows();
    for (int i=0; i<elementNumber; i++)((Agent)workers.elementAt(i)).setFlows();

    fitness = fitness();

}
    public boolean load ( String path, String file) {
        if (file.endsWith("bmap")) return loadBasicMap( path, file);
        return loadFullMap( path, file);
    }
public void loadAgentParameters () {
    periods = periods>0?periods:Integer.parseInt(resmarket.getString("periods"));
    n = n>0?n:Double.valueOf( resmarket.getString("n") ).doubleValue();

}
/*********************************************************
 * Loads a set of vertex + graphs from the file *load*.map *
 *********************************************************/
public boolean loadBasicMap ( String dir, String file) {
    this.name = file;

    BufferedReader data = null;
    int number = 0;
    Vector v = new Vector();
    workers = new Vector();
    String buff = null;
    java.awt.Point[] position = null;

    int flowMatrix[][] = null;

    properties = new Properties();

    System.out.println("Opening file "+file.toString());

    data = IoUtil.makeBufferedReader( dir, file );

    try {
        boolean reader = false;
        if ( data!=null) do {
            try {
                buff = data.readLine();
            } catch ( java.io.IOException e) { System.out.println(e); }

            if ( buff != null ) {
                StringTokenizer str = new StringTokenizer( buff, ";= ", false);

                if (str.countTokens() > 0)
                if (buff.startsWith("Positions")) position =IoUtil.readPointVec(data);
                else if (buff.startsWith("Flows")) flowMatrix =
                    IoUtil.readIntMatrix( position.length, position.length, data);
                else setParameter ( str.nextToken(), str.nextToken() );
            }
        } while ( buff!=null );
        else return false;

        elementNumber = position.length;
        if ( elementNumber == 0) return false;
        for (int i=0; i<elementNumber; i++)
            workers.addElement( new Agent( i, position[i], this) );

        int target = 0, packets = 0;
        String s;

        for (int i=0; i<elementNumber; i++) for (int j=0; j<elementNumber; j++)
            ((Agent)workers.elementAt(i)).addAgent( new AgentModel(
                j, ((Agent)workers.elementAt(j)).getPosition(), flowMatrix[i][j] ));
        init();
    } catch (Exception e) {
        System.out.println("Error while parsing data");
        return false;
    }
    init();
    notifyActionEvent();
    return true;
}
public boolean loadFullMap ( String path, String file) {
    setMargin(0);
    this.name = file;
```

```
    System.out.println("Opening file " + file);
    int knotenzahl = 0;
    BufferedReader data = IoUtil.makeBufferedReader( path, file);
    String buff = null;
    java.util.StringTokenizer tokenizer = null;
    try { do {
            String str = data.readLine();
            tokenizer = new java.util.StringTokenizer(str );
    } while ( tokenizer.countTokens() == 0); } catch ( java.io.IOException e) {}
    try {
        knotenzahl =  Integer.parseInt( tokenizer.nextToken());
    } catch (NumberFormatException e) { System.out.println( e); return false; }
    try { do {
        String str = data.readLine();
        tokenizer = new java.util.StringTokenizer(str, "; /t/r/n", false );
    } while ( tokenizer.countTokens() == 0); } catch ( java.io.IOException e) {}
    try {
        width =  Integer.parseInt( tokenizer.nextToken());
        height = Integer.parseInt( tokenizer.nextToken());
    } catch (NumberFormatException e) {
        System.out.println("Problems while parsing positions! "+e);
        return false;
    }
    workers = new Vector();
/*  Laden der Matrizen für normale Kosten, reduzierte Kosten, Kosten der
    Infrastruktur und Transportbedarfe für jeden Knoten.
*/
    java.awt.Point[] position = null;
    int normalCostMatrix[][] = null;
    int smallCostMatrix[][]  = null;
    int connCostMatrix[][]   = null;
    int marginMatrix[][]     = null;
    int infrastructureMatrix[][] = null;
    position = IoUtil.readPointVec( data);
    normalCostMatrix = IoUtil.readIntMatrix( knotenzahl, knotenzahl, data);
    smallCostMatrix  = IoUtil.readIntMatrix( knotenzahl, knotenzahl, data);
    connCostMatrix   = IoUtil.readIntMatrix( knotenzahl, knotenzahl, data);
    marginMatrix     = IoUtil.readIntMatrix( knotenzahl, knotenzahl, data);
    infrastructureMatrix = IoUtil.readIntMatrix( knotenzahl, knotenzahl, data);
    int flowMatrix[][][] = new int [knotenzahl][][];
    for ( int i=0; i<knotenzahl; i++)
        flowMatrix[i] = IoUtil.readIntMatrix( knotenzahl, knotenzahl, data);
    workers = new Vector();
    for ( int i=0; i<knotenzahl; i++) {
        workers.addElement( new Agent( i, position[i], this) );
        for ( int j=0; j<knotenzahl; j++) {
            int flow = 0;
            boolean is = infrastructureMatrix[i][j]==1;
            for ( int h=0; h<knotenzahl; h++) if (h!=j)
                flow -= flowMatrix[i][j][h] - flowMatrix[i][h][j];
            AgentModel am = new AgentModel( j, position[j], (i==j?0:flow), is );
        am.setCost(connCostMatrix[i][j],normalCostMatrix[i][j],smallCostMatrix[i][j]);
            am.setMargin(marginMatrix[i][j]);
            ((Agent)workers.elementAt(i)).addAgent(am);
        }
    }
    elementNumber = knotenzahl;
    init();
    notifyActionEvent();
    return true;
}
public boolean loadURL( java.net.URL url, String file) {
    this.name = file;
    BufferedReader data = null;
    int number = 0;
    Vector v = new Vector();
    workers = new Vector();
    String buff = null;
    java.awt.Point[] position = null;
    int flowMatrix[][] = null;
    properties = new Properties();
    System.out.println("Opening file "+file.toString());
    data = IoUtil.makeBufferedReader( url, file );
    try {
        boolean reader = false;
        if ( data!=null) do {
            try {
```

```
                buff = data.readLine();
            } catch ( java.io.IOException e) { System.out.println(e); }
            if ( buff != null ) {
                StringTokenizer str = new StringTokenizer( buff, ";= ", false);
                if (str.countTokens() > 0)
                if (buff.startsWith("Positions")) position =IoUtil.readPointVec(data);
                else if (buff.startsWith("Flows")) flowMatrix =IoUtil.readIntMatrix(
                    position.length, position.length, data);
                else setParameter (str.nextToken(), str.nextToken());
            }
        } while ( buff!=null );
        else return false;
        elementNumber = position.length;
        if ( elementNumber == 0) return false;
        for (int i=0; i<elementNumber; i++)
                workers.addElement( new Agent( i, position[i], this) );
        int target = 0, packets = 0;
        String s;
        for (int i=0; i<elementNumber; i++) for (int j=0; j<elementNumber; j++)
                ((Agent)workers.elementAt(i)).addAgent(new AgentModel( j,
                ((Agent)workers.elementAt(j)).getPosition(), flowMatrix[i][j] ));
        init();
    } catch (Exception e) {
        System.out.println("Error while parsing data");
        return false;
    }
    init();
    notifyActionEvent();
    return true;
    }
/**
 * Insert the method's description here.
 * Creation date: (6/16/2000 10:45:48 PM)
 * @param args java.lang.String[]
 */
public static void main(String[] args) {
    Market m = new Market();
    m.load(".", args[0]);
    m.staticModel();
    System.out.println( m.fitness()) ;
    }
/**
 * This method was created in VisualAge.
 */
public void notifyActionEvent() {
    Vector v;
    synchronized ( this ) {
        v = (Vector) actionListeners.clone();
    }
    int cnt = v.size();
    for ( int i=0; i<cnt; i++) ((ActionListener) v.elementAt(i)).actionPerformed(
        new ActionEvent( this, ActionEvent.ACTION_PERFORMED, "Hello"));
}
/*****************************************
 * Adds a 1 to the boolean array b          *
 * but we ignore every position where x = y *
 *****************************************/
private boolean raise (int xpos, int ypos) {
    if (xpos==ypos) xpos++;
    if (xpos>=elementNumber) return false;
    b[xpos][ypos]=!b[xpos][ypos];
    this.x=xpos;
    if (!b[xpos][ypos]) if ( !raise( (++xpos)%elementNumber,
        ((xpos==elementNumber)?(++ypos):ypos) )) return false;
    return true;
}
    public void randomize() {
        randomizeNet();
    }
/*****************************************************************
 * This method set demand of the agents. The demand is derived from
 * expectancy and standardDeviation.
 *****************************************************************/
private Vector randomizeInfo ( Point[] position ) {
    Vector w = new Vector( 0, 1);
    boolean near = true;
    for ( int i=0; i<elementNumber; i++) {
        Agent a = new Agent( i, position[i], this);
        int[] nearest = null;
        if (closeTopo) nearest = Vec.getNearest( 5, position[i], position);
        else nearest = Vec.getRandom ( 5, position[i], position);
        for ( int j=0; j<elementNumber; j++) {
            int packets = (int)Math.round(this.expectancyTransport+rand.nextGaussian()
```

```
                *this.standardDeviation);
// nur die naechsten kriegen Packete, wenn diese Bedingung gesetzt ist!
            near = Vec.contains ( j, nearest);
            packets = ((packets>0)&&(i!=j))?packets:1;
            a.addAgent(new AgentModel( j, position[j], near?packets:0  ) );
        }
        w.addElement(a);
    }
    w.trimToSize();
    return w;
}
/*****************************************************************************
 * Here we randomize a few connections. Whereby there is a given probability for
 * a connection between two agents
 *****************************************************************************/

private void randomizeNet( ) {
    boolean build;
    for ( int i=0; i<elementNumber; i++)
        for ( int j=0; j<elementNumber; j++) {
            build = (rand.nextDouble()<=this.netProb)&&(i!=j);
            ((Agent)workers.elementAt(i)).setInfrastructure(j, build);
            ((Agent)workers.elementAt(i)).resetMargin(j);
        }
    return;
}
/*****************************************************************************
 * Creates a set of m points. The distance of two points (later agents) is at least
 * the value MarketData.minDist either on the abscissa or the ordinate
 *****************************************************************************/

private Point[] randomizeSet( ) {
        Point[] position = new Point[elementNumber];

        for ( int i = 0; i < elementNumber; i++) {
            position[i] = randomPoint ( "Random", width, height );
            boolean test;
            do {
                test = true;
                for ( int j = 0; j < i; j++)
                if (( java.lang.Math.abs (position[j].x - position[i].x) < this.minDist ) ||
                    ( java.lang.Math.abs (position[j].y - position[i].y) < this.minDist ))
                        test = false;
                if ( test == false ) position[i] = randomPoint ( "Random", width, height );
            }   while ( !test );
        }
        System.gc();
    return position;
}
    private java.awt.Point randomPoint ( String str, int width, int height){
        int x=0;
        int y=0;
        if ( str.equalsIgnoreCase("Random") ) {
            x = java.lang.Math.abs(rand.nextInt()) % width;
            y = java.lang.Math.abs(rand.nextInt()) % height;
        }
        if ( str.equalsIgnoreCase("Gaussian") ) {
            x = Math.max(0, Math.min(width,
                (int)(rand.nextGaussian()*width/3 +width/2)));
            y = Math.max(0, Math.min(height,
                (int)(rand.nextGaussian()*height/3 +height/2)));
        }
        if ( str.equalsIgnoreCase("AntiGaussian") ) {
            x = (int)(( 1 - rand.nextGaussian()) * width);
            y = (int)(( 1 - rand.nextGaussian()) * height);
        }
        return new java.awt.Point(x,y);
    }
/*****************************************************************
 * Parameters for the central heuristics (PopSize; temp; cRate)
 *****************************************************************/
public void readAgentParameters (java.io.BufferedReader d) {
    String lnReader;
    boolean test;
    loadAgentParameters();

    do {
        int p = this.periods;
        test = true;
    System.out.print("How long should the agent offer a transport service ("+p+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                p = Integer.parseInt(lnReader);
                if ( p<1) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else this.periods = p;
            }
        } catch ( Exception e ) {
```

```
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);
    do {
        double n = this.n;
        test = true;
        System.out.println("Please define the agent pricing behaviour :");
        System.out.println("/tn=0 ... constant margin");
        System.out.println("/tn<1 ... degressiv");
        System.out.println("/tn=1 ... linear");
        System.out.println("/tn>1 ... progressiv");
        System.out.print("For n != 0 the formula is (1-time/p)^n (n="+(float)n+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                n = Double.valueOf( lnReader).doubleValue();
                if ( n<0) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else this.n = n;
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);

}
/*************************
 * Parameters for the map *
 *************************/
public void readBuildParameters( java.io.BufferedReader d) {
    String lnReader;
    boolean test;
    double cool;
    do {
        int el = this.elementNumber;
        test = true;
        System.out.print("How many vertex ("+el+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                el = Integer.parseInt(lnReader);
                if ( el<3) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else this.elementNumber = el;
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);
    do {
        double prob = this.expectancyTransport;
        test = true;
        System.out.print("Expected value of transport
            demand from vertex A to vertex B ("+prob+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                prob = Double.valueOf( lnReader).doubleValue();
                this.expectancyTransport = prob;
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);
    do {
        double prob = this.standardDeviation;
        test = true;
        System.out.print("Standard deviation of transport
            demand from vertex A to vertex B ("+prob+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                prob = Double.valueOf( lnReader).doubleValue();
                if (0>prob) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else this.standardDeviation = prob;
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
```

```
        }
    } while (!test);
    do {
        double prob = this.netProb;
        test = true;
        System.out.print("Probability of infrastructure from a to b ("+prob+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                prob = Double.valueOf( lnReader).doubleValue();
                if ( (0>prob)||(prob>1)) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else this.netProb = prob;
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);

}
/**
 * This method was created in VisualAge.
 * @param a java.awt.event.ActionListener
 */
public void removeActionListener( ActionListener a) {
    if ( actionListeners.contains(a) ) actionListeners.removeElement(a);
}
/**
 * This method was created in VisualAge.
 */
public void run() {
    boolean changed = false, changed2 = false;
    boolean cooledDown = false;

    long sold, purchased, profit, profit2;
    switch (m) {
        case 0:
        int eq = 0;
        for ( int go=0; go<runs; go++) {

            for ( int p=0; p<elementNumber; p++)
                ((Agent)workers.elementAt(p)).setTemperature();
            randomizeNet();
            eq = 0;
            int r=0;
            do {
                r++;
                if ( r%control == 0 ) {
                    profit = 0; sold=0; purchased=0;
                    for ( int p=0; p<elementNumber; p++) {
                        profit      += ((Agent)workers.elementAt(p)).getProfit();
                        sold        += ((Agent)workers.elementAt(p)).sold();
                        purchased   += ((Agent)workers.elementAt(p)).purchased();
                    }
                    out.println(r+"\t; "+(profit/100)+"\t; "+(sold/100)
                                +"\t, "+(purchased/100));
                    cooledDown = true;
                    for ( int p=0; p<elementNumber; p++)
                        cooledDown &= ((Agent)workers.elementAt(p)).checkDiff();
                    fitness();
                    notifyActionEvent();
                }

                /* if something changed we will check all routes again (over and
                    over till we reach an equilibrium situation)*/
                if (changed2) {
                    do {
                        changed = false;
                        for (int i=0; i<elementNumber; i++) changed|=
                            (((Agent)workers.elementAt(i)).checkForBetterPrices());
                        for (int i=0; i<elementNumber; i++)
                            ((Agent)workers.elementAt(i)).syncRoutes();
                    } while (changed);

                    for (int i=0; i<elementNumber; i++)
                        ((Agent)workers.elementAt(i)).deleteAllFlows();
                    for (int i=0; i<elementNumber; i++)
                        ((Agent)workers.elementAt(i)).setFlows();
                }
                /* check all routes and lower prices for the non-profitable ones */
                changed2 = false;
/*              for (int i=0; i<elementNumber; i++)
                    changed2 |= ((Agent)workers.elementAt(i)).testRoutes();
*/
                /* Test an (potential) infrastructure now! */
                if (!changed2) {
                    int a1 =Math.abs(rand.nextInt()%elementNumber);
```

```
                    int a2;
                    do {a2 =Math.abs(rand.nextInt()%elementNumber);} while(a1==a2);
                    changed2|=((Agent)workers.elementAt(a1)).probablisticTestRoute(a2);
                }
                if (!changed2) eq++;
                else eq = 0;

            } while (( eq < 10*control)&&( (r<maxtrans)||(maxtrans==-1)));

            String sname = name;
            if (safeOptimizedMap) {
                if ( sname.equals("")) sname = "run";
                if ( runs > 1) sname += "_"+go;
                saveFullMap(".", sname+".map");
            }
        }
        break;
/*************************************************************************/
        case 2:
            long oldprofit = -1000000;

            b = new boolean[elementNumber][elementNumber];
            for(int i=0; i<elementNumber; i++) for(int j=0; j<elementNumber; j++) {
                b[i][j]=false;
                ((Agent)workers.elementAt(i)).setInfrastructure( j, false);
            }
            int c0=0;
            do {
                ++c0;
                for(int i=0; i<=x; i++) for(int j=0; j<elementNumber; j++)
                    ((Agent)workers.elementAt(i)).setInfrastructure( j, b[i][j]);
                int t=0;
                do {
                    t++;
                    changed = false;
                    for (int i=0; i<elementNumber; i++) changed |=
                        (((Agent)workers.elementAt(i)).checkForBetterPrices());
                    for (int i=0; i<elementNumber; i++)
                        ((Agent)workers.elementAt(i)).syncRoutes();
                } while (changed);
                test.addElement(new Integer(t));

        if ( test.size() == 10000 ) {
            int sum = 0;
            for( int s=0; s<test.size(); s++)
                sum += ((Integer)test.elementAt(s)).intValue();
            System.out.println( elementNumber+"; 10000; "+sum);
            System.exit(0);
        }

                for (int i=0; i<elementNumber; i++)
                    ((Agent)workers.elementAt(i)).deleteAllFlows();
                for (int i=0; i<elementNumber; i++)
                    ((Agent)workers.elementAt(i)).setFlows();
                profit = 0;
                for ( int p=0; p<elementNumber; p++)
                    profit += ((Agent)workers.elementAt(p)).getProfit();
                if ( oldprofit < profit) {
                    oldprofit = profit;
                    out.println( (profit/100)+"\t, "
                        +(100*c0)/(2<<(elementNumber*(elementNumber-1))));
                    notifyActionEvent();
                }
            } while ( raise( 0, 0));
        break;
        default:;
    }
}
public boolean save ( String path, String file ) {
    return saveBasicMap( path, file );
}
public boolean saveBasicMap ( String dir, String file) {
    BufferedWriter data = IoUtil.makeBufferedWriter( dir, file);
    System.out.println("Outputfile for new map "+file);
    java.awt.Point[] pt = new java.awt.Point[elementNumber];
    for (int i = 0; i<elementNumber; i++)
        pt[i] = ((Agent)workers.elementAt(i)).getPosition();

    IoUtil.writeString (data, "Height "+this.height);
    IoUtil.writeString (data, "Width "+this.width);
    IoUtil.writeString (data, "Infrastructure "+this.fixedInfrastructureCost);
    IoUtil.writeString (data, "Transport "+this.variableTransportCost);
    IoUtil.writeString (data, "TvI "+this.variableInfrastructureTransportCost);
    IoUtil.writeString (data, "Margin "+this.initMargin);
    IoUtil.writeString (data, "Positions");
    IoUtil.writePointVec( data, pt);

    int flowMatrix[][]   = new int [elementNumber][elementNumber];
    for (int i=0; i<elementNumber; i++) for (int j=0; j<elementNumber; j++)
```

```
            flowMatrix[i][j] =
            ((AgentModel)((Agent)workers.elementAt(i)).network.elementAt(j)).sendData;
    IoUtil.writeString (data, "Flows");
    IoUtil.writeIntMatrix( data, flowMatrix);
    try { data.close();
    } catch ( java.io.IOException e) {}
    return true;
}
public boolean saveFullMap ( String path, String file) {
    System.out.println("Writing file " + file);

    BufferedWriter data = IoUtil.makeBufferedWriter( path, file);
    String buff = null;
    java.awt.Point[] pt = new java.awt.Point[elementNumber];

    long normalCostMatrix[][] = new long[elementNumber][elementNumber];
    long smallCostMatrix[][]  = new long[elementNumber][elementNumber];
    long connCostMatrix[][]   = new long[elementNumber][elementNumber];
    long marginMatrix[][]      = new long[elementNumber][elementNumber];

    int infrastructureMatrix[][] = new int [elementNumber][elementNumber];
    int flowMatrix[][][]   = new int [elementNumber][elementNumber][elementNumber];
    for (int i = 0; i<elementNumber; i++) for (int j = 0; j<elementNumber; j++)
        for (int h = 0; h<elementNumber; h++) flowMatrix[i][j][h]=0;

    for (int i = 0; i<elementNumber; i++) {
        pt[i] = ((Agent)workers.elementAt(i)).getPosition();
        for (int j = 0; j<elementNumber; j++) {
            AgentModel am = ((Agent)workers.elementAt(i)).getAgentModel(j);
            normalCostMatrix[i][j] = am.getTransportCost(false, 1);
            smallCostMatrix[i][j]  = am.getTransportCost(true, 1);
            connCostMatrix[i][j]   = am.getInfrastructureCost(true);
            marginMatrix[i][j]     = am.getMargin();
            infrastructureMatrix[i][j] = am.isInfrastructure()?1:0;
        }
    }
    IoUtil.writeString( data, String.valueOf(elementNumber));
    IoUtil.writeString( data, String.valueOf(width)+"; "+String.valueOf(height));
    IoUtil.writePointVec( data, pt);
    IoUtil.writeLongMatrix( data, normalCostMatrix);
    IoUtil.writeLongMatrix( data, smallCostMatrix);
    IoUtil.writeLongMatrix( data, connCostMatrix);
    IoUtil.writeLongMatrix( data, marginMatrix);
    IoUtil.writeIntMatrix( data, infrastructureMatrix);

    for (int i = 0; i<elementNumber; i++) {
        Agent a = ((Agent)workers.elementAt(i));
        for (int j = 0; j< elementNumber; j++) {
            Flow f = (Flow) a.newFlows.elementAt(j);
            int send = ((AgentModel)a.network.elementAt(j)).sendData;
            if ( send > 0 ){
                flowMatrix[i][i][f.getTarget()] += send;
                if ( f.getTarget()!=j ) {
                    int from = f.getTarget();
                    do {
                        int to = ((Flow)((Agent)workers.elementAt(from)).
                            newFlows.elementAt(f.getFinalTarget())).getTarget();
                        flowMatrix[i][from][to] += send;
                        from = to;
                    } while ( from != f.getFinalTarget() );
                }
            }
        }
    }
    for (int i = 0; i<elementNumber; i++)
        IoUtil.writeIntMatrix( data, flowMatrix[i]);
    try {
        data.close();
    } catch ( java.io.IOException e) {}
    return true;
}
public void setAgentParameters ( int periods, double n) {
    this.periods = periods;
    this.n = n;
}
public void setBuildParameters ( int height, int width, int minDist, int elements,
              double expectancyTransport, double standardDeviation ) {
    this.height = height;
    this.width  = width;
    this.minDist = minDist;
    this.elementNumber = elements;
    this.expectancyTransport = expectancyTransport;
    this.standardDeviation = standardDeviation;
}
public void setCost( long variableTransportCost, long
        variableInfrastructureTransportCost, long fixedInfrastructureCost ) {
    this.variableTransportCost = variableTransportCost;
    this.variableInfrastructureTransportCost = variableInfrastructureTransportCost;
    this.fixedInfrastructureCost = fixedInfrastructureCost;
```

```
}
public void setMargin ( int m) {
    initMargin = m;
}
public void setMarket(Individual m) {
    this.workers = ((Market)m).workers;
}
public void setMethod(int m) {
    this.m=m;
    if ( m==0) for (int i=0; i<elementNumber; i++)
        ((Agent)workers.elementAt(i)).setTemperature();

}
public void setOutput( PrintWriter out) {
    this.out = out;
}
public void setParameter ( String buff, String value) {
    if ( buff.startsWith("Height") ) try {
        this.height = Integer.parseInt( value);
        } catch (NumberFormatException e) { System.out.println(e); }
    else if ( buff.startsWith("Width") ) try {
        this.width = Integer.parseInt( value);
        } catch (NumberFormatException e) { System.out.println(e); }
    else if ( buff.startsWith("Infrastructure") ) try {
            this.fixedInfrastructureCost = Long.parseLong( value);
        } catch (NumberFormatException e) { System.out.println(e); }
    else if ( buff.startsWith("Transport") ) try {
            this.variableTransportCost = Long.parseLong( value);
        } catch (NumberFormatException e) { System.out.println(e); }
    else if ( buff.startsWith("TvI") ) try {
            this.variableInfrastructureTransportCost = Long.parseLong( value);
        } catch (NumberFormatException e) { System.out.println(e); }
    else if ( buff.startsWith("Margin") ) try {
            this.initMargin = Long.parseLong( value);
        } catch (NumberFormatException e) { System.out.println(e); }
}
public void setTemperature ( double start, double stop, double cR) {
    startTemperature = start;
    stopTemperature = stop;
    cRate = cR;
}
public void startThread (int method) {
    m = method;
    if (t!=null) t.stop();
    t = new Thread( this);
    t.start();
}
public void staticModel() {
    for ( int i=0; i<elementNumber; i++) {
        Agent a = ((Agent)workers.elementAt(i));
        for ( int j=0; j<elementNumber; j++) if (j != i) {
            AgentModel b = a.getAgentModel(j);
            double minus = b.getInfrastructureCost(true);
            double perPaket = b.getTransportCost(false, 1) - b.getTransportCost(true, 1);
            double plus = b.sendData * perPaket;

            double k_ab = b.getTransportCost(false, 1);

            for ( int k=0; k<elementNumber; k++) if ((k!=i) && (k!=j)) {
                Agent kn = ((Agent)workers.elementAt(k));
                double k_kb = kn.getAgentModel(j).getTransportCost(false, 1);
                double k_ka = kn.getAgentModel(i).getTransportCost(false, 1);
                double p1 = k_kb / ( k_ka + k_ab);
                plus += p1 * kn.getAgentModel(j).sendData;

                double p2 = k_ka / ( k_ab + k_kb);
                plus += p2 * a.getAgentModel(k).sendData;
            }
            if ( minus <= plus ) b.setInfrastructure(true);
            else b.setInfrastructure(false);
        }
    }
}
public void stopThread () {
    if ( t!=null) {
        while (t.isAlive() ) { t.stop(); }
        t = null;
        System.gc();
    }
}
}

package Network;

import java.awt.*;
import java.lang.Math;

public class MarketGraph extends java.awt.Canvas {
    public boolean showTransport = true;
    public boolean showInfrastructure = true;
```

```
    private Market market = null;
    private Market marketBuffer = null;
    private int elementNumber = 0;

    private Color agentColor = Color.red;
    private Color infrastructureColor = Color.blue;
    private Color transportColor = Color.black;
    private Color bgColor = Color.lightGray;

    private int xPos[], yPos[];
    private int polx [] = new int [8];
    private int poly [] = new int [8];
    private boolean painting = false;


    float arrowLength = 15;
    float arrowWidth  = 3; //half the real width of the arrow
    float arrowLineWidth = 3;
    float infrastructureWidth = 2;


    private Image offScreenImage;
    private Graphics offScreenGraphics;
    private Dimension offScreenSize;
/**************************************
 * MarketGraph constructor comment.
 **************************************/
public MarketGraph() {
    super();
    setBackground( bgColor);
}
public void changeInfrastructureVisibilityState () {
    showInfrastructure = !showInfrastructure;
    repaint();
}
public void changeTransportVisibilityState () {
    showTransport = !showTransport;
    repaint();
}
/*****************************************
 * Update without erasing the background.
 *****************************************/
public void paint (java.awt.Graphics g) {
    if ( market != null) {
        painting = true;
        float xMax = market.width;
        xMax /= getSize().width-20;
        float yMax = market.height;
        yMax /= getSize().height-20;
        g.setColor( Color.lightGray);
        g.fillRect( 0, 0, getSize().width, getSize().height);
/*
    Oben links beschriften
*/      g.setColor( Color.black);
        g.drawString("Cost of Infrastructure : ", 20, 20);
        g.drawString(""+market.getInfrastructureCost(), 150, 20);
        g.drawString("Cost of Transport : ", 20, 40);
        g.drawString(""+market.getTransportCost(), 150, 40);
        g.drawString("Profit of Transport : ", 20, 60);
        g.drawString(""+market.getTransportProfit(), 150, 60);
        g.drawString("Fitness : ", 320, 20);
        g.drawString(""+market.fitness, 450, 20);
/*
    Unten links beschriften
        g.setColor( Color.black);
        g.drawString("Cost of Infrastructure : ", 20, getSize().height-70);
        g.drawString(""+market.getInfrastructureCost(), 150, getSize().height-70);
        g.drawString("Cost of Transport : ", 20, getSize().height-50);
        g.drawString(""+market.getTransportCost(), 150, getSize().height-50);
        g.drawString("Profit of Transport : ", 20, getSize().height-30);
        g.drawString(""+market.getTransportProfit(), 150, getSize().height-30);
        g.drawString("Fitness : ", 20, getSize().height-10);
        g.drawString(""+market.fitness, 150, getSize().height-10);
*/
        if (showInfrastructure) {
            g.setColor( infrastructureColor);
            for (int i=0; i<elementNumber; i++) for (int j=0; j<elementNumber; j++)
                if (i!=j) if (((AgentModel)((Agent)market.workers.elementAt(i))
                        .network.elementAt(j)).isInfrastructure()) {
                int sx = (int)(xPos[i]/xMax);
                int sy = (int)(yPos[i]/yMax);
                int ex = (int)(xPos[j]/xMax);
                int ey = (int)(yPos[j]/yMax);

                double dz = Math.sqrt ( (ex-sx)*(ex-sx) + (ey-sy)*(ey-sy) );
                double dx = (ex-sx);
                double dy = (ey-sy);

                double px = ex - arrowLength*dx/dz;
                double py = ey - arrowLength*dy/dz;
```

```
                double alpha = ((dx != 0)?Math.atan (dy/dx):Math.PI/2);
                double diffx = - ( (infrastructureWidth/2 ) * Math.sin( alpha ));
                double diffy = + ( (infrastructureWidth/2 ) * Math.cos( alpha ));
                double changex =  infrastructureWidth * Math.sin( alpha );
                double changey = -infrastructureWidth * Math.cos( alpha );
/*              if ( i > j ) {
                    polx [0] = 10+(int) Math.round( sx + diffx + changex);
                    poly [0] = 10+(int) Math.round( sy + diffy + changey);
                    polx [1] = 10+(int) Math.round( sx - diffx + changex);
                    poly [1] = 10+(int) Math.round( sy - diffy + changey);
                    polx [2] = 10+(int) Math.round( px - diffx + changex);
                    poly [2] = 10+(int) Math.round( py - diffy + changey);
                    polx [3] = 10+(int) Math.round(
                        px + arrowWidth * Math.sin( alpha ) - diffx + changex);
                    poly [3] = 10+(int) Math.round(
                        py - arrowWidth * Math.cos( alpha ) - diffy + changey);
                    polx [4] = 10+(int) Math.round( ex + diffx + changex);
                    poly [4] = 10+(int) Math.round( ey + diffy + changey);
                } else {
                    polx [0] = 10+(int) Math.round( sx - diffx - changex);
                    poly [0] = 10+(int) Math.round( sy - diffy - changey);
                    polx [1] = 10+(int) Math.round( sx + diffx - changex);
                    poly [1] = 10+(int) Math.round( sy + diffy - changey);
                    polx [2] = 10+(int) Math.round( px + diffx - changex);
                    poly [2] = 10+(int) Math.round( py + diffy - changey);
                    polx [3] = 10+(int) Math.round(
                        px - arrowWidth * Math.sin( alpha ) + diffx - changex);
                    poly [3] = 10+(int) Math.round(
                        py + arrowWidth * Math.cos( alpha ) + diffy - changey);
                    polx [4] = 10+(int) Math.round( ex - diffx - changex);
                    poly [4] = 10+(int) Math.round( ey - diffy - changey);
                }
*/
                polx [0] = 10+(int) Math.round( sx + diffx );
                poly [0] = 10+(int) Math.round( sy + diffy );
                polx [1] = 10+(int) Math.round( sx - diffx );
                poly [1] = 10+(int) Math.round( sy - diffy );
                polx [2] = 10+(int) Math.round( px - diffx );
                poly [2] = 10+(int) Math.round( py - diffy );
                polx [3] = 10+(int) Math.round(
                    px + arrowWidth * Math.sin( alpha ) - diffx );
                poly [3] = 10+(int) Math.round(
                    py - arrowWidth * Math.cos( alpha ) - diffy );
                polx [4] = 10+(int) Math.round( ex );
                poly [4] = 10+(int) Math.round( ey );
                polx [5] = 10+(int) Math.round(
                    px - arrowWidth * Math.sin( alpha ) + diffx );
                poly [5] = 10+(int) Math.round(
                    py + arrowWidth * Math.cos( alpha ) + diffy );
                polx [6] = 10+(int) Math.round( px + diffx );
                poly [6] = 10+(int) Math.round( py + diffy );
                g.fillPolygon ( polx, poly, 7 );
            }
        }
        if (showTransport) {
            g.setColor( transportColor);
            int sendData[] = new int[elementNumber];
            for ( int i=0; i< elementNumber; i++) {
                for ( int j=0; j< elementNumber; j++) if (i!=j) sendData[j] = 0;
                for ( int j=0; j< elementNumber; j++) if (i!=j) sendData[((Flow)
                    ((Agent)market.workers.elementAt(i)).newFlows.elementAt(j))
                    .getTarget()]+=((AgentModel)((Agent)market.workers.elementAt(i))
                    .network.elementAt(j)).sendData;
                Agent a = (Agent)market.workers.elementAt(i);
                for ( int j=0; j<a.soldFlows.size(); j++) sendData[((Flow)
                    a.soldFlows.elementAt(j)).getTarget()]
                    += ((Flow)a.soldFlows.elementAt(j)).sendData;
                for ( int j=0; j< elementNumber; j++) if (i!=j) if (0<sendData[j])
                    if ( !(showInfrastructure && ((AgentModel)((Agent)market
                    .workers.elementAt(i)).network.elementAt(j)).isInfrastructure()))
                    {
                    // if we show Infrastructures we do not need to see the packetflows
                        int sx = (int)(xPos[i]/xMax);
                        int sy = (int)(yPos[i]/yMax);
                        int ex = (int)(xPos[j]/xMax);
                        int ey = (int)(yPos[j]/yMax);
                        float dz = (float)Math.sqrt ( (ex-sx)*(ex-sx) + (ey-sy)*(ey-
sy) );
                        float dx = (ex-sx);
                        float dy = (ey-sy);
                        double alpha = ((dx != 0)?(float)Math.atan (dy/dx):Math.PI/2);
                        float px = ex - arrowLength*dx/dz;
```

```
                    float py = ey - arrowLength*dy/dz;

                    polx [0] = 10+ ex; poly[0] = 10+ ey;
/*                  Maybe this is a helpfull hint!
                    polx [1] = Math.round(
                        (float)(px + (arrowWidth) *Math.cos( alpha+Math.PI/2 )));
                    poly [1] = Math.round(
                        (float)(py + (arrowWidth) *Math.sin( alpha+Math.PI/2 )));
                    polx [2] = Math.round(
                        (float)(px - (arrowWidth) *Math.cos( alpha+Math.PI/2 )));
                    poly [2] = Math.round(
                        (float)(py - (arrowWidth) *Math.sin( alpha+Math.PI/2 )));
*/
                    polx[1]=10 +Math.round((float)(px -arrowWidth *Math.sin( alpha)));
                    poly[1]=10 +Math.round((float)(py +arrowWidth *Math.cos( alpha)));
                    polx[2]=10 +Math.round((float)(px +arrowWidth *Math.sin( alpha)));
                    poly[2]=10 +Math.round((float)(py -arrowWidth *Math.cos( alpha)));

                    g.drawLine ( (int)10+ sx, (int)10+ sy, 10+ ex, 10+ ey );
                    g.fillPolygon ( polx, poly, 3 );
                }
            }
        }

        g.setColor( agentColor);
        for (int i=0; i< elementNumber; i++) g.fillOval(
            10+Math.round(xPos[i]/xMax)-8, 10+Math.round(yPos[i]/yMax)-8, 15, 15);
        painting = false;
    }
}
public void setMarket ( Market m) {
    if (!painting) {
        this.market = m;
        elementNumber = m.workers.size();
        xPos = new int[elementNumber];
        yPos = new int[elementNumber];
        for ( int i=0; i< elementNumber; i++) {
            xPos[i] = ((Agent)market.workers.elementAt(i)).getPosition().x;
            yPos[i] = ((Agent)market.workers.elementAt(i)).getPosition().y;
        }
        this.marketBuffer = null;
        repaint();
    } else this.marketBuffer = m;
}
    public final synchronized void update ( Graphics theG ) {
        Dimension d = getSize();
        if((offScreenImage == null) || (d.width != offScreenSize.width) ||
            (d.height != offScreenSize.height)) {
        offScreenImage = createImage(d.width, d.height);
        offScreenSize = d;
        offScreenGraphics = offScreenImage.getGraphics();
        offScreenGraphics.setFont(getFont());
    }
      offScreenGraphics.fillRect( 0, 0, d.width, d.height);
      paint( offScreenGraphics );
      theG.drawImage(offScreenImage, 0, 0, null);
    }
}
package Network;

import java.util.Vector;
import java.util.Random;
import java.io.PrintWriter;
import java.awt.event.*;

public class Population implements java.lang.Runnable{
    // *** Settings ***
    boolean safeOptimizedMap = true;

    Thread t = null;
    private int m = 0; //choose the method
    private PrintWriter out = new PrintWriter( System.out);
    private double startTemperature;
    private double stopTemperature;
    private double temperature;

    private double cRate;
    private int popSize = 10;
    public int runs = 10;

    private Vector indivs;
    private int pop10, count, allbest;
    private double bestfitness, diffz, total = 0;
    private Random rand;
    private boolean output = true;
    private int run, j, s;
    public Individual indiv;

    private Vector actionListeners = new Vector( 0, 1);
```

```
    private static java.util.ResourceBundle resmarket
        = java.util.ResourceBundle.getBundle("market");
public Population () {
    super();
    popSize = Integer.parseInt(resmarket.getString("popSize"));
    startTemperature =
        Double.valueOf( resmarket.getString("startTemperature") ).doubleValue();
    stopTemperature =
        Double.valueOf( resmarket.getString("stopTemperature") ).doubleValue();
    cRate = Double.valueOf( resmarket.getString("cRate") ).doubleValue();
}
public Population( Individual x) {
    super();
    this.indiv = x;
}
public void addActionListener( ActionListener a) {
    if ( !actionListeners.contains(a) ) actionListeners.addElement(a);
}
public Individual getBest() {
    return (Individual) indivs.elementAt(allbest);
}
public Individual getIndiv() {
    return indiv;
}
public void loadBuildParameters () {
    popSize = Integer.parseInt(resmarket.getString("popSize"));
    startTemperature =
        Double.valueOf( resmarket.getString("startTemperature") ).doubleValue();
    stopTemperature =
        Double.valueOf( resmarket.getString("stopTemperature") ).doubleValue();
    cRate = Double.valueOf( resmarket.getString("cRate") ).doubleValue();
}
public void notifyActionEvent() {
    Vector v;
    synchronized ( this ) {
        v = (Vector) actionListeners.clone();
    }
    int cnt = v.size();
    for ( int i=0; i<cnt; i++) ((ActionListener) v.elementAt(i)).actionPerformed(
        new ActionEvent( this, ActionEvent.ACTION_PERFORMED, "Hello"));
}
/*****************************************************************
 * Parameters for the central heuristics (PopSize; temp; cRate)
 *****************************************************************/
public void readParameters( int m, java.io.BufferedReader d) {
    String lnReader;
    boolean test;
    double cool;
    loadBuildParameters ( );
    do {
        int p = popSize;
        test = true;
        switch (m) {
            case 1:
            case 2: System.out.print("How many simulations to run ("+p+")? ");
            break;
            case 3: System.out.print("Population size ("+p+")? ");
            break;
            default: System.out.println("Error! ");
                     return;
        }
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                p = Integer.parseInt(lnReader);
                if ( p<0) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else popSize = p;
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);
    do {
        double t = startTemperature;
        test = true;
        System.out.print("Initial temperature ("+(int)t+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                t = Double.valueOf( lnReader).doubleValue();
                if ( t<0) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else startTemperature = t;
```

```
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);
    do {
        double t = stopTemperature;
        test = true;
        System.out.print("Stop temperature ("+(int)t+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                t = Double.valueOf( lnReader).doubleValue();
                if (( t<=0)||(t>startTemperature)) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else stopTemperature = t;
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);

    do {
        double c = cRate;
        test = true;
        System.out.print("Cooling rate - value between 0 and 1 ("+c+")? ");
        try {
            lnReader = d.readLine();
            if (!lnReader.equals("")) {
                c = Double.valueOf( lnReader).doubleValue();
                if ( (0>c)||(c>1)) {
                    System.out.println("Wrong parameter!");
                    test = false;
                } else cRate = c;
            }
        } catch ( Exception e ) {
            System.out.println("Wrong parameter!");
            test = false;
        }
    } while (!test);
}
public void removeActionListener( ActionListener a) {
    if ( actionListeners.contains(a) ) actionListeners.removeElement(a);
}
public void run () {
    if ( output) {
        out.println("Population size: "+ popSize+"; Initial temperature: "
            +startTemperature+"; Cooling rate: "+cRate);
        out.println("");
        out.println("Transitionen;\taverage fitness;\tbest fitness");
    }
    rand = new java.util.Random();
    indivs = new Vector();
    pop10 = 10*popSize;

    for (int go=0; go<runs ; go++) {
        if ( output) {
            out.println("");
            out.println("Run: "+go+" ("+popSize+"/"+cRate+"/"+temperature+")");
        }
        indivs.removeAllElements();
        System.gc();

        temperature =startTemperature;
        allbest = 0; count = 0; total = 0;
        for ( int i = 0; i < popSize; i++) {
            indivs.addElement( indiv.clone() );
        //  indivs.elementAt(i).initMargin = 0;
            total += ((Market)indivs.elementAt(i)).fitness;
            if ( ((Market)indivs.elementAt(i)).fitness >
                    ((Market) indivs.elementAt(allbest)).fitness ) allbest = i;
        }
        bestfitness = ((Market) indivs.elementAt(allbest)).fitness;
        double oldbestfitness = bestfitness;
        if ( output)out.println(
            count+"\t;"+(total/popSize)+"\t;"+(bestfitness)+"\t;"+temperature);
        switch (m) {
            case 0:
            do {
                count += popSize;
                for ( int i = 0; i < popSize; i++)
                diffz += ((Market)indivs.elementAt(i)).comutate ( null, tempera-
ture );
                if ( count%pop10 == 0) {
//                  total -= diffz;
                    total = 0;
```

```
                    if ( diffz > 0 ) temperature = temperature*cRate;
                    diffz = 0;
                    allbest = 0;
// Workaround because I got some small differences in the fitness ....
                    for ( int i = 0; i < popSize; i++) {
                        double tempfitness = ((Market) indivs.elementAt(i)).fitness();
                        total += tempfitness;
                        ((Market) indivs.elementAt(i)).fitness = tempfitness;
                        if ( tempfitness > ((Market) indivs.elementAt(allbest)).fitness ) all-
best = i;
                    }
                    bestfitness = ((Market) indivs.elementAt(allbest)).fitness;
                    boolean changed = (oldbestfitness!=bestfitness);
                    oldbestfitness = bestfitness;
                    if ( output) out.println( count+"\t;"+(total/popSize)+"\t;"+(bestfitness)+"\t;"+ter
                    if (changed) notifyActionEvent();
                }
            } while ( stopTemperature < temperature );
            break;
            case 1:
            do {
                count += popSize;
                for ( int i = 0; i < popSize; i++)
                diffz += ((Market)indivs.elementAt(i)).comutate ( (Market)indivs.elementAt(
                        (int)(rand.nextFloat()*popSize)%popSize), temperature );
                if ( count%pop10 == 0) {
                    total -= diffz;
                    if ( diffz > 0 ) temperature = temperature*cRate;
                    diffz = 0;
                    allbest = 0;
                    for ( int i = 0; i < popSize; i++) if ( ((Market) indivs.elementAt(i)).fitness <
                        ((Market) indivs.elementAt(allbest)).fitness ) allbest = i;
                    bestfitness = ((Market) indivs.elementAt(allbest)).fitness;
                    boolean changed2 = (oldbestfitness==bestfitness);
                    oldbestfitness = bestfitness;
                    if ( output) out.println( count+"\t;"+(total/popSize)+"\t;"+(bestfitness)+"\t;"+ter
                    if (changed2) notifyActionEvent();
                }
            } while ( stopTemperature < temperature );
            break;
            default:
        }
        if ( output) out.println("Temperature: "+temperature);

        if (safeOptimizedMap) {
            String name = ((Market) indivs.elementAt(allbest)).name;
            if ( name.equals("")) name = "run_";
            if ( runs > 1) name += "_"+go;
            ((Market) indivs.elementAt(allbest)).saveFullMap(".", name+".map");
        }
    }
}
public void setIndiv( Individual i) {
        this.indiv = i;
    }
public void setMethod(int m) {
    this.m=m;
}
public void setOutput( java.io.PrintWriter out) {
    this.out = out;
}
public void setPopulation ( int p) {
    popSize = p;
}
public void setTemperature ( double start, double stop, double cR) {
    startTemperature = start;
    stopTemperature = stop;
    cRate = cR;
}
public void startOptimization ( int m) {
    this.m = m;
    if ( t!=null) t.stop();
    t = new Thread( this);
    t.start();
}
public void stopThread () {
    if ( t!=null) {
        while (t.isAlive() ) {
            t.stop();
        }
        this.indiv = getBest();
        t = null;
        System.gc();
    }
}
}
package Network;
```

```
/* verschiedene lustige statische Methoden..... die's ja alle fuer Arrays gene-
risch nicht gibt ;-((
 */

import java.util.Vector;

public class Vec {
/*******************************************************
 * glues the colums of matrix2 to the "right" of matrix1
 * (assuming them to have the same number of rows!)
 *******************************************************/
public static float[][] colAppend (float[][] matrix1, float[][] matrix2) {
    int cols1 = matrix1[0].length;
    int cols2 = matrix2[0].length;
    int rows = matrix1.length;
    float[][] result = new float[rows][cols1+cols2];
    for(int i=0; i < rows; i++) {
        for(int j=0; j < cols1; j++) result[i][j] = matrix1[i][j];
        for(int j=0; j < cols2; j++) result[i][j+cols1] = matrix2[i][j];
    }
    return result;
}
/* glues the colums of matrix2 to the "right" of matrix1
   (assuming them to have the same number of rows!)
 */
public static float[][] colAppend (float[][] matrix1, float[] col) {
    int cols = matrix1[0].length;
    int rows = matrix1.length;
    float[][] result = new float[rows][cols+1];
    for(int i=0; i < rows; i++) {
        for(int j=0; j < cols; j++) result[i][j] = matrix1[i][j];
        result[i][cols] = col[i];
    }
    return result;
}
public static boolean contains ( int e, int[] vec) {
        for (int i=0; i<vec.length; i++) if (vec[i]==e) return true;
        return false;
}
public static long distance (java.awt.Point p1, java.awt.Point p2) {
        return (long) java.lang.Math.sqrt( square(p1.x-p2.x)+square(p1.y-p2.y));
}
public static int[] getNearest ( int n, java.awt.Point local, java.awt.Point[] p) {
        int[] m = new int[n];
        long[] dist = new long[p.length];

        for ( int i=0; i<p.length; i++) dist[i] = distance ( local, p[i]);
        for ( int i=0; i<n; i++)    m[i] = i;

        for ( int i=n; i<p.length; i++) {
            boolean sortet = true;
            do {
                sortet = true;
                for ( int j=0; j<n-1; j++) if ( dist[j]>dist[j+1] ) {
                    int mh = m[j];
                    m[j] = m[j+1];
                    m[j+1] = mh;
                    long md = dist[j];
                    dist[j] = dist[j+1];
                    dist[j+1] = md;
                    sortet = false;
                }
            } while ( !sortet );
            if ( dist[i]<dist[n-1] ) {
                m[n-1] = i;
                dist[n-1] = dist[i];
            }
        }
        return m;
}
public static int[] getRandom ( int n, java.awt.Point local, java.awt.Point[] p) {
    int nm;
    int[] m = new int[n];

    for ( int i=0; i<n; i++) {
        nm = (int)(Math.random() * p.length);
        if ( contains (nm, m))  i--;
        else m[i] = nm;
    }
    return m;
}
/*********************************************
 * ueberschreibt arr1 ab position start mit arr2
 *********************************************/
public static void overwrite (char[] arr1, char[] arr2, int start) {
    int len=arr2.length;
    for (int i=start; i<start+len; i++) arr1[i]=arr2[i-start];
}
/************************************************************
 * bestimmt die Position von arr2 in arr1 ab der start Position
 * [(-1) wenn nicht vorhanden!]
```

```
 ************************************************************/
public static int position (char[] arr1, char[] arr2, int start) {
    int len=arr2.length;
    for (int i=start; i<arr1.length-len+1; i++) {
        for (int j=0; j<len; j++) {
            if(arr1[i+j]!=arr2[j]) break;
            if(j==len-1) return i;
        }
    }
    return -1;
}
/* bildet die TRANSPONIERTE eines Arrays
 */
public static float[][] rowAppend (float[][] matrix1, float[][] matrix2) {
    int rows1 = matrix1.length;
    int rows2 = matrix2.length;
    int cols = matrix1[0].length;
    float[][] result = new float[rows1+rows2][cols];
    for(int i=0; i < rows1; i++)
        for(int j=0; j < cols; j++) result[i][j] = matrix1[i][j];
    for(int i=0; i < rows2; i++)
        for(int j=0; j < cols; j++) result[rows1+i][j] = matrix2[i][j];
    return result;
}
/*
 */
public static float[][] rowAppend (float[][] matrix1, float[] row) {
    int rows1 = matrix1.length;
    int cols = matrix1[0].length;
    float[][] result = new float[rows1+1][cols];
    for(int i=0; i < rows1; i++)
        for(int j=0; j < cols; j++) result[i][j] = matrix1[i][j];
    for(int j=0; j < cols; j++) result[rows1][j] = row[j];
    return result;
}
/*
 */
public static float[][] rowAppend (float[] row1, float[] row2) {
    int cols = row1.length;
    float[][] result = new float[2][cols];
    for(int j=0; j < cols; j++) {
        result[0][j] = row1[j];
        result[1][j] = row2[j];
    }
    return result;
}
public static int square (int a) {
    return a*a;
}
/* bildet die TRANSPONIERTE eines Arrays
 */
public static float[] toFloatArray (Object obj) {
    if(obj.getClass().getName().equals("[F")) {
        return (float[])obj;
    } else return toFloatArray((Vector) obj);
}
public static float[] toFloatArray (Vector vec) {
        int rows = vec.size();
        float[] result = new float[rows];
        for(int i=0; i < rows; i++)
          result[i] = ((Float)vec.elementAt(i)).floatValue();
        return result;
}
/* bildet die TRANSPONIERTE eines Arrays
 */
public static float[][] toFloatMatrix (java.util.Vector vec) {
    int rows = vec.size();
    float[][] result = new float[rows][];
    for(int i=0; i < rows; i++)
        result[i] = toFloatArray(vec.elementAt(i));
    return result;
}
/* bildet die TRANSPONIERTE eines Arrays
 */
public static int[] toIntArray (Object obj) {
    if(obj.getClass().getName().equals("[I")) {
        return (int[])obj;
    } else return toIntArray((Vector) obj);
}
public static int[] toIntArray (java.util.Vector vec) {
    int rows = vec.size();
    int[] result = new int[rows];
    for(int i=0; i < rows; i++)
        result[i] = ((Integer)vec.elementAt(i)).intValue();
    return result;
}
/**
 *macht einen "schoenen" String aus einem Object
 */
```

```
public static String toString(float[][] obj) {
        String result= "";
        for (int i=0; i < obj.length; i++) result=result+toString(obj[i])+"\n";
        return result;
}
public static String toString(int[][] obj) {
        String result= "";
        for (int i=0; i < obj.length; i++) result=result+toString(obj[i])+"\n";
        return result;
}
public static String toString(long[][] obj) {
        String result= "";
        for (int i=0; i < obj.length; i++) result=result+toString(obj[i])+"\n";
        return result;
}
/**
 *macht einen "schoenen" String aus einem Object
 */
public static String toString(float[] obj) {
        String result= "";
        for (int i=0; i < obj.length; i++) result=result+obj[i]+"; ";
        return result;
}
public static String toString(int[] obj) {
        String result= "";
        for (int i=0; i < obj.length; i++) result=result+obj[i]+"; ";
        return result;
}
public static String toString(long[] obj) {
        String result= "";
        for (int i=0; i < obj.length; i++) result=result+obj[i]+"; ";
        return result;
}
/**
 *macht einen "schoenen" String aus einem Object
 */
public static String toString(java.awt.Point[] pt) {
        StringBuffer result = new StringBuffer();
        for (int i=0; i<pt.length; i++)result.append( " ("+pt[i].x+"; "+pt[i].y+")");
        result.append("\n");
        return result.toString();
}
/**
 *macht einen "schoenen" String aus einem Object
 */
public static String toString(Object obj) {
        return obj.toString();
}
/* bildet die TRANSPONIERTE eines Arrays
 */
public static float[][] transpose (float[][] matrix) {
    int rows = matrix.length;
    int cols = matrix[0].length;
    float[][] result = new float[cols][rows];
    for(int i=0; i < rows; i++)
        for(int j=0; j < cols; j++) result[j][i] = matrix[i][j];
    return result;
}
/* bildet die TRANSPONIERTE eines Arrays
 */
public static int[][] transpose (int[][] matrix) {
    int rows = matrix.length;
    int cols = matrix[0].length;
    int[][] result = new int[cols][rows];
    for(int i=0; i < rows; i++)
        for(int j=0; j < cols; j++) result[j][i] = matrix[i][j];
    return result;
}
}
package Network;

import java.io.*;
/**
 * This type was created in VisualAge.
 */
public class Workbench {
    private int m=0; // ** method value **
    private Market market = null;
    private Population pop;
    private Thread optimizerThread;

    private PrintWriter out;
    private String mapfilename = null;
    private String mapfilepath = null;

    private BufferedReader d = new BufferedReader(new InputStreamReader(System.in));
    private File f;
    public static boolean application = false;
/***********************************************
```

```
 * This method parses all the command line options *
 ************************************************/
public Workbench( String[] n) {
    super();
    out = new PrintWriter( new OutputStreamWriter(System.out), true);
    boolean loadedMap = false;
    boolean newParameters = false;
    int w = -1, h = -1;
    long infr = -1, trans = -1;
    long tvi = -1, im = -1;
    if ( n.length > 0) for ( int i=0; i < n.length; i++) {
        if (n[i].startsWith("/") && ( (n[i].length()==2) ||
            (n[i].charAt(2)=='=') || (n[i].charAt(2)==':')))
        switch (n[i].charAt(1)) {
            case 'w' :
                if ((n[i].length()==2)&&(n.length>i)) {
                    try { w = Integer.parseInt(n[++i]);
                    } catch ( NumberFormatException e ) {
                        i--;
                        System.out.println("Problem ocured while parsing width.");
                    }
                } else {
                    try { w = Integer.parseInt(n[i].substring(3,n[i].length()));
                    } catch ( NumberFormatException e ) {
                        System.out.println("Problem ocured while parsing width.");
                    }
                }
                System.out.println("Width set to "+w);
            break;
            case 'h' :
                if ((n[i].length()==2)&&(n.length>i)) {
                    try { h = Integer.parseInt(n[++i]);
                    } catch ( NumberFormatException e ) {
                        i--;
                        System.out.println("Problem ocured while parsing height.");
                    }
                } else {
                    try { h = Integer.parseInt(n[i].substring(3,n[i].length()));
                    } catch ( NumberFormatException e ) {
                        System.out.println("Problem ocured while parsing height.");
                    }
                }
                System.out.println("Height set to "+h);
            break;
            case 'o' :
                if (( n[i].length()==2)&&(n.length>i)) {
                    f = new File( n[++i]+".txt");
                } else {
                    f = new File( n[i].substring(3,n[i].length())+".txt");
                }
                try {
                   out=new PrintWriter(new DataOutputStream(new FileOutputStream(f)));
                } catch ( IOException e) { System.out.println(e); }
                System.out.println("Outputfile "+f);
            break;
            case 'l' :
                market = new Market();
                if (( n[i].length()==2)&&(n.length>i))
                    loadedMap = market.load( ".", n[++i]);
                else loadedMap = market.load( ".", n[i].substring(3,n[i].length()));
                if ( loadedMap == false ) {
                    System.out.println ("Load failed.");
                    if (application) System.exit(0);
                }
            break;
            case 's' :
                if (m<8) {
                    if (( n[i].length()==2)&&(n.length>i)) mapfilename = n[++i];
                    else mapfilename = n[i].substring(3,n[i].length());
//                  if (!mapfilename.endsWith(".map")) mapfilename += ".map";
                    mapfilepath=".";
                    System.out.println("Outputfile for new map "+mapfilename);
                    m+=8;
                }
            break;
            case 'm' :
                if ((n[i].length()==2)&&(n.length>i)) {
                    if (n[++i].equalsIgnoreCase("d"))   m+=1;
                    if (n[i].equalsIgnoreCase("sa"))    m+=2;
                    if (n[i].equalsIgnoreCase("cosa"))  m+=3;
                    if (n[i].equalsIgnoreCase("enum"))  m+=4;
                    if ( m%8==0) {
                        i--;
                        System.out.println("Problem ocured while parsing method.");
                    }
                }
                else if(n[i].substring(3,n[i].length()).equalsIgnoreCase("d"))  m+=1;
                else if(n[i].substring(3,n[i].length()).equalsIgnoreCase("sa")) m+=2;
```

```
            else if(n[i].substring(3,n[i].length()).equalsIgnoreCase("cosa"))m+=3;
            else if(n[i].substring(3,n[i].length()).equalsIgnoreCase("enum"))m+=4;
            else if( m%8==0)
                System.out.println("Problem ocured while parsing method.");
        break;
        default  :
            System.out.println("Ooops a "+n[i]+" ?");
            System.out.println("Did you read the help? [/help]");
    }

    if ( n[i].equalsIgnoreCase("/standard")) loadedMap=true;

    if ( n[i].startsWith("/") && n[i].substring( 1,
        Math.min( 5, n[i].length())).equalsIgnoreCase("load")) {

        market = new Market();
        if (( n[i].length()==5)&&(n.length>i))
            loadedMap = market.load( ".", n[++i]);
        else loadedMap = market.load( ".", n[i].substring(5,n[i].length()) );

        if ( loadedMap == false ) {
            System.out.println ("Load failed.");
            if (application) System.exit(0);
        }
    }
    if (n[i].startsWith("/") && n[i].substring( 1,
        Math.min( 5, n[i].length())).equalsIgnoreCase("safe"))
    if ( m<8 ) {
        java.io.File f;
        if (( n[i].length()==5)&&(n.length>i)) mapfilename = n[++i];
        else mapfilename = n[i].substring(5,n[i].length());
        if (!mapfilename.endsWith(".map")) mapfilename += ".map";
        mapfilepath = ".";
        System.out.println("Outputfile for new map "+mapfilename);
        m+=8;
    }
    if (n[i].startsWith("/") &&  n[i].substring( 1,
            Math.min( 7, n[i].length())).equalsIgnoreCase("output")) {
        File f = new File( n[i].substring(8,n[i].length())+".txt");
        try {
            out =new PrintWriter(new DataOutputStream(new FileOutputStream(f)));
        } catch ( IOException e) { System.out.println(e); }
        System.out.println("Outputfile "+f);
    }
    if (n[i].startsWith("/") &&  n[i].substring( 1,
            Math.min( 6, n[i].length())).equalsIgnoreCase("width")) {
        if (( n[i].length()==6) && ( n.length>i))
        try {w = Integer.parseInt( n[++i] );
        } catch ( NumberFormatException e ) {
            System.out.println("Can't parse width");
        } else try { w = Integer.parseInt(n[i].substring(7,n[i].length()));
        } catch ( NumberFormatException e ) {
            System.out.println("Can't parse width");
        }
        System.out.println("Width set to "+w);
    }
    if (n[i].startsWith("/") &&  n[i].substring( 1,
            Math.min( 7, n[i].length())).equalsIgnoreCase("height")) {
        if (( n[i].length()==7)&&(n.length>i))
        try { h =Integer.parseInt(n[++i]);
        } catch ( NumberFormatException e ) {
            System.out.println("Can't parse height");
        } else try { h = Integer.parseInt(n[i].substring(8,n[i].length()));
        } catch ( NumberFormatException e ) {
            System.out.println("Can't parse height"); }
        System.out.println("Height set to "+h);
    }
    if (n[i].startsWith("/") &&  n[i].substring( 1,
            Math.min( 7, n[i].length())).equalsIgnoreCase("method")) {
        if (n[i].length()>7) {
            if (n[i].substring(8, n[i].length()).equalsIgnoreCase("d"))    m=1;
            if (n[i].substring(8, n[i].length()).equalsIgnoreCase("sa"))   m=2;
            if (n[i].substring(8, n[i].length()).equalsIgnoreCase("cosa")) m=3;
            if (n[i].substring(8, n[i].length()).equalsIgnoreCase("enum")) m=4;
            if (m==0) System.out.println("Problem ocured while parsing method.");
        } else {
            if (n[++i].equalsIgnoreCase("d"))   m=1;
            if (n[i].equalsIgnoreCase("sa"))    m=2;
            if (n[i].equalsIgnoreCase("cosa"))  m=3;
            if (n[i].equalsIgnoreCase("enum"))  m=4;
            if ( m==0) {
                System.out.println("Problem ocured while parsing method.");
                i--;
            }
        }
    }
    if (n[i].startsWith("/") &&  n[i].substring(1,
            Math.min( 10, n[i].length())).equalsIgnoreCase("Transport")) {
```

```
            if ( n[i].length()>10 ) {
                try { trans = Integer.parseInt(n[i].substring(11,n[i].length())); }
                catch ( NumberFormatException e ) {
                    System.out.println("Can't parse TransportCost"); }
            } else {
                try { trans = Integer.parseInt( n[++i]); }
                catch ( NumberFormatException e ) {
                    System.out.println("Can't parse TransportCost");
                    i--;
                }
            }
            System.out.println("Cost for transport set to "+trans);
            newParameters = true;
        }
        if (n[i].startsWith("/") &&  n[i].substring(1,
                Math.min( 15, n[i].length())).equalsIgnoreCase("Infrastructure")) {
            if ( n[i].length()>15 ) {
                try { infr = Integer.parseInt(n[i].substring(16,n[i].length())); }
                catch ( NumberFormatException e ) {
                    System.out.println("Can't parse InfrastructureCost"); }
            } else {
                try { infr = Integer.parseInt(n[++i]); }
                catch ( NumberFormatException e ) {
                    System.out.println("Can't parse InfrastructureCost");
                    i--;
                }
            }
            System.out.println("Infrastructure cost set to "+infr);
            newParameters = true;
        }
        if (n[i].startsWith("/") &&  n[i].substring(1,
                Math.min( 17, n[i].length())).equalsIgnoreCase("reducedTransport")) {
            if (n[i].length()>17) {
                try { tvi = Integer.parseInt(n[i].substring(18,n[i].length())); }
                catch ( NumberFormatException e ) {
      System.out.println("Can't parse transport cost for infrastructure connection");}
            } else {
                try { tvi = Integer.parseInt(n[++i]); }
                catch ( NumberFormatException e ) {
      System.out.println("Can't parse transport cost for infrastructure connection");
                    i--;
                }
            }
            System.out.println("Variable transport cost set to "+tvi);
            newParameters = true;
        }
        if (n[i].startsWith("/") &&  n[i].substring(1,
                Math.min( 7, n[i].length())).equalsIgnoreCase("Margin")) {
            boolean percent=false;
            if (n[i].length()>7) {
                if (n[i].endsWith("%")) percent = true;
                try { im = Integer.parseInt(
                    n[i].substring( 8, percent?n[i].length()-1:n[i].length()));
                } catch ( NumberFormatException e ) {
                    System.out.println("Can't parse Margin"); }
            } else {
                if (n[++i].endsWith("%")) percent = true;
                try { im = Integer.parseInt(n[i].substring(
                    percent?n[i].length()-1:n[i].length())); }
                catch ( NumberFormatException e ) {
                    System.out.println("Can't parse Margin"); i--;}
            }
            System.out.println("Initial Margin set to "+im);
            newParameters = true;
            im = percent?im:im*100;
        }
    }
    if (market == null) {
        market = new Market();
        if (!loadedMap) market.readBuildParameters(d);
                //loadedMap = true -> use standardparameters
        if (h>0) market.height = h;
        if (w>0) market.width = w;
        market.create();
    } else if (newParameters) market.changeMarketParameters();
    if (newParameters) {
        if (trans>0) market.variableTransportCost = trans;
        if ( infr>0) market.fixedInfrastructureCost = infr;
        if ( tvi>=0) market.variableInfrastructureTransportCost = tvi;
        if ( im >=0) market.initMargin = im;

    }
    market.init();
    if (m>=8) {
        market.name = mapfilename;
```

```
        market.save ( mapfilepath, mapfilename );
        m%=8;
    }
    switch (m) {
        case 0: //default -- for now decentral simulation
        case 1: System.out.println("Method is set to market simulation");
                market.setOutput( out);
                market.setMethod(0);
//              market.readParameters( m%8);
                market.readAgentParameters(d);
                optimizerThread = new Thread( market, "optimizer");
                optimizerThread.start();
        break;
        case 2: System.out.println("Method is set to Simulated Annealing");
                pop = new Population ( (Individual)market);
                pop.setOutput( out);
                pop.setMethod(0);
                pop.readParameters( m%8, d); //sets parameters direct in Population
                optimizerThread = new Thread( pop, "optimizer");
                optimizerThread.start();
        break;
        case 3: System.out.println("Method is set to CoOperativ Simulated Annealing");
                pop = new Population ( (Individual)market);
                pop.setOutput( out);
                pop.setMethod(1);
                pop.readParameters( m%8, d); //sets parameters direct in Population
                optimizerThread = new Thread( pop, "optimizer");
                optimizerThread.start();
        break;
        case 4: System.out.println("Method is set to Enumeration");
                market.setOutput( out);
                market.setMethod(2);
                optimizerThread = new Thread( market, "optimizer");
                optimizerThread.start();
        break;
        default:;
    }
}
/**
 * Gets the applet information.
 * @return java.lang.String
 */
public String getAppletInfo() {
    return "Network.Start created using VisualAge for Java.";
}
/**
 * main entrypoint - starts the application
 * @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
    if ( args.length==0 || args[0].equalsIgnoreCase("/help") || args[0].equals("/?")
        || args[0].equalsIgnoreCase("/h") || args[0].trim().equals("help") ) {
System.out.println("Networksimulation (c) Tim Stockheim 1998 Germany");
System.out.println("-------------------------------------------------");
System.out.println("Available options :");
System.out.println("/g or /graphic                      - show GUI");
System.out.println("/l or /load name.map                - load a prebuild map");
System.out.println("/s or /save name.map                - build and safe a map");
System.out.println("/o or /output [Name of output file] - save output to a file");
System.out.println("/m or /method { d/sa/cosa/enum}");
System.out.println("/w or /width number     - set width of map");
System.out.println("/h or /height number    - set height of map");
System.out.println("/Infrastructure float - set relative infrastructure cost");
System.out.println("/Transport float        - set relative transport cost
                    (without infrastructure)");
System.out.println("/reducedTransport float - set relative transport cost
                    (with infrastructure)");
System.out.println("/Margin float           - set a margin a vertex take");
System.out.println("If output file is not specified the program will
                    write to the standard output.");
System.out.println("This programm will simulate a network where each
                    player has the choice between");
System.out.println("a infrastructure connection (high fixed cost) and
                    volume dependent cost (high");
System.out.println("variable cost).");
System.out.println("For a clearer understanding try to get one of the really
                    rare manuals.");
System.out.println("Some options require additional input.");
System.out.println();
    } else if ( args[0].equalsIgnoreCase("/version") ) {
        System.out.println("Networksimulation (c) Tim Stockheim 2000 Germany");
        System.out.println("-------------------- version 1.0 ------------------
----");
    } else if (args[0].equalsIgnoreCase("/g")||args[0].equalsIgnoreCase("/graphic")){
        try {
```

```
            VisualNetwork.Main aMain = new VisualNetwork.Main();
            try {
                Class aCloserClass = Class.forName("com.ibm.uvm.abt.edit.WindowCloser");
                Class parmTypes[] = { java.awt.Window.class };
                Object parms[] = { aMain };
                java.lang.reflect.Constructor aCtor=aCloserClass.getConstructor(parmTypes);
                aCtor.newInstance(parms);
            } catch (java.lang.Throwable exc) {};
            aMain.setVisible(true);
        } catch (Throwable exception) {
            System.err.println("Exception occurred in main() of java.awt.Frame");
            exception.printStackTrace(System.out);
        }
    } else  {
        application = true;
        Workbench main = new Workbench( args);
    }
} }
```

Literaturverzeichnis

[Aarts 1997] Aarts, E.H.L.; Lenstra, J.K.:*Local Search in Combinatorial Optimization*, New York, 1997

[Alstyne 1997] van Alstyne, Marshall: *The State of Network Organization: A Survey in Three Frameworks*, J. of Organizational Computing 7(3), 1997

[Berge 1958] Berge, Claude: *Theorie des Graphes et ses applications*, Paris: Dunod, S. 27-30. Englische Übersetzung: Theory of Graphs and its Application, trans. Alison Doig (London: Methuen and New York: Wilez, 1962)

[Buxmann 1996] Buxmann, P.: *Standardisierung betrieblicher Informationssysteme* Gabler, 1996

[Casey 1972] Casey, R. G.: *Allocation of Copies of a File in an Information Network*, Proceedings of AFIPS, Spring Joint Computer Conference, Arlington Virginia, 1972, S. 617-625

[Cerny 1985] Cerny, V.:*Thermodynamical Approach to the Traveling Salesman - A Efficient Simulation Algorithm*, J. of Optimization Theory 45, 1985, S. 41-51

[Chu 1969] Chu, W.: *Optimal File Allocation in a Multiple Computer System*, IEEE Trans. Comput. C-18(10), 1969, S. 865-889

[Domschke 1993] Domschke, Wolfgang: *Standortplanung, innerbetriebliche*, Enzyklopädie der Betriebswirtschaftslehre, 1993, S. 3950-3962

[Eisenführ 1995] Eisenführ, F.; Weber, M.: *Rationales Entscheiden*, Springer Verlag 3. Auflage, 1999

[Farrell 1985] Farrell, J.; Saloner, G.:*Standardization, Compatibilitz and Innovation*, J. of Economics 16, 1985, S. 70-83

[Gosh 1992] Gosh, D.; Murthy, I.; Moffett, A.: *File Allocation Problem: Comparision of Models with worst case and average Communication Delays*, Operations Research Vol. 40, Nr. 6, 1992, S. 1074-1085

[Haggett 1977] Haggett, Peter; Cliff, Andrew D.; Frey, Allan: *Locational Methods*, 1977

[Hajek 1988] Hajek, B.: *Cooling Schedules for Optimal Annealing*, Mathematics of Operation Research 13, 1988, S. 311-329

[Holler 1996] Holler, M., Illing, G.: *Einführung in die Spieltheorie*, 3. Auflage Springer, 1996

[Kansky 1963] Kansky, K. J.: *Structure of Transportation Networks: Relationships between Network Geometry and Regional Characteristics*, Chicago: University of Chicago Press, 1963

[Kirkpatrik 1983] Kirkpatrick, S; Gelatt Jr.C.D.; Vecchi, M.P.: *Optimization by Simulated Annealing*, Science 220, 1983, S. 671-680

[Laarhoven 1988] van Laarhoven, P.J.M.; Aarts, E.H.L.: *Simulated Annealing - a Review of the Theory and Applications*, Dordrecht(Kluwer), 1988

[Lundqvist 1998] Lundqvist, L.:*A Combined Model for Analysing Network Infrastructure and Land-Use/Transportation Interactions*, in Network Infrastructure and the Urban Environment, Springer, 1998

[Maes 1996] Maes, P.; Chavez, A.: *Kasbah: An Agent Marketplace for Buying and Selling Goods*, Proceedings of the First International Conference on the Practical Appication of Intelligent Agents and Multi-Agent Technology, London, UK, April 1996, S. 113-125

[Owen 1982] Owen, G.: *Game Theory*, 2.Auflage Academic Press, New York, 1982

[Powell 1990] Powell, W.W.:*Neither Market Nor Hierarchie: Network Forms of Organization*, Research in Organizational Behaviour, 1990, S. 295-336

[Sahni et al. 1976] Sahni, S.; Gonzales, T.:*P-complete approximization problems*, J. of the Association for Computing Machinery, 1976, S. 555-565

[Wegener 1994] Wegener, M.:*Operational Urban Models: State of the Art*, J. of the American Planning Association, vol. 60, S. 17-29

[Wellman 1993] Wellman, M.P.:*A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems*, J. of Artificial Inetlligence Research 1, 1993, S. 1-23

[Wendt 1995] Wendt, O.: *Tourenplanung durch Einsatz naturanaloger Verfahren*, Wiesbaden, 1995

[Wendt 2000] Wendt, O.: *Diffusion Follows Structure - A Network Model of the Software Market*, Proceedings of the 33rd Hawaii International Conference of System Science, 2000