UNIVERSITÄT  WIEN

Auswahl und Implementierung eines Ameisenalgorithmus‘ zur Steuerung von Patienten im Planspiel „INVENT“

eingereicht von

Sabine Graf

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Magistra rerum socialum oeconomicarumque
Magistra der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec.)

Fakultät für Wirtschaftswissenschaften und Informatik
der Universität Wien
Fakultät für Technische Naturwissenschaften und Informatik
der Technischen Universität Wien

Studienrichtung: Wirtschaftsinformatik

eingereicht bei

o. Univ.-Prof. Dr. Kurt Heidenberger

Betreuung:

Dipl. oec. Axel Focke

Wien, im März 2003

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ACO	Ant Colony Optimization
ACS	Ant Colony System
ANTS	Approximate Nondeterministic Tree Search
AS	Ant System
AS_{rank}	Rank Based Version of Ant System
ATSP	Asymmetric Travelling Salesman Problem
FANT	Fast Ant System
GKV	Gesetz zur Reform der gesetzlichen Krankenversicherung
GLB	Gilmore-Lawler Lower Bound
HAS	Hybrid Ant System
IV	Integrierte Versorgungsform
MMAS	MAX-MIN Ant System
QAP	Quadratic Assignment Problem
SGB	Sozialgesetzbuch
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem

Symbolverzeichnis

A	Knoten
B	Knoten
b_x	Wert des x-ten Kriteriums
c	Anzahl der Touren, die für die Berechnung der Durchschnittstour in ANTS herangezogen werden
cl	Anzahl der Knoten in der Kandidatenliste
d_{ij}	Distanz von Knoten i zu Knoten j
e	Menge an „elitist ants“
h	Anzahl der Kolonien
i	Knoten
J_i^k	Liste aller Knoten, die von Knoten i aus in der momentanen Iteration von der Ameise k noch nicht besucht worden sind
j	Nachbarknoten von Knoten i
K	Kosten einer Lösung
k	Nummer einer Ameise (k-te Ameise)
$L(t)$	Länge der Tour t der einzigen Ameise in FANT
$L^k(t)$	Länge der Tour t der Ameise k
L^+	Die Länge der kürzesten Tour
$\overline{L}$	Durchschnittliche Länge der letzten c Touren
LB	Untere Grenze der Tourlänge
l	Knoten
m	Gesamtzahl an Ameisen
n	Anzahl von bestimmten Knoten
$P_{j,z}^i(t)$	Wahrscheinlichkeit, mit der ein Datenpaket oder eine Ameise in AntNet von Knoten i zu Knoten j in der Iteration t wechselt, unter der Voraussetzung, dass Knoten z der Zielknoten ist
$p_{ij}^k(t)$	Wahrscheinlichkeit, mit der die Ameise k vom Knoten i zum Knoten j in der Iteration t übergeht
Q	heuristisch ermittelte, optimale Länge einer Tour
q	Zufallszahl zwischen 0 und 1
q_0	Parameter der Übergangsregel in ACS
r	Parameter, der während des Lösungsprozesses in FANT verändert wird
r^*	Fixer Parameter in FANT

$S_{s\to z}^{k}$	Tabelle in AntNet, die von der Ameise k vom Startknoten s bis zum Zielknoten z getragen wird und in der alle Knoten und die Zeit vom Startknoten bis zum jeweiligen Knoten enthalten sind
s	Startknoten in AntNet
$T^k(t)$	Tour der Ameise k in der Iteration t
T^+	Die bisher kürzeste Tour
t	Nummer der Tour einer Ameise bzw. Iterationsnummer
t_{max}	Maximale Anzahl an Touren einer Ameise bzw. maximale Anzahl an Iterationen
u	Knoten
w	Gewichtung des x-ten Kriteriums
x	Kriterium
z	Zielknoten eines Datenpaketes oder einer Ameise in AntNet
α	Relative Wichtigkeit der Pheromonspuren
β	Relative Wichtigkeit der lokalen Information
Γ_i	Matrix eines Knoten i, die $\mu_{i\to z}$ und $\sigma_{i\to z}^2$ aller Zielknoten z enthält
γ_i	Faktor, der die Pheromonmenge alle Kanten, die den Knoten i berühren, bei einer Veränderung der Problemstruktur beeinflusst
δ	Parameter in AntNet mit einem Wert > 1
ε	Fixer Wert mit dem Definitionsbereich [0, 1]
η_{ij}	Lokale Information der Kante (i, j); in Mehroptimierungsproblemen bezüglich des ersten Kriteriums
η'_{ij}	Lokale Information der Kante (i, j) bezüglich des zweiten Kriteriums
λ	Relative Wichtigkeit des ersten Kriterium bei Mehrzieloptimierungsproblemen
ψ	Rang einer Ameise
$\mu_{i\to z}$	Geschätzte, durchschnittliche Zeit von Knoten i zu Knoten z
ρ	Verdampfungsfaktor
$\sigma_{i\to z}^2$	Varianz der geschätzten, durchschnittlichen Zeit von Knoten i zu Knoten z
$\tau_{ij}(t)$	Globale Information über die Kante (i, j) bzw. die Menge an Pheromonen der Kante (i, j); in Mehroptimierungsproblemen bezüglich des ersten Kriteriums
$\tau'_{ij}(t)$	Globale Information über die Kante (i, j) bzw. die Menge an Pheromonen der Kante (i, j) bezüglich des zweiten Kriteriums
$\Delta\tau_{ij}^k(t)$	Menge an Pheromonen, um die eine Ameise k die Kante (i, j) ihrer Tour t verstärkt; in Mehroptimierungsproblemen bezüglich des ersten Kriteriums
$\Delta\tau'^{\,k}_{ij}(t)$	Menge an Pheromonen, um die eine Ameise k die Kante (i, j) ihrer Tour t bezüglich des zweiten Kriteriums verstärkt

$\Delta\tau_{ij}(t)$	Gesamte Menge an Pheromonen, die in der Iteration t zu der Kante (i, j) hinzugefügt wird
$\Delta\tau_{ij}^{+}$	Zusätzliche Pheromonmenge, die zu der Kanten (i, j) hinzugefügt wird, wenn diese in der besten Tour T^{+} enthalten ist
τ_0	Pheromonmenge, mit der alle Kanten initialisiert werden
τ_{max}	Maximale Menge an Pheromonen pro Kante
τ_{min}	Minimale Menge an Pheromonen pro Kante
Ψ	Knoten
ω	Anzahl der Ameisen, die in der momentanen Iteration berechtigt sind, Pheromonmengen zu aktualisieren

Abstract

In dem bereits bestehenden Planspiel INVENT, das im Rahmen dieser Arbeit nur kurz beschrieben wird, sollen Patienten, die einer sogenannten Integrierten Versorgungsform angehören, im Hinblick auf deren Strategie zu entsprechenden Krankenhäusern gelenkt werden. Die Strategie der Integrierten Versorgungsform kann auf eine Minimierung der Patientenweg, eine Einweisungen der Patienten in möglichst gut geeignete Stationen oder eine Gleichverteilung der Auslastungen aller Stationen abzielen, wobei diese Strategien auch gemischt eingesetzt werden können. Dieses Optimierungsproblem soll mittels eines Ameisenalgorithmus gelöst werden. Um einen geeigneten Ameisenalgorithmus auswählen zu können, wird daher ein breiter Überblick über diese Thematik gegeben. Nach der Auswahl des Algorithmus' und deren Begründung sollen die wesentlichen Bestandteile des Algorithmus' beschrieben und in Form von Pseudo-Code dargestellt werden. Des Weiteren werden Hinweise auf eine effiziente Programmierung des Problems gegeben. Abschließend wird gezeigt, dass der implementierte Algorithmus gute Ergebnisse für das vorliegende Problem liefert, wobei die Qualität der Lösung im Hinblick auf eine unterschiedliche Anzahl an Patienten genauer analysiert wird.

1 Einleitung

Grundlage dieser Arbeit ist das bereits bestehende, jedoch noch nicht veröffentlichte Planspiel INVENT, das von Dipl. oec. Axel Focke im Rahmen seiner Dissertation entwickelt wurde. Das Planspiel richtet sich vorwiegend an Krankenhäuser und niedergelassene Ärzte, sowie an Wissenschaftler, die sich die möglichen Auswirkungen von in Deutschland seit dem 1. Jänner 2000 zulässigen sogenannten „Integrierten Versorgungsformen“ [1] im Rahmen verschiedener Spielvarianten vor Augen führen möchten. Integrierte Versorgungsformen haben das Ziel, die bis zum Jahr 2000 im deutschen Gesundheitswesen fest verankerten Grenzen zwischen dem stationären und dem niedergelassenen Bereich aufzuweichen, ja sogar partiell aufzuheben und Krankenhäuser und niedergelassene Ärzte zu einer deutlich intensiveren Zusammenarbeit zu motivieren. Dadurch sollen die vielfach kritisierten Schnittstellenprobleme, die als einer der wesentlichen Gründe für Ineffizienzen im Gesundheitswesen ausgemacht wurden, gemildert werden. Das Beseitigen dieser Probleme soll auch dazu führen, dass sich die Pfade, die die Patienten von Leistungsanbieter zu Leistungsanbieter durch das Gesundheitswesen gehen, verändern.[2]

Im Kern des Planspiels erfolgt folgerichtig die Steuerung von Patienten durch das sich – meist regional begrenzt - verändernde Gesundheitssystem, die bei integrierten Versorgungsformen von entscheidender Bedeutung ist. In einer Vorversion des Planspiels wurde zur Steuerung der Patientenpfade auf ein regelbasiertes System zurückgegriffen, das jedoch den Ansprüchen an das Planspiel nicht ausreichend gerecht wurde.

Eine Aufgabe dieser Arbeit war es nun, aus den sogenannten Ameisenalgorithmen den- oder diejenigen herauszufinden, die für das Planspiel anwendbar sind und in angemessener Zeit zu guten Ergebnissen führen. Die zu diesem Zweck erstellte Literaturrecherche über die bestehenden Ameisenalgorithmen wurde daher, obwohl ein breiter Überblick gegeben wird, jeweils dort verkürzt oder abgebrochen, wo die dargestellten Verfahren für das Planspiel INVENT nicht von Nutzen zu sein schienen.

Eine andere Aufgabe dieser Arbeit lag darin, den oder die als geeignet herausgearbeiteten Ameisenalgorithmen zusätzlich zu implementieren. Dies bedeutete, dass, über das Maß der reinen Implementierung des Algorithmus‘ hinaus, auch die im Planspiel vorhandene Datenbank an die Bedürfnisse einer schnellen Ausführung des Ameisenalgorithmus‘ anzupassen war.

[1] Nach GVK-Gesundheitsreformgesetz 2000, Artikel 22, Absatz 5.
[2] Vgl. Glaeske, G. (2002).

2 Das Programm „INVENT“

Bevor auf die Ameisenalgorithmen und die Implementierung in das Planspiel INVENT erfolgen kann, soll in diesem Kapitel nach der Erklärung der Begriffe „Planspiel“ und „Integrierte Versorgungsform“ eine kurze Beschreibung des Programms erfolgen. Diese soll jedoch nur einen groben Überblick über das Ziel des Spiels, die Funktionen und die beiden Spielvarianten von „INVENT“ geben und die Aufgaben bzw. Entscheidungen eines Spielers beschreiben. Da dem Programm reale Daten zu Grunde liegen, wird auch auf diese Daten und auf die Adaptierung der Daten zur Verwendung im Planspiel eingegangen. Abschließend soll auf den Spielablauf näher eingegangen werden.

2.1 Begriffserklärung

Bevor das Programm selbst beschrieben wird, soll erklärt werden, was ein Planspiel ist, aus welchen Komponenten dieses normalerweise besteht und nach welchem Prinzip der Spielablauf erfolgt. Des Weiteren sollen die Fragen, in welchen Bereichen ein Planspiel zum Einsatz kommt und welchen Nutzen der Einsatz eines Planspiels bringt, kurz behandelt werden. Danach wird auf den eigentlichen Hintergrund von „INVENT“ eingegangen und beschrieben, was eine Integrierte Versorgungsform (IV) ist.

2.1.1 Allgemeines zum Thema „Planspiel“

Unter einem Planspiel versteht man ein strategisches Spiel, dem ein Simulationsmodell zu Grunde liegt. Dieses Simulationsmodell stellt eine vereinfachte Abbildung eines bestimmten Bereiches der Realität dar. Ein Spiel kann je nach Art des Planspiels von einem oder mehreren Spielern oder auch von Spielergruppen gespielt werden. Im Folgenden wird jedoch der Einfachheit immer angenommen, dass mehrere Spieler an dem Spiel beteiligt sind. Planspiele umfassen überwiegend mehrere Perioden, in denen die Spieler Entscheidungen treffen müssen. Eine Periode wird, abgesehen von zeitlichen Beschränkungen, die die Spieler berücksichtigen müssen, dann beendet, wenn die Spieler ihre Entscheidungen getroffen haben. Diese Entscheidungen dienen als Input für das Simulationsmodell. Die Berechnungen des Modells an Hand dieser Entscheidungen führen zu einem Output, der den Spielern in Form von unterschiedlich aufbereiteten Ergebnissen mitgeteilt wird. Basierend auf diesen Ergebnissen, auf Ergebnissen der Vorperioden und auf Grund von Erwartungen über zukünftige Ergebnisse treffen die Spieler weitere Entscheidungen, die wiederum im Simulationsmodell verarbeitet werden.

Eine wichtige Rolle nimmt meist die Spielleitung ein, die als Mittler zwischen Spieler und Simulationsmodell agiert. Sie hat die Aufgabe, den reibungslosen Ablauf des Spiels zu gewährleisten und im Fall von unerwünschten Abweichungen lenkend einzugreifen. Des Weiteren kontrolliert sie, ob alle Spielregeln eingehalten werden und überwacht gegebenenfalls die Entscheidungen der Spieler. Bei manchen Planspielen kann es auch nötig sein, dass die Spielleitung von Zeit zu Zeit diverse Zusatzinformationen, interpretierbar als der Praxis nachempfundene Ratschläge von Beratern oder Ergebnisse von Marktforschungen, den Spielern zur Verfügung stellt. Eine weitere wichtige Aufgabe der Spielleitung ist die Organisation und Durchführung der Vorbereitungsphase. Um die Spieler auf das Planspiel vorzubereiten, müssen diese über den Ablauf des Planspiels, die Spielregeln, die jeweilige Anfangssituation und die Zielsetzung informiert werden. Auch das zu Grunde liegende Modell sollte erklärt werden. Dabei können jedoch bestimmte Bereiche, die von den Spielern eigenständig verstanden werden sollen, ausgeblendet werden. Die Spielleitung ist aber nicht nur für die Vorbereitung der Spielteilnehmer verantwortlich sondern auch für jene des Simulationsmodells. Dazu gehört vor allem das Setzen von Parametern. Da die meisten Planspiele bereits computerunterstützt gespielt werden, gehört auch das Installieren der nötigen Software, eventuell das Bereitstellen von Daten, auf denen das Programm basiert, und das Erfüllen aller weiteren Anforderungen, die den fehlerfreien Ablauf des Programms gewährleisten, zu den Aufgaben der Spielleitung.[3]

Die Beziehungen zwischen Simulationsmodell, Spielleitung und Spieler sollen anschließend nochmals an Hand von Abbildung 2.1 dargestellt werden.

[3] Vgl. Orth, C. (1999) S. 14.

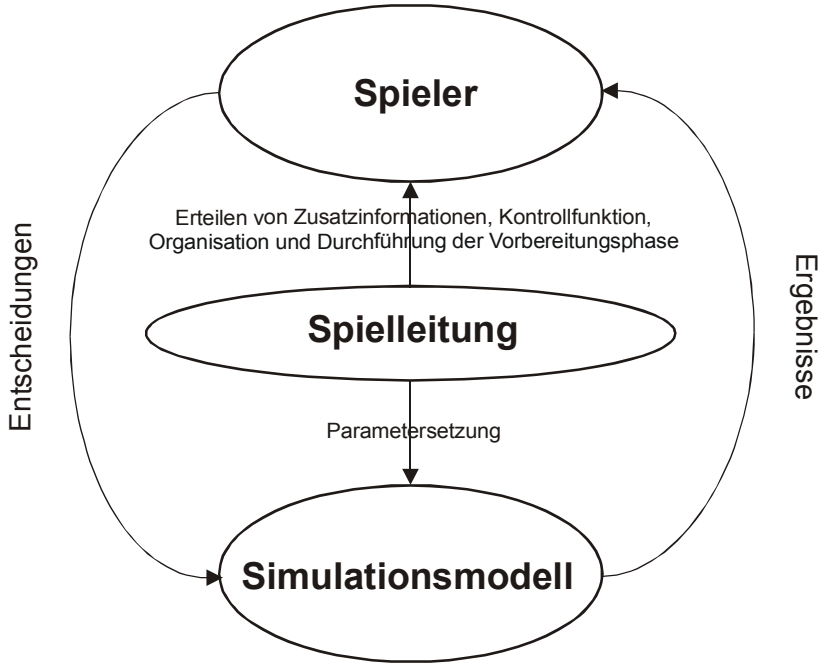

Abbildung 2.1: Darstellung der Komponenten und deren Beziehungen in einem Planspiel
(Quelle: In Anlehnung an Orth, C., 1999, S. 15)

Die Einsatzbereiche von Planspielen lassen sich in drei Kategorien unterteilen, wobei der Zweck des Spielens je nach Kategorie unterschiedlich ist. Einen wichtigen Einsatzbereich stellt die Aus- und Weiterbildung dar. Hierbei ist der Sinn des Spielens, den Teilnehmern auf spielerische Weise Wissen zu vermitteln. Ein weiterer Einsatzbereich ist das Lösen von unternehmerischen Aufgaben, wodurch ein Planspiel beispielsweise als Prognoseinstrument genutzt werden kann. Dabei stehen die Ergebnisse des Planspiels im Vordergrund, aber auch die Tatsache, dass durch mehrfaches Spielen eine geeignete Strategie erlernt und dann in der Realität angewendet werden kann, ist entscheidend. Der dritte Einsatzbereich ist die Forschung, in dem der Nutzen des Spiels vom jeweiligen Forschungsgebiet abhängig ist. Beispielsweise werden in der psychologischen Forschung Planspiele zur Untersuchung des Problemlösungs- und Lernverhalten der Teilnehmer verwendet. Hingegen werden in der wirtschaftswissenschaftlichen Forschung beispielsweise mikroökonomische Theorien simuliert.[4]

Integrierte Versorgungsformen

Integrierte Versorgungsformen sind in Deutschland trotz ihrer gesetzlichen Einführung bereits im Jahre 2000 immer noch nicht vollständig durchgesetzt. Noch immer gibt es auch in

[4] Vgl. Orth, C. (1999) S. 13.

Deutschland nur wenige davon.[5] Die USA nimmt auf diesem Gebiet die Vorreiterrolle ein, da sich Integrierte Versorgungsformen, die dort unter dem Namen „Managed Care“ bekannt sind bereits weitgehend durchgesetzt haben. Auch in der Schweiz ist dieses Modell unter dem Namen „Managed Care“ bekannt und ist bereits in der Etablierungsphase. [6] In Österreich jedoch wird es aller Wahrscheinlichkeit nach noch einige Jahre dauern, bis Integrierte Versorgungsformen eingeführt werden.

Um zu beschreiben, was eine Integrierte Versorgungsform nach bundesdeutschem Muster ist, soll hier auf den § 140 des SGB V der Bundesrepublik Deutschland Bezug genommen werden.

Eine Integrierte Versorgungsform ermöglicht gemäß § 140a Abs.1 eine „verschiedene Leistungssektoren übergreifende Versorgung der Versicherten“. Wie bereits in der Einleitung angesprochen, ist diese Aufhebung der sektoralen Trennung zwischen Krankenhäusern und niedergelassenen Ärzten eines der markantesten Wesensmerkmale der Integrierten Versorgungsformen. Eine weitere Besonderheit der Integrierten Versorgung ist jene, dass die teilnehmenden Leistungsanbieter in einer der weitreichendsten Formen einer IV gegenüber den Krankenkassen als ein Gesamtanbieter mit eigenem Management auftreten können.

Laut § 140b Abs. 1 können nun also Krankenkassen mit anderen festgelegten Vertragspartnern wie zum Beispiel Krankenhäusern oder niedergelassenen Ärzten Verträge über Integrierte Versorgungsformen abschließen.[7] In diesen Verträgen werden unter anderem die Verpflichtungen der Vertragspartner und die Vergütung der Leistungen festgelegt. Je nach Ausgestaltung dieser Verträge kann sogar die Übernahme des Risikos der Erkrankung, das bislang von den Krankenkassen getragen wurde, auf die Leistungserbringer übergehen.[8]

Die in § 140b geregelten Verpflichtungen der Leistungsanbieter stellen jedoch die sowohl qualitativ als auch quantitativ ausreichende Versorgung der Versicherten sicher, sodass für den Versicherten einer Integrierten Versorgungsform keine Nachteile durch die Teilnahme

[5] Preuß, K.-J. / Räbiger, J. / Sommer, J.H. (Hrsg) (2002) gingen (Stand: September 2001) im Vorwort Ihres Buches sogar davon aus, dass es noch gar keine integrierten Versorgungsformen im engeren Sinne in Deutschland gab.
[6] Vgl. Baur, R./ Stock, J. (2002).
[7] Diese Vertragspartner müssen laut § 140 b Abs. 2 zur ambulanten oder stationären Versorgung zugelassen sein, wodurch beispielsweise Pharmaunternehmen ausgeschlossen sind.
[8] Vgl. Glaeske, G. (2002).

daran entstehen dürfen. In § 140c wird darüber hinaus noch einmal sichergestellt, dass die Teilnehmer einer integrierten Versorgungsform unabhängig von der Ausgestaltung der Teilnahmebedingungen der IV jedenfalls auch Leistungen in Anspruch nehmen dürfen, die von der IV selbst nicht angeboten werden. Nähere Details zu den Verträgen, deren Inhalte und deren Rahmenbedingungen können dem § 140b-d entnommen werden.

Eine Integrierte Versorgungsform ermöglicht einer Krankenkasse somit die Zusammenarbeit mit bestimmten Krankenhäusern, niedergelassenen Ärzten, Rehabilitationskliniken und/oder anderen zugelassenen Vertragspartnern unter bestimmten, zuvor schon erwähnten, vertraglichen Bedingungen. Für die Versicherungsnehmer sollen Integrierte Versorgungsformen eine weitere, mögliche Versicherungsform neben der herkömmlichen Form der Krankenversicherung darstellen, die sie ausdrücklich freiwillig wählen können. Motivation für den Beitritt einer Integrieren Versorgungsform können beispielsweise das Angebot von besonderen Serviceleistungen wie Abendsprechstunden oder Telefonhotlines sein. Des Weiteren kann laut § 140g dem Versicherungsnehmer ein Bonus ausbezahlt werden, wenn er mindestens ein Jahr alle Teilnahmebedingungen eingehalten und der Integrierten Versorgungsform damit zu Einsparungen verholfen hat. Andererseits muss beachtet werden, dass für den Versicherungsnehmer ein gewisser Druck entsteht, nur noch Versorgungsanbieter innerhalb der Integrierten Versorgungsform zu nutzen, da bei Inanspruchnahme anderer Anbieter je nach Vertragsgestaltung die entsprechenden Vorteile einer integrierten Versorgungsform weitgehend wegfallen können.

Profitieren können jedenfalls alle Beteiligten einer Integrierten Versorgungsform: Die Patienten durch die bereits genannten Bonusleistungen und eine bessere Abstimmung der Leistungsanbieter, die Leistungsanbieter durch eine bessere Verzahnung und sektorenübergreifende Zusammenarbeit, die Abläufe verbessert und dadurch auch Geld spart, sowie die Krankenkassen, die bei entsprechender Vertragsgestaltung ebenfalls von den finanziellen Vorteilen einer Integrierten Versorgungsform profitieren.

2.2 Beschreibung des Programms

Da nun die beiden Begriffe „Planspiel“ und „Integrierte Versorgungsform“ kurz erläutert worden sind, kann die Beschreibung des Programms erfolgen. Hierbei wird „INVENT“ zuerst in allgemeiner Form erklärt. Dabei sollen das Ziel des Spiels, die Spielinhalte und die Aufgaben eines Spielers beschrieben werden. Da das Spiel in zwei Varianten gespielt werden kann, wird auch auf deren Unterschiede eingegangen. Auf Grund der Verwendung von realen Daten in dem Planspiel, soll auf diese näher eingegangen werden. Dabei wird

unter anderem gezeigt, wie sämtliche Informationen, die für das Planspiel benötigt werden, aus diesen Daten hergeleitet wurden. Abschließend soll der Ablauf des Spiels dargestellt werden.

2.2.1 Allgemeine Beschreibung

„INVENT“ ist ein Planspiel, welches das Einführen und Etablieren einer oder zwei miteinander konkurrierender Integrierter Versorgungssysteme gemäß §140 a-h in Deutschland zum Inhalt hat. Da das Programm auf Echtdaten aus der Praxis basiert, kann es dazu verwendet werden, die Auswirkungen von bestimmten Strategien anhand von realen Daten und Gegebenheiten vor Ort aufzuzeigen. Eine mögliche Zielgruppe wären daher neben Wissenschaftlern mit eher allgemeinem Interesse auch Mitglieder des Vorstandes eines konkreten Krankenhauses, die die Einführung einer IV planen und vorab an einer Abschätzung der möglichen Folgen anhand von Echtdaten aus dem eigenen Krankenhaus interessiert sind.

Das Spiel kann in zwei unterschiedlichen Varianten gespielt werden. Die erste Variante kann nur von einem Spieler gespielt werden, dessen Aufgabe es ist, eine IV in einem konkurrenzfreien Umfeld zu etablieren. Dabei soll der Spieler die Grundvoraussetzungen für das Aufbauen und Fortbestehen einer IV lernen. Da Konkurrenz aber ein wichtiger Einflussfaktor ist, wurde zusätzlich eine Zweispielervariante implementiert, in der zwei IVs, jede durch einen Spieler repräsentiert, um die Gunst der Patienten, aber auch der niedergelassenen Ärzte werben. Im Folgenden soll das Planspiel an Hand der Einspielervariante näher beschrieben werden, anschließend wird auf die Zweispielervariante eingegangen.

Die integrierte Versorgungsform soll in einer Region aufgebaut werden, in der es eine bestimmte Anzahl von Krankenhäusern gibt. Diese Krankenhäuser schließen sich zu einer IV zusammen, welche von dem Spieler verwaltet werden soll. Des Weiteren wurden Arztgruppen gebildet, wobei jede Gruppe aus niedergelassenen Ärzten eines bestimmten Fachbereiches und eines bestimmten Teils der Region, z.B. eines Bezirkes, besteht. Die Arztgruppen, die vom Computer simuliert werden, „verkaufen“ sozusagen einen Teil ihrer Kapazität an die IV. Diese Kapazität kann dann für IV-Patienten genutzt werden. Je mehr die Kapazitäten sowohl hinsichtlich der Bezirke als auch hinsichtlich der Fachgebiete verteilt sind und je mehr Kapazität zur Verfügung steht, desto bessere Bedingungen finden die IV-Patienten vor, da sie unter mehreren verschiedenen Anbietern wählen können. Andererseits

muss darauf geachtet werden, dass nicht zu viel Kapazität gekauft wird, da der IV bei entsprechender Vertragsgestaltung dann Nachteile entstehen.

Durch die Ausgestaltung der Verträge, deren Auswirkungen später durch die Ameisen simuliert werden sollen, werden die Patienten zu bestimmten Krankenhäusern gelenkt, und konkretisieren damit die Strategie der Integrierten Versorgungsform. Patienten, die der Integrierten Versorgungsform angehören, werden also andere Wege durch das Gesundheitssystem nehmen, als die „normalen“ Patienten.

Im Planspiel INVENT sind 3 Strategien darstellbar, wobei auch gemischte Strategien angewendet werden können. Dabei zielt die erste auf die Gleichverteilung der Auslastung aller Krankenhäuser, die zweite auf die Minimierung der Patientenwege und die dritte Strategie soll gewährleisten, dass die Patienten in das für sie bestgeeignete Krankenhaus eingewiesen werden. Als vierte Strategie soll hier noch jene erwähnt werden, die den Gewinn der IV maximiert, da aber zum Zeitpunkt der Fertigstellung dieser Arbeit noch keine Echtdaten zum Implementieren dieser Strategie zur Verfügung standen, wird diese im weiteren Verlauf dieser Arbeit vernachlässigt. Im Planspiel wird davon ausgegangen, dass IV-Patienten über ihren finanziellen Bonus hinaus den Vorteil haben, dass sie bei hoher Auslastung zu einer höheren Wahrscheinlichkeit im jeweiligen Krankenhaus aufgenommen werden als Patienten, die keiner IV angehören.

Die Anwerbung neuer Teilnehmer der Integrierten Versorgungsform erfolgt im Planspiel primär über das Bereitstellen von günstigen Bedingungen für die potentiellen Patienten. Um die Angebote der Integrierten Versorgungsform bekannt zu machen und deren Vorteile zu präsentieren, kann auch in Werbung investiert werden. Um Spielentscheidungen zielgerichteter treffen zu können, soll dem Spieler auch ermöglicht werden, Informationen wie beispielsweise simulierte Marktforschungsergebnisse zu erkaufen. Ziel einer IV ist es, ihren Gewinn zu maximieren, die Zahl der teilnehmenden Patienten auszubauen und auch möglichst viele Ärzte für eine Zusammenarbeit zu gewinnen. Die jeweilige Gewichtung dieser einzelnen Faktoren wird durch die Spielleitung vor Spielbeginn festgelegt.

Bei der Zweispielervariante besteht lediglich der Unterschied, dass zwei Spieler gegeneinander spielen. Dies bedeutet, dass es zwei Integrierte Versorgungsformen in der Region gibt, wobei sich die gleichen Krankenhäuser auf zwei IVs aufteilen. Die Aufteilung der Krankenhäuser wird durch die Spielleitung vor Spielbeginn vorgenommen, die Spieler können lediglich wählen, welche IV sie vertreten möchten. In dieser Variante des Spiels

können die Patienten nicht nur entscheiden, ob sie einer IV beitreten wollen, sondern können auch noch zwischen den beiden IVs wählen. Auch die niedergelassenen Ärzte gehen von einer anderen Ausgangssituation aus, da sie ihre Kapazität an zwei Integrierte Versorgungsformen verkaufen können. Die Patienten werden wiederum in die Krankenhäuser ihrer IV weitergeleitet, kann ihre Krankheit in keinem der IV-Krankenhäuser behandelt werden, so werden sie in ein Krankenhaus der anderen IV eingewiesen, erhalten dort jedoch keine bevorzugte Behandlung.

2.2.2 Daten

Momentan liegen dem Programm Echtdaten aus einer größeren Stadt in Deutschland zu Grunde, die aus Datenschutzgründen in dieser Arbeit nicht genannt werden darf. Jedoch ist das Planspiel so flexibel programmiert, dass auch Daten einer anderen Region verwendet werden können. Dafür müssen lediglich einige Parameter verändert und die bereits bestehende Access-Datenbank mit den neuen Daten gefüllt werden. Im weiteren Verlauf dieser Arbeit und auch schon bei der Beschreibung des Programms wird jedoch von den momentan vorhandenen Daten ausgegangen.

Im Folgenden soll beschrieben werden, welche Daten zur Verfügung stehen und wie diese Daten für das Verwenden im Programm adaptiert werden mussten.

Vorhandene Daten

Bei den Daten handelt es sich um Patientendaten, die pro Patient, der in einem von sieben Krankenhäuser behandelt worden ist, aufgenommen wurden. Diese Daten erstrecken sich über das gesamte Jahr 2001, und beinhalten Informationen über ungefähr 150.000 Patienten. In den Daten sind aus Gründen des Datenschutzes keine Namen oder sonstige eindeutige Identifikationsmerkmale enthalten, jedoch die Adresse allerdings ohne Hausnummer, das Geburtsjahr und das Geschlecht wurden mitgeliefert. Des Weiteren ist das Aufnahmedatum und das Entlassungsdatum bekannt. Über den Krankenhausaufenthalt selbst wurden die festgestellte Diagnose und das Krankenhaus mit der Station, in der der Patienten behandelt wurde, angegeben, wobei die Diagnose letztlich überwiegend für die Zuteilung der Patienten entscheidend ist. Kinder kommen bei gleicher Diagnose selbstverständlich in die Kinderabteilung.

Datenaufbereitung

Da die Daten im Programm mittels einer Datenbank verwaltet werden, müssen die entsprechenden Datentabellen mit den zur Verfügung stehenden Daten vor dem Aufrufen

des Programms gefüllt werden. In manchen Fällen war eine Adaptierung der Daten oder sogar eine Herleitung aus den Daten erforderlich, um die nötigen Informationen in die Datenbank zu importieren und in weitere Folge in das Programm einbauen zu können. Die Werte für die drei im Planspiel maßgeblich zur Steuerung der Patientenpfade verwendeten Kriterien „Entfernung“, „Eignung“ und „Auslastung“ waren von diesen Bearbeitungen hauptsächlich betroffen. Im Folgenden sollen diese kurz beschrieben werden.

Da in den Rohdaten nur die Adressen der Patienten mitgeliefert wurden, vom Planspiel aber die Entfernungen der einzelnen Patienten zu den sieben Krankenhäusern gespeichert werden müssen, wurde ein Geoinformationssystem eingesetzt. Werden die Patientenadressen in ein solches System importiert und die Standorte der Krankenhäuser festgelegt, so können die Entfernungen jedes Patienten zu jedem Krankenhaus berechnet werden. Das Kriterium Entfernung im Planspiel ist jedoch nicht nur von der absoluten Anzahl an Kilometern abhängig, sondern auch von der durchschnittlichen Entfernung, die Patienten mit der gleichen Diagnose bereit sind, auf sich zu nehmen. Des Weiteren soll die Entfernung zum nähersten Krankenhaus berücksichtigt werden, da ein Patient, der im Extremfall gegenüber eines Krankenhauses wohnt, bevorzugt auch in dieses gehen möchte. Auf die exakte Berechnung der Entfernungswerte soll hier nicht näher eingegangen werden, als Ergebnisse resultieren jedoch Werte von 0 bis 100, die in der Datenbank gespeichert werden.

Des Weiteren muss die Eignung der jeweiligen Stationen in den Krankenhäusern für jede Diagnose zur Verfügung stehen. Derartige Daten wurden jedoch nicht mitgeliefert. Daher wird die Eignung im Planspiel auf Grund der schon behandelten Patienten - und damit auf Grund der Erfahrung im Haus bzw. auf der Station – errechnet. Um auf einen realistischen Eignungswert zu kommen, spielt jedoch nicht nur die Anzahl an Patienten, die in der jeweiligen Station mit der gleichen Diagnose schon behandelt wurden, eine Rolle, sondern auch die Erfahrung im gesamten Krankenhaus, da sich die Ärzte gegebenenfalls konsiliarisch untereinander austauschen können. Die Eignungswerte errechnen sich aus diesen beiden Faktoren, die jeweils mit einer Gewichtung versehen werden, wobei die Anzahl an schon behandelten Patienten höher gewichtet wird. Ergibt sich auf Grund dieser Berechnung, dass ein Krankenhaus für eine bestimmte Diagnose nicht geeignet ist, so wird nachgeprüft, in welchen Stationen die Diagnose in anderen Krankenhäusern behandelt wurde. Existiert eine Station mit der gleichen Fachrichtung als jene, wo die Diagnose schon behandelt wurde, im entsprechenden Krankenhaus, so wird trotz der Tatsache, dass die Diagnose laut Rohdaten im abgelaufenen Jahr nicht in dem Krankenhaus behandelt wurde,

ein geringer Eignungswert eingetragen, da es theoretisch möglich wäre, den Patienten auf besagter Station auch im eigenen Krankenhaus zu behandeln. Ein Eignungswert von Null schließt hingegen eine Behandlung und damit auch eine Einlieferung vollständig aus. Liegt der Eignungswert bei 100 so bedeutet dies, dass die Station am besten für die Krankheit geeignet ist. In der Datenbank werden jeweils nur jene Stationen samt ihren Eignungswerten gespeichert, die sich nach diesen Kriterien als am bestgeeigneten für die entsprechende Diagnose herausstellen. Dies bedeutet, dass ein Patient in ein Krankenhaus eingewiesen wird und dort immer in jener Station behandelt wird, die sich am besten in dem entsprechenden Krankenhaus für seine Diagnose eignet. Eine zweite Möglichkeit wäre, nicht nur die bestgeeignete Station sondern alle Stationen in einem Krankenhaus zu speichern. Dann könnte der Patient auch in verschiedene Stationen innerhalb eines Krankenhauses eingeliefert werden. Diese Möglichkeit ist aber im Hinblick auf einen größeren Rechenaufwand bislang im Planspiel nicht realisiert worden.

Um die Auslastungswerte berechnen zu können, muss bekannt sein, wie viele Betten pro Station vorhanden sind. Die Anzahl der Planbetten kann entweder aus dem Krankenhausplan des jeweiligen Bundeslandes oder aber von den die Daten liefernden Krankenhäusern direkt bezogen werden. Des Weiteren müssen die Anfangsauslastungen aller Stationen festgestellt werden. Sind diese nicht in den gelieferten Daten enthalten, so kann die Auslastung der Stationen am Ende des betreffenden Jahres näherungsweise als angenommene Anfangsauslastung herangezogen werden.

Um das Kriterium Auslastung realistisch abzubilden, wird eine untere und eine obere Grenze bestimmt. Diese Grenzwerte können sich für IV-Patienten und Patienten, die keiner IV angehören, unterscheiden. Unterschreitet die Auslastung einer bestimmten Station die untere Grenze, so entspricht der Auslastungswert 0, überschreitet er die obere Grenze, so wird er mit 100 festgelegt. Zwischen diesen beiden Grenzen erfolgt die Berechnung des Auslastungswertes im Rahmen dieser Arbeit durch eine lineare Funktion.

2.2.3 Spielablauf

Wird INVENT gestartet, so kann zwischen der Einspieler- und der Zweispielervariante gewählt werden. Bei letzterer muss zusätzlich ein gemeinsames Verzeichnis oder, falls kein Netzwerk vorhanden ist, ein Laufwerk angegeben werden, über das Daten zum zweiten Spieler ausgetauscht werden können. Zusätzlich muss angegeben werden, ob das Programm als „Spielführer“ agieren soll. Dazu gehören interne Aufgaben, wie zum Beispiel das Koordinieren der Periodenübergänge. Danach kann der Spieler die Strategie seiner IV,

nach der die IV-Patienten in die jeweiligen Krankenhäuser gelenkt werden sollen, angeben, wobei, wie oben schon erwähnt, auch gemischte Strategien zulässig sind. Sind diese Entscheidungen getroffen, so kann das Spiel gestartet werden. Abbildung 2.2 zeigt das Planspiel nach dem Start. Dabei soll aus Datenschutzgründen nicht jene Stadt dargestellt werden, von der die Daten stammen, sondern Wien.

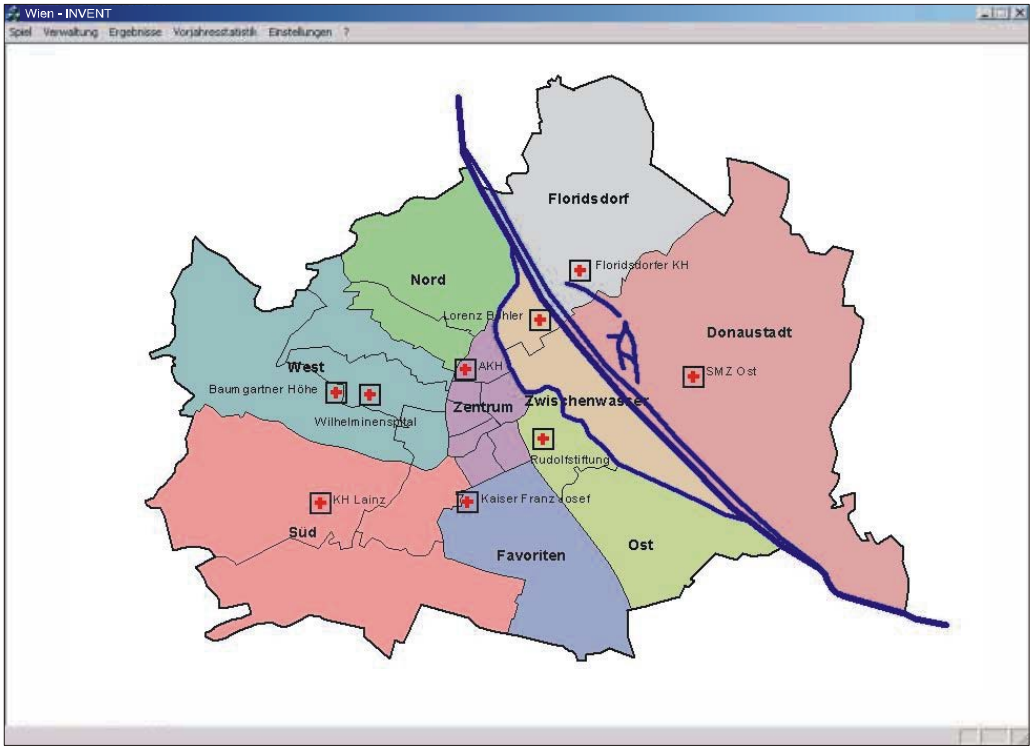

Abbildung 2.2: Darstellung des Planspiels INVENT nach Spielbeginn

Nun kann der Spieler seine Entscheidungen treffen. Dabei helfen ihm Zusatzinformationen aus einer Vorjahresstatistik, in der beispielsweise die Herkunft der Patienten oder die Auslastungen der Stationen im Zeitverlauf als Diagramm dargestellt werden. Aber auch allgemeine Informationen, wie beispielsweise die Anzahl an Ärzten, die einer bestimmten Arztgruppe zugehören, oder die Einwohnerzahl einer bestimmten Teilregion, können abgefragt werden. Sind Ergebnisse der Vorperiode des laufenden Spiels vorhanden, so können diese ebenfalls in ähnlicher Weise wie die Vorjahresstatistik angesehen werden, sind keine Ergebnisse vorhanden, so wird lediglich die Anfangssituation angezeigt.

Hat ein Spieler alle Entscheidungen getroffen, so teilt er dies dem Simulationsmodell über den Aufruf des Menüeintrages „Nächste Periode berechnen“ mit. Das Simulationsmodell hat einerseits die Aufgabe, zu errechnen, welche und wie viele Versicherungsnehmer im nächsten Monat den jeweiligen IVs angehören, und ist andererseits für die Einweisung der

Patienten in die Krankenhäuser zuständig. Für das Lösen der zweiten Aufgabe werden die Patienten in jene, die einer IV angehören, und jene, die keiner IV angehören, unterteilt. IV-Patienten sollen, wie zuvor schon erwähnt, entsprechend der Strategie der Integrierten Versorgungsform gelenkt werden. Diese Steuerung der Patienten soll als Optimierungsproblem von einem Ameisenalgorithmus gelöst werden. Existieren auf Grund der Zweispielervariante zwei Integrierte Versorgungsformen, so werden die Patienten hinsichtlich dieser unterschieden und beide Gruppen von Patienten getrennt von dem Algorithmus bearbeitet. Patienten, die keiner IV angehören, werden danach von dem Algorithmus zugeordnet, da diese aber nicht oder weniger von der Integrierten Versorgungsform gesteuert werden können, werden die Parameter des Algorithmus für diese Patienten anders gesetzt. Die Zuordnung der Patientengruppen nacheinander ist zulässig, da IV-Patienten bevorzugt in einer Station aufgenommen werden soll und daher Patienten, die nicht einer IV angehören, die Aufnahme eines IV-Patienten nicht beeinflussen.

Da sich diese Arbeit mit der Auswahl und der Implementierung eines solchen Algorithmus’ beschäftigt, soll das Programm hier nicht detaillierter beschrieben werden. Stattdessen soll im nächsten Kapitel auf Ameisenalgorithmen näher eingegangen und im darauffolgenden Kapitel gezeigt werden, wie einer dieser Algorithmen in dem Planspiel realisiert werden kann.

3 Ameisenalgorithmen

Dieses Kapitel soll eine detaillierte Einführung in die Theorie der Ameisenalgorithmen geben und zeigen, wie diese in der Praxis eingesetzt werden können. Dafür wird, nach einem kurzen, generellen Überblick, das Verhalten der natürlichen Ameisen bei der Futtersuche beschrieben und gezeigt, wie dieses Verhalten als Algorithmus programmiertechnisch dargestellt werden kann, sodass Optimierungsprobleme von der Art der Futtersuche der Ameisen gelöst werden können. Im Anschluss daran wird beschrieben, wie man diesen Algorithmus erweitern und verbessern kann, um bessere Lösungen mit geringerem Rechenaufwand zu finden. Ameisenalgorithmen können allerdings nicht nur Optimierungsprobleme ähnlich der Futtersuche lösen, sondern auf jedes beliebige, diskrete Optimierungsproblem angewendet werden. Besonders eignen sich Ameisenalgorithmen für sogenannte NP-harte Probleme, wobei diese auch nach mehreren Zielen optimiert werden können. Da natürliche Ameisen ständig auf ihre sich verändernde Umwelt reagieren müssen, liegt es nahe, dass Ameisenalgorithmen auch für Probleme eingesetzt werden können, deren Struktur sich während des Problemlösungsprozesses verändert. Auch verteilte Probleme, in denen das Problem innerhalb eines sogenannten verteilten Systems auftritt, können Ameisenalgorithmen auf Grund ihrer indirekten Kommunikationsart bewältigen. All diese Einsatzmöglichkeiten werden im Rahmen dieses Kapitels mit Beispielen beschrieben, und es wird gezeigt, wie man den Algorithmus anpassen muss, um die entsprechenden Probleme zu lösen. Abschließend werden die Vor- und Nachteile von Ameisenalgorithmen dargestellt und verdeutlicht, wie man einen Algorithmus auf eine bestimmte Problemstellung adaptiert.

3.1 Überblick

Das Heranziehen von Phänomenen der Natur ist eine weitgehend verbreitete Methode zum Lösen von komplexen Problemen. Schon 1960 wurde auf einem Kongress in Dayton/Ohio der Begriff „Bionik“ von J. E. Steele geprägt. Der Name „Bionik“ leitet sich von den Begriffen „Biologie“ und „Technik“ ab und stellt heute ein neues, interdisziplinäres Forschungsgebiet dar, das die Natur als Vorbild für technische Problemlösungen nimmt.[9] Bei dem technischen Problem, Muster zu erkennen, orientiert man sich beispielsweise an den menschlichen Gehirnzellen. Das Verfahren dazu nennt man Neuronale Netze. Ein weiteres Beispiel wären Genetische Algorithmen, die sich auf diverse Problemstellungen in unterschiedlichsten wissenschaftlichen Bereichen wie Technik oder auch Ökonomie anwenden lassen. Dabei

[9] Vgl. Homepage der TU Berlin, http://www.bionik.tu-berlin.de/kompetenznetz/bionik/bionik.html vom 10. 1. 2003.

wird die Evolution von Lebewesen auf das Überleben von guten Lösungen in einem Suchraum adaptiert, bei dem diejenigen Lösungen gefördert und weiterentwickelt werden, die den größten Erfolg erzielt haben.

Seit Anfang der 90er Jahre versucht man, das Verhalten von Insekten, im Speziellen von Ameisen, zum Lösen von komplexen Problemen heranzuziehen. Es gibt verschiedene Tätigkeiten von Ameisen, die für diesen Zweck genutzt werden können. Ameisen bringen beispielsweise eine so flexible Arbeitsteilung zu Stande, so dass trotz ständiger Veränderung des Aufwandes und der Priorität der zu erledigenden Aufgaben, die Arbeit nahezu optimal verteilt wird. Diese Fähigkeit kann für Problemlösungen bei der Arbeitsteilung im wirtschaftlichen Bereich genutzt werden. Eine weitere Tätigkeit der Ameisen, die man zum Beispiel zur Verbesserung von Datenanalysen verwenden kann, ist ihr Sortierverhalten, wenn sie ihre Larven positionieren, oder wenn sie eine Art „Friedhof“ für ihre Verstorbenen errichten. Auch der Nestbau der Ameisen ist interessant, da manche Ameisenkolonien komplexe und geometrisch präzise Bauten errichten können, obwohl die einzelnen Ameisen keinen fertigen „Bauplan“ für diese Bauten in sich tragen. Die Tatsache, dass es Ameisen schaffen, sich so zu koordinieren, dass sie Futter in einer Gruppe zu ihrem Nest transportieren können, kann ebenso genutzt werden. Das für diese Arbeit wichtigste Verhalten ist allerdings jenes, bei dem Ameisen Futter suchen. Dabei schaffen sie es, in kürzester Zeit den besten Weg in Hinblick auf Länge und auch Sicherheit von ihrem Nest zur Futterstelle zu finden. Auf Grund dieses Verhaltens wurde der Basisalgorithmus „Ant System“ (AS) und diverse, auf AS aufbauende Erweiterungen und Verbesserungen entwickelt.

Diese Erweiterungen und Verbesserungen und auch AS selbst können unter dem Oberbegriff „Ant Colony Optimization“ (ACO)-Algorithmus zusammengefasst werden. Der Begriff „Ant Colony Optimization“ wurde erst 1999 von Dorigo und Di Caro eingeführt. In ihrem Artikel „The Ant Colony Optimization Meta-Heuristic“, der in dem Sammelwerk „New Ideas in Optimization“ veröffentlicht wurde, definieren sie ACO als eine Metaheuristik, die sich am Verhalten der Ameisen bei der Futtersuche orientiert.[10] Diese stellt eine allgemeine Beschreibung dar, die jeden Ameisenalgorithmus, der seit Beginn der 90er Jahre entwickelt wurde und sich mit dem Auffinden von Futter beschäftigt, als ACO-Algorithmus definiert. ACO ist außerdem so generell formuliert, dass man schon mit geringfügigen Änderungen an einem ACO-Algorithmus verschiedene Optimierungsprobleme lösen kann.

[10] Vgl. Dorigo, M. und Di Caro, G. (1999) S. 11.

Da in dieser Arbeit das Verhalten der Ameisen bei der Futtersuche von vorrangiger Bedeutung ist, wird im Folgenden vorwiegend auf dieses Verhalten eingegangen.

3.2 Natürliche Ameisen

Ameisen sind relativ einfache Individuen mit geringen kognitiven Fähigkeiten. Alleine können sie nicht viel bewirken, in der Gruppe können sie sich allerdings so gut organisieren, dass sie komplexe Probleme wie zum Beispiel das Suchen nach Futter schnell lösen können.

Diese Fähigkeit beruht, wie Bonabeau, Dorigo und Theraulaz in ihrem Buch „Swarm Intelligence: From Nature to Artificial Systems“ beschreiben, auf der Selbstorganisation der Kolonie.[11] Es gibt kein Individuum, das anderen Befehle erteilt, sondern jede Ameise entscheidet dezentral. Die Kommunikation läuft dabei meist über eine chemische Substanz, sogenannten Pheromone, ab.[12] Viele Ameisenarten befolgen das „trail-laying trail-following“-Verhalten[13], bei dem jede Ameise ihren Weg mit Pheromonen markiert und somit eine Pheromonspur bildet oder die vorhandene verstärkt. Andere Ameisen können dadurch aus den Erfahrungen der vorherigen Ameisen Nutzen ziehen, da sie die Pheromonspur erkennen und die Wege mit einer Wahrscheinlichkeit proportional zu der darauf befindenden Menge an Pheromonen wählen. Durch die verschiedenen Mengen an Pheromon auf den Wegen bildet sich so zu sagen ein „globales Gedächtnis“, durch welches jede Ameise weiß, welche Wege schon wie oft erfolgreich gegangen worden sind. Pheromone können nicht nur verstärkt werden, sie haben auch die Eigenschaft, dass sie mit der Zeit verdampfen, was bedeutet, dass Wege, die nicht oder nicht mehr oft erfolgreich benutzt werden, aus dem „globalen Gedächtnis“ der Ameisenkolonie gestrichen und somit vergessen werden. Ameisen folgen jedoch nicht immer streng der Pheromonspur, es existieren auch Ameisen, die neue Wege erforschen. Dies führt dazu, dass die bisherigen Wege immer weiter verbessert werden.

Wie Ameisen den kürzesten Weg von ihrem Nest zu einer Futterstelle finden können, soll im Folgenden genauer beschrieben werden. Es sei eine Ausgangssituation wie in Abbildung 3.1 gegeben, bei der den Ameisen zwei Wege vom Nest zur Futterstelle zur Auswahl stehen, wobei der eine Weg deutlich kürzer ist als der andere. Die Ameisen verlassen nacheinander das Nest und wählen anfangs zufällig einen Weg, da noch keine Pheromonspur vorhanden ist. Jede Ameise markiert dabei ihren Weg mit einer Pheromonspur. Die Ameisen, die den kürzeren Weg gewählt haben, sind schneller bei der Futterstelle und treten damit auch

[11] Vgl. Bonabeau, E. / Dorigo, M. / Theraulaz, G. (1999) S. 9 ff.
[12] Vgl. Bonabeau, E. / Dorigo, M. / Theraulaz, G. (1999) S. 26.
[13] Vgl. Hölldobler, B. und Willson, E. O. (1990), S. 265 ff.

schneller den Rückweg an, bei dem sie wieder Pheromone streuen. Kommen danach die ersten Ameisen, die den längeren Weg gewählt haben, zur Futterstelle und müssen sich für einen Rückweg entscheiden, so befinden sich am kürzeren Weg mehr Pheromone als am längeren. Da die Ameisen ihren Weg proportional zur Menge der Pheromone, die auf dem Weg positioniert sind, wählen, wird der kürzere Weg mit einer höheren Wahrscheinlichkeit gewählt. Auch nachfolgende Ameisen, die sich beim Nest für einen Weg entscheiden müssen, werden den kürzeren Weg mit einer höheren Wahrscheinlichkeit wählen, sobald die ersten Ameisen von der Futterstelle zurückgekehrt sind. Je mehr Ameisen den kürzeren Weg wählen, desto mehr Pheromone werden darauf gestreut, was wieder andere Ameisen dazu animiert, diesen Weg mit einer größeren Wahrscheinlichkeit zu wählen. Dieses Experiment wurde mit einer Kolonie von Linepithema humile (früher Iridomyrmex humilis) Ameisen und einer etwas komplizierteren Pfadstruktur in einem Artikel von Goss[14] beschrieben und führte zu den gleichen Erkenntnissen.

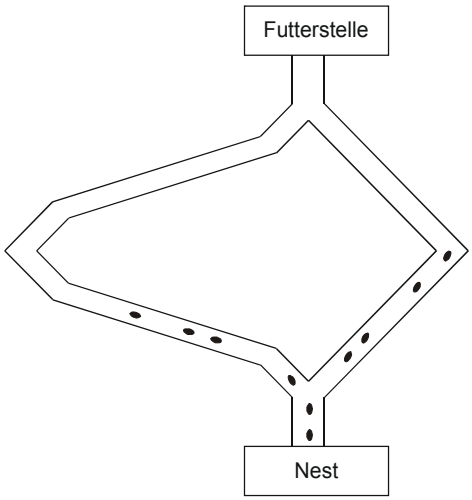

Abbildung 3.1: Linepithema humile Ameisen bei der Futtersuche:
Linepithema humile Ameisen suchen in einem künstlich angelegtem Wegenetz, in dem zwei unterschiedlich lange Wege existieren, nach dem kürzesten Weg vom Nest zur Futterstelle.
(Quelle: In Anlehnung an Goss, 1989, S. 579, der ein Experiment mit einer etwas komplexeren Pfadstruktur durchgeführt hat.)

In einem zweiten Experiment zeigt Deneubourg[15] wie sich die gleiche Ameisenart verhält, wenn die beiden Wege gleich lang sind. Es stellt sich heraus, dass die beiden Wege nicht gleichwertig benutzt werden, sondern einer bevorzugt wird. Das erklärt sich dadurch, dass die Ameisen zu Beginn, wenn noch keine Pheromonspuren vorhanden sind, ihren Weg zufällig wählen. Benutzen anfangs mehr Ameisen den linken Weg, so wird auch die

[14] Vgl. Goss, S. et al., *Naturwissenschaften* 76 (1989).
[15] Vgl. Deneubourg, J.-L. et al., *Journal of Insect Behavior* 3 (1990).

Pheromonspur dieses Weges stärker, was wiederum impliziert, dass mehr Ameisen den linken Weg wählen. Es wird auch gezeigt, dass es nicht unbedingt auf die Anfangsverteilung ankommt, sondern dass das Durchsetzen eines Weges bis zu einer bestimmten Pheromonstärke noch wechseln kann.

In einem weiteren Experiment, das jedoch nichts mit der Futtersuche zu tun hat, zeigt Aron[16] wie sich Linepithema humile Ameisen zwischen drei Nestern bewegen. Die Nester sind wie in Abbildung 3.2 an den Ecken eines gleichseitigen Dreiecks positioniert, das heißt, alle drei Wege sind gleich lang. Nach zwei Wochen wurden nur noch die Wege B und C genutzt, Weg A kaum noch. Die Ameisen haben also die Verbindungswege der einzelnen Nester auf ein Minimum beschränkt und damit das Problem des minimalen Spannbaumes gelöst. Das Ziel, den kürzesten Weg von einem Nest zum anderen zu finden, ist hier nicht gegeben, jedoch kann man aus dem Experiment weitere Schlüsse über die Wichtigkeit der Pheromone und die Flexibilität des Verhaltens der Ameisen ziehen, die sich auch auf die Futtersuche übertragen lassen. Um zu beweisen, dass sich Ameisen hauptsächlich an den Pheromonspuren orientieren und nicht an visuellen Merkmalen, hat Aron das Dreieck um 120 Grad gedreht. Wie erwartet ergab diese Änderung keine Veränderung an den genutzten Wegen. Weg B und Weg C wurden weiterhin stark genutzt, Weg A kaum. Die Flexibilität der Ameisen wurde dadurch bewiesen, dass Aron den meist frequentierten Weg (Weg B) gesperrt hat, was zur Folge hatte, dass Weg A wieder benutzt wurde.

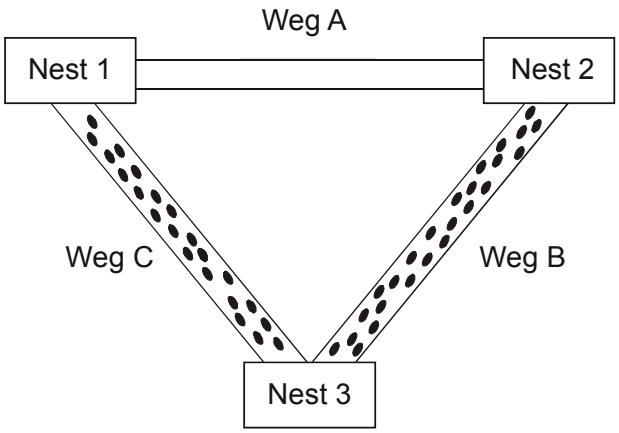

Abbildung 3.2: Das Verhalten von Limepithema humile Ameisen innerhalb ihres Nestes: Das Experiment von Aron zeigt, dass Limepithema humile Ameisen nur zwei von den drei vorhandenen Wegen nutzen und damit einen minimalen Spannbaum darstellen.
(Quelle: In Anlehnung an Aron, S. et al., 1990, S. 537)

[16] Vgl. Aron, S. et al. (1990).

Das Studieren des Verhaltens von sozialen Insektenkolonien, insbesondere von Ameisen, aber auch von Bienen und Wespen, ist von großen Interesse für die Computerwissenschaft. Die Tatsache, dass einfache Wesen in der Gemeinschaft komplexe Probleme lösen können, die sie alleine nicht einmal annähernd bewältigen können, kann für ein effizientes Lösen von Optimierungsproblemen verwendet werden.

3.3 Algorithmen mit künstlichen Ameisen

Es gibt viele Algorithmen, die sich am Verhalten der Ameisen orientieren. In dieser Arbeit sind jedoch nur diejenigen von Interesse, die sich mit dem Verhalten bei der Futtersuche beschäftigen. Der erste Algorithmus, der basierend auf diesem Verhalten entwickelt wurde, war Ant System, bei dem die Kommunikation zwischen den Ameisen, wie zuvor beschrieben, auf der Basis von Pheromonen abläuft. In weiterer Folge wurde dieser Algorithmus erweitert und verbessert, teilweise auch auf spezielle Optimierungsprobleme zugeschnitten. Dadurch entstanden Algorithmen wie zum Beispiel Ant Colony System (ACS), Hybrid Ant System (HAS) und MAX-MIN Ant System (MMAS).

Nicht alle Algorithmen verwenden Pheromone als Kommunikationsmittel. In einem Artikel von Monmarché[17] wird ein Algorithmus beschrieben, der sich an dem Verhalten von Pachycoldyla apicalis Ameisen orientiert. Diese Ameisenkolonie kommuniziert über visuelle Orientierungspunkte anstatt über Pheromone. Der Algorithmus, genannt API, hat jedoch sein Hauptanwendungsgebiet in numerischen Optimierungsproblemen und ist daher für diese Arbeit von geringerem Interesse.

3.3.1 Ant System

Ant System wurde 1991 von Colorni, Dorigo und Maniezzo entwickelt und in deren Artikel „Distributed optimization by ant colonies"[18] erstmals präsentiert. Da Bonabeau, Dorigo und Theraulaz in ihrem Buch „Swarm Intelligence: From Nature to Artificial Systems"[19] unter anderem einen leicht nachvollziehbaren Einblick in Ant System geben, soll im Folgenden darauf Bezug genommen werden.

Der Algorithmus von Ant System realisiert das bereits in Kapitel 3.2 dargestellte und – wie schon beschrieben - von Goss[20] experimentell bewiesene Verhalten von Ameisen bei der

[17] Vgl. Monmarché, N. / Venturini, G. / Slimane, M., *Future Generation Computer Systems* 16 (2000).
[18] Vgl. Colorni, A. / Dorigo, M. / Maniezzo, V. (1991).
[19] Vgl. Bonabeau, E. / Dorigo, M. / Theraulaz, G. (1999) S. 41 ff.
[20] Vgl. Goss, S. et al., *Naturwissenschaften* 76 (1989).

Futtersuche. Um das Problem etwas zu vereinfachen, wird im Folgenden von einem Modell ausgegangen, das bestimmte, vordefinierte Wege vom Nest zur Futterstelle vorsieht. Abgebildet wird dies durch einen Graphen, dessen Kanten Teilstrecken eines Weges darstellen und dessen Knoten diese Teilstrecken begrenzen. Abbildung 3.3 soll den Graphen zu dem Experiment von Goss, das in Abbildung 3.1 dargestellt wurde, zeigen. Da nicht jeder Knoten mit jedem verbunden ist, handelt es sich hierbei um einen unvollständigen Graphen. Die Distanz zwischen zwei Knoten i und j wird mit d_{ij} bezeichnet und stellt die Gewichtung der einzelnen Kanten dar. Des Weiteren kann unterschieden werden, ob der Graph gerichtet oder nicht gerichtet ist. Beim letzteren Fall entspricht die Distanz von Knoten i zu Knoten j immer der Distanz des umgekehrten Weges, im ersteren Fall können hier unterschiedliche Werte auftreten. Ant System kann Probleme beider Fälle lösen, der Einfachheit halber wird aber im Folgenden der nicht gerichtete Fall angenommen.

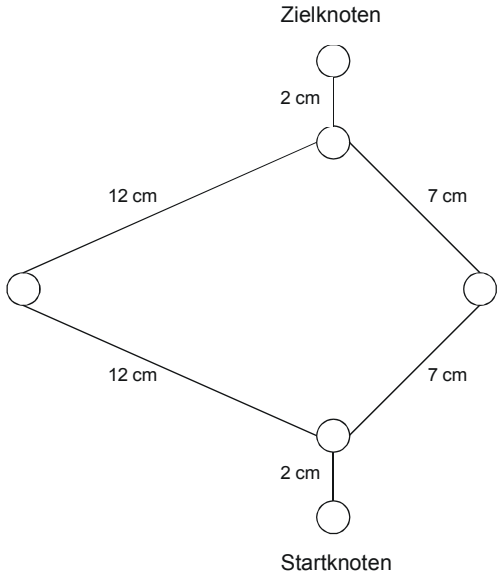

Abbildung 3.3: Darstellung des Problems der Futtersuche als Graphen

AS beschränkt sich darauf, dass es eine gewisse Anzahl m an Ameisen gibt, die eine Tour vom Nest zur Futterstelle machen. Eine Iteration endet dann, wenn alle Ameisen ihre Tour beendet haben. Mit der Variable t_{max} wird festgelegt, wie viele Iterationen durchgeführt werden, ehe der Algorithmus beendet wird.

Übergangsregel

Für jede Ameise in jeder Iteration hängt der Übergang von Knoten i zu Knoten j von drei Faktoren ab:

1. Eine Ameise soll keine zyklischen Wege gehen, dass heißt, der Knoten j darf in der gleichen Tour noch nicht besucht worden sein. Um diese Einschränkung zu implementieren, existiert für jede Ameise k und jeden Knoten i eine Liste J_i^k. Diese Liste enthält alle Knoten, die von der Ameise k in der momentanen Iteration von Knoten i aus noch nicht besucht worden sind. Nach jedem Übergang muss die entsprechende Liste aktualisiert werden, und am Anfang jeder Iteration müssen alle Listen mit den jeweils erreichbaren Knoten, mit Ausnahme des Startknotens, neu gefüllt werden.

2. Befindet sich eine Ameise in Knoten i, so existiert eine lokale Information, welche darlegt, welcher Knoten j aus lokaler Sicht am besten für einen Übergang geeignet ist. Diese Information wird als „heuristic desirability“ ($\eta_{i,j}$) bezeichnet, und entspricht in diesem Modell dem inversen Wert der Distanz $1 / d_{ij}$. Da sich die Distanz der Wege im Laufe des Problemlösungsprozesses nicht verändert, kann man die Information als statisch bezeichnen.

3. Neben lokaler Information gibt es auch globale Information, die in Form der Pheromonmenge $\tau_{ij}(t)$ der Kante (i, j) in der Tour t die sogenannte „learned desirability“ darstellt. Diese Information kann in einem globalen Zusammenhang gesehen werden und verändert sich während des Problemlösungsprozesses. Sie repräsentiert die bisher erworbenen Erfahrungen der Ameisen.

Die Wahrscheinlichkeit, mit der eine Ameise k in ihrer t-ten Tour von Knoten i zu Knoten j wechselt, kann mittels einer Übergangsregel errechnet werden. Diese lautet:

$$p_{ij}^k(t) = \frac{\left[\tau_{ij}(t)\right]^\alpha \cdot \left[\eta_{ij}\right]^\beta}{\sum_{l \in J_i^k} \left(\left[\tau_{il}(t)\right]^\alpha \cdot \left[\eta_{il}\right]^\beta\right)} \tag{3.1}$$

Diese Formel kann nur dann angewendet werden, wenn j in der Liste J_i^k enthalten ist, andernfalls ist die Wahrscheinlichkeit p_{ij}^k gleich Null. Die Variablen α und β können im Algorithmus frei gewählt werden und stellen die Einflussgröße von lokaler und globaler Information dar. Ist $\alpha = 0$, so fallen sämtliche Erfahrungswerte, die durch Pheromonspuren repräsentiert werden, weg, und der Algorithmus wird zu einem klassisch stochastischen Algorithmus. Ist hingegen $\beta = 0$, so werden nur noch Erfahrungswerte für die Entscheidung über den nächsten Knoten herangezogen, und die lokale Information wird vollkommen vernachlässigt. Dies würde dazu führen, dass jene Tour als Lösung resultiert, die am häufigsten benutzt worden ist, was aber nicht bedeutet, dass diese Tour auch die optimale

Lösung darstellt. Touren, die anfangs nicht benutzt werden, werden somit endgültig vernachlässigt. Das in Kapitel 3.2 bei der Beschreibung der natürlichen Ameisen dargestellte Experiment von Aron[21] zeigt, obwohl damit nicht das Verhalten der Futtersuche darstellt wird, die möglichen Konsequenzen. Fällt von den zwei benutzten Pfaden einer aus, so können sich die Ameisen an den dritten, später bekanntlich nicht mehr genutzten, Pfad nicht mehr erinnern. Daher ist es wichtig, ein für den Problemlösungsprozess geeignetes Verhältnis zwischen α und β zu finden.

Durch diese Übergangsregel wird nicht immer jener Knoten, der im Hinblick auf lokale und globale Information als der beste erscheint, gewählt, sondern es werden auf Grund der Wahrscheinlichkeitsverteilung auch Knoten mit auf den ersten Blick niedriger Qualität ausgewählt, wodurch es den Ameisen möglich wird, neue, auf die gesamte Tour bezogen jedoch eventuell bessere Wege zu finden.

Pheromon-Update-Regel

Jede Ameise, die ihre Tour beendet hat, verstärkt jene Kanten (i, j), die sie benutzt hat, um eine gewisse Pheromonmenge $\Delta\tau_{ij}^{k}(t)$, die von der Qualität der Lösung abhängt:

$$\Delta\tau_{ij}^{k}(t) = Q / L^{k}(t), \tag{3.2}$$

wobei $L^{k}(t)$ der Länge der Tour t der Ameise k entspricht und Q ein Parameter ist, dessen Wert möglichst nahe an der optimalen Tourlänge liegen sollte. Deshalb wird Q vorab mittels einer heuristischen Methode berechnet. Ist eine Kante (i, j) nicht in der Tour t der Ameise k enthalten, so ist $\Delta\tau_{ij}^{k}(t) = 0$.

Um die Erforschung neuer Wege zu fördern und die Wahrscheinlichkeit der Stagnation in einer nicht optimalen Lösung zu minimieren, verdunstet ein gewisser Faktor ρ der Pheromone nach jeder vollendeten Iteration.

Die Pheromon-Update-Regel lautet daher:

$$\tau_{ij}(t) \leftarrow (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij}(t), \tag{3.3}$$

wobei ρ einen Wert zwischen 0 und 1 annimmt. $\Delta\tau_{ij}(t)$ ergibt sich seinerseits aus der Formel:

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t) \tag{3.4}$$

[21] Vgl. Aron, S. et al. (1990).

Initialisiert werden die einzelnen Kanten mit einem kleinen, positiven Wert τ_0, was impliziert, dass zum Zeitpunkt $t = 0$ eine homogene Verteilung der Pheromone vorliegt.

Der Algorithmus

/* Initialisierung */
For each Kante (i, j)
    $\tau_{ij} = \tau_0$
End For
$L^+ = -1$ //es gibt noch keine minimale Tourlänge

/* Hauptschleife */
For t = 1 **to** t_{max}
    Fülle die Liste J_i^k für alle Ameisen k und alle Knoten i

    /* Für jede Ameise eine Tour finden */
    For k = 1 **to** m
        Knoten i = Anfangsknoten
        T^k besteht nur aus dem Anfangsknoten
        While Knoten i **not** = Endknoten
            Wähle den nächsten Knoten j mit der Wahrscheinlichkeit

$$p_{ij}^k = \frac{\left[\tau_{ij}\right]^\alpha \cdot \left[\eta_{ij}\right]^\beta}{\sum_{l \in J_i^k} \left(\left[\tau_{il}\right]^\alpha \cdot \left[\eta_{il}\right]^\beta\right)},$$

            wobei der Knoten j in J_i^k enthalten sein muss.

            Aktualisiere die Liste J_i^k

            Füge den Knoten j zu T^k hinzu
            Knoten i = Knoten j
        End While
    End For

    /* Tourlänge berechnen und kürzeste Tour bzw. deren Länge auf T^+ und L^+ speichern*/
    For k = 1 **to** m
        Berechne die Tourlänge L^k der Tour T^k der Ameise k
        If $L^k < L^+$ oder $L^+ = -1$ **then**
            Aktualisiere T^+ und L^+
        End if
    End For

    /* Pheromonspur aktualisieren */
    For each Kante (i, j)
        Berechne die neue Menge an Pheromonen laut der Pheromon-Update-Regel:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \Delta\tau_{ij},$$

        wobei

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

und

$$\Delta\tau_{ij}^{k} = Q / L^{k},$$

wenn die Kante (i, j) in der Tour T^k enthalten ist, andernfalls ist $\Delta\tau_{ij}^{k}$ = 0.

End for

End for

Print die kürzeste Tour T^+ mit ihrer Länge L^+

Abbildung 3.4: Darstellung des AS-Algorithmus' für die Futtersuche.
(Quelle: In Anlehnung an Bonabeau, Dorigo und Theraulaz, 1999, S. 45)

Der Algorithmus beginnt mit einer Initialisierungsschleife, in der jeder Kante die gleiche Menge an Pheromonen zugewiesen wird. Des Weiteren wird L^+ auf –1 gesetzt, um darzustellen, dass es momentan noch keine minimale Tour gibt. Daraufhin folgt die Hauptschleife, die jeweils eine Iteration repräsentiert und t_{max}-mal durchgeführt wird. In dieser Schleife suchen zunächst alle Ameisen mittels der Übergangsregel ihre Tour und speichern diese jeweils in T^k. Danach werden die Längen aller Touren berechnet und, falls nötig, die momentan kürzeste Tour T^+ aktualisiert. Anschließend wird auf jede Kante die Pheromon-Update-Regel angewandt, und die Schleife von neuem gestartet. Nach Beenden der Schleife wird die kürzeste Tour T^+ mit ihrer Länge L^+ ausgegeben.

Bei sämtlichen Variablen wurde im Algorithmus auf den Zusatz der Iteration t verzichtet, da dies sonst bedeuten würde, dass die Werte jeder Iteration gespeichert werden. Da es aber gerade für diese Art von Algorithmen wichtig ist, Speicherplatz und Rechenaufwand zu sparen, um schneller zu einer guten Lösung zu kommen, werden immer nur die aktuellen Werte gespeichert.

3.3.2 Ant Colony System

AS liefert eine gute Lösung in annehmbarer Zeit für kleindimensionierte Probleme, erhöht man jedoch die Anzahl der Knoten und Kanten, so erzielt AS meist schlechtere Ergebnisse als beispielsweise Genetische Algorithmen oder Simulated Annealing. Ant Colony System verwendet den AS-Algorithmus als Basis, erweitert und verbessert diesen jedoch, sodass er in Bezug auf die zuvor genannten Heuristiken konkurrenzfähig wird. Hierfür wird die Übergangsregel verändert, und die Pheromon-Update-Regel wird pro Iteration nur von jener Ameise durchgeführt, die die bisher beste Tour gefunden hat. Hingegen wird die

Pheromonspur einer Kante jedes Mal, wenn eine Ameise diese benutzt, um einen gewissen Anteil verringert. Bei sehr großen Optimierungsproblemen kann auch für jeden Knoten eine Kandidatenliste gespeichert werden. Im Folgenden sollen diese Erweiterungen detaillierter beschrieben werden.

Übergangsregel

Während die Übergangsregel bei AS durch die Wahrscheinlichkeitsverteilung immer die Möglichkeit zum Erforschen neuer Wege offen lässt, gibt es in der Übergangsregel von ACS zwei mögliche Vorgehensweisen. Eine davon ist, das bisher erworbene Wissen zu nutzen und genau den Knoten j zu wählen, dessen lokale und globale Information gemeinsam den größten Nutzen erbringen. Damit wird jedoch Forschung verhindert. Die zweite mögliche Vorgehensweise ist der Übergangsregel von AS sehr ähnlich, da auch hier die Wahrscheinlichkeit des Auswählens proportional zum Nutzen der lokalen und globalen Information steigt und somit das Erforschen neuer Wege ermöglicht wird. Die beiden Methoden werden jedoch nicht gleichzeitig zu Rate gezogen, sondern es wird zufällig bestimmt, welche der beiden gewählt wird, wobei die Häufigkeitsverteilung durch den Parameter q_0 festgelegt werden kann. Formal lautet die Übergangsregel somit:

$$j = \begin{cases} \arg\max_{u \in J_i^k} \left\{ [\tau_{iu}(t)] \cdot [\eta_{iu}]^\beta \right\} & \text{wenn } q \le q_0 \\ \Psi & \text{wenn } q > q_0 \end{cases} \tag{3.5}$$

wobei q eine Zufallszahl, gleichverteilt im Bereich von 0 bis 1, und q_0 ein fixer Parameter zwischen 0 und 1 ist. Die Zufallszahl q wird nach jedem Übergang neu bestimmt, der Parameter q_0 wird am Anfang des Algorithmus' festgelegt. Ψ ist ein Knoten, der zufällig mittels der Wahrscheinlichkeit

$$p_{i\Psi}^k(t) = \frac{[\tau_{i\Psi}(t)] \cdot [\eta_{i\Psi}]^\beta}{\sum_{l \in J_i^k} \left([\tau_{il}(t)] \cdot [\eta_{il}]^\beta \right)} \tag{3.6}$$

ausgewählt wird.

Wie aus Formel 3.5 hervorgeht, wird, wenn q größer als q_0 ist, ähnlich der Übergangsregel in AS, die Erforschung neuer Wege unterstützt. Ist q allerdings kleiner oder gleich q_0, so wird das bisher erworbene Wissen ohne jegliche Forschung genützt. Wird q_0 auf einen Wert nahe 1 gesetzt, so wird wenig geforscht und die bisherigen Ergebnisse mehr genutzt. Das führt dazu, dass die Ameisen mit einer sehr geringen Wahrscheinlichkeit ein globales Optimum finden, da sie, sobald sie ein lokales Optimum gefunden haben, diesen Weg benutzen und kaum mehr nach einem anderen Weg forschen bzw. sich dieser, falls er gefunden wird, nicht

durchsetzen kann. Entspricht hingegen q_0 einem Wert nahe 0, so werden sämtliche lokalen Optima durchsucht, was für Probleme von großem Umfang zu zeitaufwendig sein kann.

Pheromon-Update-Regel

Im Gegensatz zu AS, wo alle Ameisen eine gewisse Menge an Pheromonen, abhängig von der Qualität ihrer Tour, streuen, darf in ACS nur jene Ameise eine Pheromonspur legen, die die bisher beste Lösung gefunden hat. Dies impliziert, dass in jeder Iteration genau eine Ameise eine Pheromonspur legt, und dass nicht alle Kanten bestreut werden, sondern nur jene, die in dieser besten Tour enthalten sind. Das hat zur Folge, dass die Ameisen vorwiegend in der Nähe der bisher besten Tour nach neuen Lösungen suchen.

Die Pheromon-Update-Regel lautet somit:

$$\tau_{ij}(t) \leftarrow (1-\rho)\cdot\tau_{ij}(t) + \rho\cdot\Delta\tau_{ij}(t), \tag{3.7}$$

wobei die Kante (i, j) zu der bisher besten Tour T^+ gehören muss, ρ der Faktor der Pheromonverdampfung pro Iteration ist und

$$\Delta\tau_{ij}(t) = 1/L^+, \tag{3.8}$$

wobei L^+ die Länge der Tour T^+ bezeichnet.

Lokale Pheromon-Update-Regel

Zusätzlich zum Verstärken der bisher besten Tour wird jedes Mal, wenn eine Ameise von Knoten i zu Knoten j wechselt, die Pheromonmenge der Kante (i, j) entsprechend der folgenden Formel verringert:

$$\tau_{ij}(t) \leftarrow (1-\rho)\cdot\tau_{ij}(t) + \rho\cdot\tau_0 \tag{3.9}$$

Dies hat zur Folge, dass diese Kante für die nachfolgenden Ameisen weniger attraktiv wird, und die Wahrscheinlichkeit steigt, dass die Ameisen einen anderen Weg erforschen. Ohne diese Regel würden alle Ameisen in einer sehr engen Umgebung der besten Tour nach einer neuen Lösung suchen, und es steigt somit das Risiko, dass der Algorithmus in einem lokalen Optimum stagniert. Durch das Senken der Pheromonmengen der gerade besuchten Kanten wird der Suchraum ausgeweitet und somit die Möglichkeit des Auffindens neuer Wege geboten.

Verwenden von Kandidatenlisten

Bei umfangreichen Problemen kann eine Kandidatenliste zur Beschleunigung des Algorithmus' verwendet werden. Dabei bekommt jeder Knoten eine Kandidatenliste, in der

die cl nähest gelegenen Knoten abgespeichert sind. Der Parameter cl muss anfangs festgelegt werden und ist für jeden Knoten gleich groß.

Steht eine Ameise k auf dem Knoten i, so berücksichtigt sie für den Übergang zum nächsten Knoten j nur jene Knoten, die in der Kandidatenliste enthalten sind. Existieren in der Kandidatenliste nur noch Knoten, die schon besucht wurden, so wird jener Knoten aus der Menge der noch nicht besuchten Knoten gewählt, der die geringste Distanz zum Knoten i aufweist.

Der Algorithmus

/* Initialisierung */
For each Kante (i, j)
$\tau_{ij} = \tau_0$
End For
$L^+ = -1$ //es gibt noch keine minimale Tourlänge
Fülle die Kandidatenlisten mit cl Knoten für alle Knoten

/* Hauptschleife */
For t = 1 **to** t_{max}
Fülle die Liste J_i^k für alle Ameisen k und alle Knoten i

/* Für jede Ameise eine Tour finden */
For k = 1 **to** m
Knoten i = Anfangsknoten
T^k besteht nur aus dem Anfangsknoten

While Knoten i **not** = Endknoten
If wenigstens ein Knoten j in der Kandidatenliste von i enthalten ist, der noch nicht besucht wurde **then**

Errechne eine Zufallszahl q

Wähle den nächsten Knoten j aus den Knoten der Kandidatenliste mittels der Formel:

$$j = \begin{cases} \arg\max_{u \in J_i^k} \left\{ [\tau_{iu}] \cdot [\eta_{iu}]^\beta \right\} & \text{wenn } q \le q_0 \\ \Psi & \text{wenn } q > q_0 \end{cases}$$

wobei $\Psi \in J_i^k$ und mittels der Wahrscheinlichkeit

$$p_{i\Psi}^k = \frac{[\tau_{i\Psi}] \cdot [\eta_{i\Psi}]^\beta}{\sum_{l \in J_i^k} \left([\tau_{il}] \cdot [\eta_{il}]^\beta \right)}$$

gewählt wird.

Else
Wähle den nähest gelegenen Knoten aus J_i^k als Knoten j
End If

Berechne die neue Menge an Pheromonen laut der lokalen Pheromon-Update-Regel:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \rho\cdot\tau_0$$

Aktualisiere die Liste J_i^k
Aktualisiere die Kandidatenliste von Knoten i
Füge den Knoten j in T^k hinzu
Knoten i = Knoten j

End While
End For

/* Tourlänge berechnen und kürzeste Tour bzw. deren Länge auf T^+ und L^+ speichern*/
For k = 1 **to** m
Berechne die Tourlänge L^k der Tour T^k der Ameise k
If $L^k < L^+$ oder $L^+ = -1$ **then**
Aktualisiere T^+ und L^+
End if
End For

/* Pheromonspur updaten */
For each Kante (i, j) $\in T^+$
Berechne die neue Menge an Pheromonen laut der Pheromon-Update-Regel:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \rho\cdot\Delta\tau_{ij},$$

wobei $\Delta\tau_{ij} = 1/L^+$
End for
End for

Print die kürzeste Tour T^+ mit ihrer Länge L^+

Abbildung 3.5: Darstellung des ACS-Algorithmus für die Futtersuche.
(Quelle: In Anlehnung an Bonabeau, Dorigo und Theraulaz, 1999, S. 51)

Der Algorithmus beginnt mit der gleichen Initialisierungsphase wie der AS-Algorithmus. Alle Kanten werden mit der Pheromonmenge τ_0 initialisiert und L^+ wird wieder auf –1 gesetzt. Zusätzlich werden nun auch noch die Kandidatenlisten aller Knoten initialisiert. In der nachfolgenden Hauptschleife wird zunächst für jede Ameise k eine Tour gebildet, wobei bei jedem Übergang zu einem anderen Knoten die lokale Pheromon-Update-Regel angewendet wird. Haben alle Ameisen ihre Tour beendet, wird von jeder Tour die Länge L^k berechnet und gegebenenfalls die beste Tour neu festgelegt. Anschließend werden die Kanten der besten Tour T^+ mit Pheromonen laut der Pheromon-Update-Regel verstärkt, und die Hauptschleife erneut gestartet. Der Algorithmus endet nach einer festgelegten Anzahl t_{max} an Schleifendurchläufen, und gibt dann die beste Tour T^+ und deren Länge L^+ aus. Auch hier

wurden aus den gleichen Gründen, wie zuvor bei der Beschreibung des AS-Algorithmus', auf die Angabe der Iteration bei sämtlichen Variablen verzichtet.
Dorigo und Gambardella haben in ihrem Artikel „Ant Colonies for the Travelling Salesman Problem"[22], der zusammen mit dem Artikel „Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem"[23] den ACS-Algorithmus erstmals vorstellte, ACS mit anderen Optimierungsmethoden verglichen. Dabei wird ersichtlich, dass dieser beispielsweise gegenüber Genetischen Algorithmen oder Neuronalen Netzen konkurrenzfähig ist und teilweise sogar bessere Ergebnisse erzielt. Auch die Auswirkung von Kandidatenlisten wurde getestet, wobei sich herausgestellt hat, dass Kandidatenlisten mit einer geringen Anzahl an Knoten, beispielsweise 20 Knoten, den Algorithmus bei großen Optimierungsproblemen nochmals stark verbessern.

3.3.3 Weitere ACO-Algorithmen

Im Laufe der Zeit wurde immer wieder versucht, AS oder auch den erweiterten Algorithmus ACS zu verbessern. Einige dieser Verbesserungen werden im Folgenden näher beschrieben.

Ant System mit „Elitist Strategy"

Diese Erweiterung des AS Algorithmus' wurde von den Erfindern von AS entwickelt, jedoch wird im Folgenden wieder auf Bonabeau, Dorigo und Theraulaz[24] Bezug genommen, da diese die „elitist strategy" verständlicher und somit auch leichter nachvollziehbar beschrieben haben. Die Idee zu dieser Erweiterung stammt aus dem Bereich der Genetischen Algorithmen, wo „elitist strategies" in einer ähnlichen Weise verwendet werden.

Eine „elitist ant" ist eine Ameise, die die bisher beste Tour T^+ in jeder Iteration zusätzlich mit einem Faktor Q/L^+ verstärkt. In jeder Iteration werden e „elitist ants" zu den m Ameisen hinzugefügt, was bedeutet, dass die beste Tour zusätzlich um $e \cdot Q / L^+$ verstärkt wird. Durch die Verstärkung der besten Tour werden mehr Ameisen dazu gebracht, in der Umgebung der bisher besten Tour nach einer besseren Lösung zu forschen. Der Grundgedanke dahinter ist, dass es bessere Lösungen gibt, die Kanten der bisher besten Tour enthalten. Forschen mehr Ameisen bei der bisher besten Tour, so steigt die Wahrscheinlichkeit, dass diese besseren Lösungen schneller gefunden wird.

[22] Vgl. Dorigo, M. und Gambardella, L. M., *BioSystems* 43 (1997b).
[23] Vgl. Dorigo, M. und Gambardella, L. M., *IEEE Transactions on Evolutionary Computation* (1997a).
[24] Vgl. Bonabeau, E. / Dorigo, M. / Theraulaz, G. (1999) S. 44 f.

Der Algorithmus selbst entspricht dem AS-Algorithmus mit dem einzigen Unterschied, dass die Pheromon-Update-Regel um den Anteil der „elitist ants“ folgender Maßen erweitert wird:

$$\tau_{ij}(t) \leftarrow (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij}(t) + e\cdot\Delta\tau_{ij}^{e}(t), \tag{3.10}$$

wobei $\Delta\tau_{ij}^{e}(t) = Q / L^{+}$, wenn die Kante (i, j) in der bisher besten Tour T^{+} enthalten ist, anderen Falls ist $\Delta\tau_{ij}^{e}(t) = 0$.

In Experimenten hat sich herausgestellt, dass dieser Ansatz eine Verbesserung der Performance bringt, allerdings muss eine geringe Anzahl an „elitist ants“ gewählt werden. Wählt man zu viele „elitist ants“, so kann es dazu führen, dass lediglich ein lokales Optimum, jedoch nicht ein globales, und somit keine optimale Lösung für das gesamte Problem, gefunden wird.

Rank Based Version von Ant System

Ant System basiert darauf, dass gute Lösungen mehr verstärkt werden als schlechte und daher auch mehr erforscht werden als schlechte Lösungen. Dies funktioniert gut, wenn sich die Qualität der Lösungen deutlich unterscheidet. Existieren jedoch mehrere Wege mit ähnlich hoher Qualität, so wird auch auf diesen Wegen mit ähnlich hoher Wahrscheinlichkeit weitergeforscht, was der Grundstrategie, gute Lösungen mehr zu erforschen als schlechtere, widerspricht. Der hier beschriebene Algorithmus AS_{rank}[25] soll diese Situation verhindern.

AS_{rank} ist eine Erweiterung von AS, die auch die „elitist strategy“ einsetzt. Nachdem alle m Ameisen ihre Tour beendet haben, wird eine Reihung aller Ameisen entsprechend ihrer Tourlänge vorgenommen, wobei die Ameise mit der kürzesten Tour den ersten Rang einnimmt. Im Gegensatz zu AS werden zum Aktualisieren der Pheromonspuren nur die ω besten Ameisen herangezogen, und die Pheromonmenge, mit der eine Ameise ihre Kanten verstärkt, wird entsprechend ihrem Rang ψ gewichtet. Durch das Beschränken der Anzahl der Ameisen, die aktualisieren dürfen, wird verhindert, dass zu viele Ameisen nicht optimale Wege verstärken.

Auf Grund des Anwendens der „elitist strategy“ wird die beste Lösung mit e gewichtet. Diese Gewichtung sollte von keiner anderen übertroffen oder auch nur erreicht werden. Daher wird die Gewichtung des Ranges mit e - ψ und die Anzahl der Ameisen, die Pheromonspuren aktualisieren dürfen, mit e - 1 festgelegt. Dies führt dazu, dass die höchste Gewichtung der

[25] Vgl. Bullnheimer, B. / Hartl, R. F. / Strauss, C., *Central European Journal of Operations Research and Economics* 7/1 (1999).

Ränge um eins geringer ist als die Gewichtung der „elitist ants“, und die niedrigste Gewichtung genau 1 beträgt.

Die Pheromon-Update-Regel entspricht jener, die bei dem AS-Algorithmus mit „elitist strategy“ angewendet worden ist[26], jedoch muss bei der Berechnung von $\Delta\tau_{ij}(t)$ auf die ausgewählte Menge an Ameisen, die Pheromonspuren aktualisieren dürfen, und auf die Gewichtung der Pheromonmengen entsprechend dem Rang der Ameise Rücksicht genommen werden. $\Delta\tau_{ij}(t)$ ergibt sich daher aus der Summe der $\Delta\tau_{ij}^{\psi}$-Werte der ersten ω Ameisen. Der $\Delta\tau_{ij}^{\psi}$-Wert seinerseits lässt sich mit der Formel

$$\Delta\tau_{ij}^{\psi} = (e-\psi)\cdot\frac{Q}{L_{\psi}} \tag{3.11}$$

berechnen, sofern die ψ-beste Ameise die Kante (i, j) in ihrer Tour besucht hat, andernfalls ist $\Delta\tau_{ij}^{\psi} = 0$.

An dieser Stelle soll auf einen Vergleich zwischen AS_{rank}, AS mit „elitist strategy“, AS, einem Genetischen Algorithmus und Simulated Annealing hingewiesen werden, in dem sich herausgestellt hat, dass AS_{rank} mit den anderen Algorithmen mithalten kann, und die Ergebnisse von AS hinsichtlich der durchschnittlichen Lösung und der besten Lösung übertroffen hat.

Hybrid Ant System

Der Grundgedanke des Hybrid Ant Systems ist es, einen ACO-Algorithmus mit einer lokalen Suchmethode zu kombinieren. Lokale Suchmethoden haben den Vorteil, dass sie relativ schnell aus einer Menge von guten Lösungen eine sehr gute Lösung finden. Lässt man die lokalen Suchmethoden jedoch in einem Umfeld schlechter Lösungen zu suchen beginnen, so dauert es relativ lange, bis eine Menge von guten Lösungen gefunden worden ist. ACO-Algorithmen ermöglichen es hingegen, von einer Menge von schlechten Lösungen relativ schnell zu guten Lösungen zu kommen. Ihr Manko besteht darin, von diesen guten Lösungen dann zu einer sehr guten Lösung zu kommen. Kombiniert man nun diese beiden Methoden, kann man mittels ACO schnell durchaus gute Anfangslösungen finden, die man dann mittels lokaler Suchmethode zu sehr guten Lösungen verbessern kann. Je nach Problemstellung eignen sich verschiedene Suchmethoden und auch verschiedene Wege, diese Suchmethoden in den ACO-Algorithmus einzubauen.[27]

[26] Siehe Formel 2.10.
[27] Vgl. Gambardella, L. M. und Dorigo, M., *INFORMS Journal of Computing* 12/3 (2000) S. 237.

Die am häufigsten verwendete Möglichkeit, eine lokale Suchmethode in einen ACO-Algorithmus einzubauen, ist, diese Suchmethode auf jede abgeschlossene Tour anzuwenden.[28] Dabei wird die gefundene Tour lokal, durch das Austauschen einiger Kanten, optimiert. Mit den optimierten Touren wird dann, wie in den bisher beschriebenen Algorithmen[29], mit der Berechnung der Tourlänge und dem eventuellen Aktualisieren der bisher besten Tour fortgefahren.

In zahlreichen Artikeln, darunter auch den oben genannten, wird belegt, dass das Erweitern eines Ameisenalgorithmus' mit einer lokalen Suchmethode zu starken Performanceverbesserungen führen kann, sofern die passende Suchmethode gewählt und an der richtigen Stelle eingebaut wurde. Auf lokale Suchmethoden und deren mögliche Auswirkungen wird in Kapitel 4.2.5 näher eingegangen.

MAX-MIN Ant System

Wie schon in ACS, AS mit „elitist strategy" und AS_{rank} beschrieben, kann die Performance des AS-Algorithmus' erhöht werden, wenn man nur noch die beste Ameise oder die ω besten Ameisen die Pheromonmengen aktualisieren lässt bzw. die beste Spur noch zusätzlich verstärkt. Dies kann aber dazu führen, dass ein nicht optimaler Weg zu früh eine so hohe Pheromonspur aufweist, dass kein anderer Weg mehr ausgewählt wird. Im ACS versucht man, dies durch die lokale Pheromon-Update-Regel zu verhindern, bei der „elitist strategy" schützt man sich davor, indem nur wenige „elitist ants" eingesetzt werden, und auch bei AS_{rank} werden e und ω entsprechend festgelegt.

Das MAX-MIN Ant System stellt eine Erweiterung von AS dar, die den Vorteil des Performancegewinns durch das alleinige Aktualisieren der Pheromonspur der besten Lösung nutzt, und den Nachteil der zu frühen Stagnation durch Begrenzung der Pheromonmenge jeder Kante vermeidet. Das bedeutet, es gibt einen Wert τ_{max}, der die maximale Menge an Pheromonen pro Kante repräsentiert und einen Wert τ_{min}, der die Mindestmenge an Pheromonen pro Kante bezeichnet. Die untere Grenze bewirkt, dass jede Kante zumindest zu einer geringen Wahrscheinlichkeit gewählt wird, und die obere Grenze sorgt dafür, dass es nicht zu der Situation kommt, in der Kanten mit einer Pheromonmenge t_{min} gegenüber Kanten mit einer weitaus höheren Pheromonmenge vernachlässigt werden. Die

[28] Vgl. Gambardella, L. M. und Dorigo, M., *INFORMS Journal of Computing* 12/3 (2000) S. 240 f., Dorigo, M. und Gambardella, L. M., *IEEE Transactions on Evolutionary Computation* (1997a) S. 60 f. und Gambardella, L. M. / Taillard, E. / Agazzi, G. (1999) S. 71 f.
[29] Siehe Abbildung 3.4 und Abbildung 3.5.

Begrenzungen bewirken somit, dass weder Kanten, die nie gewählt werden, noch Kanten, die immer gewählt werden, existieren.

Dennoch kann es unter Umständen zur Stagnation kommen. Droht eine solche Stagnation, so wird der „trail-smoothing"-Mechanismus eingesetzt. Dabei wird die Pheromonmenge aller Kanten proportional zu der Differenz zwischen τ_{max} und der momentanen Pheromonstärke $\tau_{ij}(t)$ erhöht.

Der Algorithmus selbst unterscheidet sich von dem AS-Algorithmus zum einen dadurch, dass alle Kanten anfangs mit τ_{max} anstatt mit τ_0 initialisiert werden. Diese Menge wird durch das Verdampfen der Pheromone pro Iteration langsam verringert. Ein weiterer Unterschied liegt darin, dass nur noch jene Ameise die Kanten ihrer Tour verstärken darf, die die bisher kürzeste Tour gefunden hat. Dahingehend verändert sich auch die Pheromon-Update-Regel. Außerdem muss die Regel beinhalten, dass die neue Pheromonmenge innerhalb der Begrenzungen liegt. Die dritte Veränderung liegt schließlich darin, dass, ähnlich wie in HAS, eine lokale Suchmethode eingebaut wird.

Entwickelt wurde dieser Algorithmus von Stützle und Hoos, die in einer Reihe von Artikeln[30] das MAX-MIN Ant System vorstellen. Der Artikel „MAX-MIN Ant System and Local Search for the Travelling Salesman Problem"[31] erscheint hier als beste Grundlage, um einen Einblick in die Funktionsweise von MMAS zu bekommen. Darin wird auch in mehreren Experimenten gezeigt, dass MMAS im Vergleich mit ACS zwar nicht immer die beste Lösung herausfindet, jedoch die Ergebnisse im Durchschnitt besser sind. Weitaus detaillierter wird MMAS in dem Artikel „Improving the Ant-System: A Detailed Report on the MAX-MIN Ant System"[32] präsentiert. Hingegen wird in dem jüngsten Artikel[33] dieser Reihe gezeigt, wie man den ACS-Algorithmus erweitern kann, indem dieser als Basis für MMAS verwendet wird.

Approximate Nondeterministic Tree Search

Dieser Algorithmus basiert auf AS und wird im Folgenden in Anlehnung an Maniezzo[34] beschrieben. Approximate Nondeterministic Tree Search (ANTS) erweitert AS dahingehend, dass eine untere Grenze[35] für die Länge der Tour verwendet wird, wodurch es möglich wird,

[30] Vgl. Stützle, T. und Hoos, H. (1996), (1997a), (1997b) und (1999).
[31] Vgl. Stützle, T. und Hoos, H. (1997b).
[32] Vgl. Stützle, T. und Hoos, H. (1996).
[33] Vgl. Stützle, T. und Hoos, H. (1999).
[34] Vlg. Maniezzo, V., *INFORMS Journal on Computing* 11/4 (1999).
[35] Bei Minimierungsproblemen werden untere Grenzen verwendet, bei Maximierungsproblemen obere Grenzen.

Probleme zu lösen, in denen sich die lokale Information η_{ij} während des Problemlösungsprozesses verändert. Des Weiteren wird eine veränderte Übergangsregel und eine erweiterte Pheromon-Update-Regel angewendet.

Zu Beginn des Algorithmus' wird die untere Grenze der Tourlänge berechnet. Dafür kann „Gilmore-Lawler Lower Bound" (GLB) verwendet werden, dessen Nachteil aber ein hoher Berechnungsaufwand[36] von $O(n^3)$ ist. Maniezzo setzt für seine Problemstellung den etwas schwächeren Berechnungsalgorithmus LBD ein, der dafür nur einen Aufwand von $O(n^2)$ benötigt.[37]

Ist die untere Grenze berechnet, so suchen alle Ameisen unabhängig voneinander ihre Touren. Um von einem Knoten i zu einem Knoten j zu kommen, wird jedoch die folgende, abgeänderte Übergangsregel verwendet:

$$p_{ij}^k(t) = \frac{\alpha \cdot \tau_{ij}(t) + (1-\alpha) \cdot \eta_{ij}}{\sum_{l \in J_j^k} \left[\alpha \cdot \tau_{lj}(t) + (1-\alpha) \cdot \eta_{lj}\right]}, \tag{3.12}$$

falls der Knoten j in der Liste J_i^k enthalten ist, andernfalls ist $p_{ij}^k(t) = 0$. Der Vorteil dieser Übergangsregel im Vergleich zur Übergangsregel bei AS[38] ist, dass der Parameter β entfällt.

Hat eine Ameise eine Lösung gefunden, so wird diese mittels lokalem Suchalgorithmus optimiert. Haben alle Ameisen die Lösungssuche und das nachfolgende Optimieren abgeschlossen, so werden die Pheromonspuren aktualisiert. Die dafür angewendete Pheromon-Update-Regel gleicht jener in AS[39], jedoch wird das Verdampfen der Pheromonspuren durch das Vergleichen jeder Tour mit dem Durchschnitt der c vorhergehenden Touren ersetzt. Die Pheromonmenge jeder Kante (i, j) wird daher folgender Maßen aktualisiert:

$$\tau_{ij}(t) \leftarrow \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{3.13}$$

[36] Der Berechnungsaufwand soll hier in der Big-O-Notation dargestellt werden. Dabei steht n für die Problemgröße, die zum Beispiel durch die Anzahl an Knoten festgelegt werden kann. Möchte man beispielsweise den Aufwand eines Algorithmus' berechnen, der die Entfernung von n Knoten zueinander berechnet, wobei die Entfernung von Knoten A zu Knoten B ungleich der Entfernung von Knoten B zu Knoten A ist, so müssen n*(n-1) Berechnungen durchgeführt werden. Da die Notation lediglich das Ziel hat, Algorithmen grob vergleichbar zu machen, und der Berechnungsaufwand meist nur für große Probleme ermittelt wird, kann man das „-1" vernachlässigen und erhält somit einen Aufwand von n^2, was in Kurzschreibweise als $O(n^2)$ bezeichnet wird.

[37] Das Vorgehen bei der Berechnung der unteren Grenze mittels GLB und LBD soll hier nicht detailliert beschrieben werden, kann jedoch in Maniezzo, V., *INFORMS Journal on Computing* 11/4 (1999) nachgelesen werden.

[38] Siehe Formel 2.1.

[39] Siehe Formel 2.3.

Der Wert $\Delta\tau_{ij}^{k}(t)$ ergibt sich aus der Formel:

$$\Delta\tau_{ij}^{k}(t) = \tau_0 \cdot \left(1 - \frac{L^k(t) - LB}{\overline{L} - LB}\right), \tag{3.14}$$

wobei LB die untere Grenze darstellt, $L^k(t)$ die Länge der Tour repräsentiert, die die Ameise k gefunden hat, und $\overline{L}$ der durchschnittlichen Länge der letzten c Touren entspricht. Der Vorteil dieser Änderung ist, dass der Verdampfungsparameter ρ wegfällt, der eine kritische Rolle in AS spielt. Wählt man ihn zu groß, so kann dies schnell zur Stagnation führen, wählt man ihn hingegen zu klein, so wird zu wenig Information von einer Iteration zur nächsten übertragen und der Lernprozess beginnt in jeder Iteration von neuem. Statt ρ existiert nun zwar der Parameter c, dieser ist aber weitaus weniger kritisch.

Maniezzo wendet den ANTS-Algorithmus auf das Quadratic Assignment Problem an, und vergleicht die Ergebnisse mit zwei heuristischen Methoden, die zum damaligen Zeitpunkt die besten Lösungen für dieses Problem erzielten. In diesem Vergleich ist ANTS sowohl hinsichtlich der besten Lösung als auch hinsichtlich der durchschnittlich besten Lösung den beiden Methoden überlegen.

Fast Ant System

Fast Ant System (FANT) unterscheidet sich von AS hauptsächlich durch die Anzahl der Ameisen und die Organisation der Pheromonmengen auf den Kanten. Außerdem enthält der Algorithmus eine lokale Suchmethode, die die einzelnen Touren nach ihrer Konstruktion lokal verbessert.

Was die Anzahl der Ameisen betrifft, so wird in FANT nur eine einzelne Ameise verwendet. Das bedeutet, dass es keine Population gibt. Auch das Verdampfen der Pheromonspur existiert in FANT nicht. Initialisiert werden alle Kanten mit dem Wert eins und nach jeder Iteration wird für jede Kante die folgende Pheromon-Update-Regel angewendet:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + r \cdot \Delta\tau_{ij} + r^* \cdot \Delta\tau_{ij}^{+}, \tag{3.15}$$

wobei für jede Kante (i, j), die in der momentanen Tour t enthalten ist, $\Delta\tau_{ij}(t) = 1 / L(t)$ gilt. Ist die jeweilige Kante nicht enthalten, so gilt $\Delta\tau_{ij}(t) = 0$. Für den Wert $\Delta\tau_{ij}^{+}$ gilt entweder $\Delta\tau_{ij}^{+} = 1/L^{+}(t)$, wenn die jeweilige Kante in der bisher besten Tour T^{+} vorkommt, oder andernfalls $\Delta\tau_{ij}^{+} = 0$. Der Wert des Parameters r kann sich während des Algorithmus' verändern, der Wert des Parameters r^* ist fix. Des Weiteren gibt es zwei Situationen, in denen anders vorgegangen wird. Ist die momentane Lösung die bisher beste, so wird r und

die Pheromonmengen aller Kanten gleich eins gesetzt. Das Löschen der bisherigen Pheromonspuren intensiviert die beste Lösung, da in der nächsten Iteration nur noch die Kanten dieser besten Lösung und die Kanten der dann gefundenen Lösung mit Pheromonen bestreut werden. Die zweite Situation, die ein anderes Vorgehen verlangt, tritt ein, wenn die momentane Lösung gleich der bisher besten Lösung ist. Dann wird r um eins erhöht und die Pheromonmengen aller Kanten auf r gesetzt. Durch die Erhöhung von r wird in der nächsten Iteration die momentane Tour höher gewichtet. Der Unterschied zwischen Kanten der momentanen Tour und Kanten der bisher besten Tour verringert sich, was zu einer Umverteilung der Pheromonmengen führt.

Diese Erweiterung wurde von Taillard und Gambardella das erste Mal in ihrem Artikel „Adaptive Memories for the Quadratic Assignment Problem"[40] präsentiert. Darin wird nicht nur der Algorithmus erklärt, sondern auch dessen Leistungsfähigkeit mit dem zuvor beschriebenen HAS-Algorithmus und anderen genetischen Methoden verglichen. Die Ergebnisse zeigen, dass bei kurzer Berechnungsdauer FANT die besten Ergebnisse liefert, da die Initialisierung wenig Zeit in Anspruch nimmt und daher schon früher mit dem Lernprozess begonnen werden kann.

3.4 Einsatzmöglichkeiten von ACO-Algorithmen

Bis jetzt wurde versucht, das Verhalten der Ameisen bei der Futtersuche nachzuahmen, und damit ausschließlich das Problem, wie man von einem Ausgangspunkt am schnellsten zu einem Zielpunkt kommt, auf verschiedene Weise gelöst. Jedoch können auch andere Problem mittels ACO-Algorithmen bewältigt werden.

3.4.1 Grundlegendes

Laut Dorigo und Di Caro[41], auf die im Folgenden Bezug genommen wird, werden ACO-Algorithmen für diskrete Optimierungsprobleme eingesetzt, die durch bestimmte Eigenschaften charakterisiert sind. Die Darstellung solcher Probleme erfolgt meist als Graph. Im Folgenden sollen diese charakteristischen Eigenschaften theoretisch erläutert werden. Diese Erläuterungen dienen als Grundlage für die im nächsten Kapitel beschriebenen Problemstellungen und werden dort an Hand von Beispielen verdeutlicht.

[40] Vgl. Taillard, E. und Gambardella, L. M. (1997).
[41] Vgl. Dorigo, M. und Di Caro, G. (1999) S. 14 ff.

Abbildung 3.6 zeigt einen Graphen, der alle charakteristischen Eigenschaften eines diskreten Optimierungsproblems darstellt. Dazu gehört eine endliche Menge von Knoten, die als Kreise dargestellt werden, und eine endliche Menge von gerichteten, gewichteten Kanten, die den Pfeilen zwischen den Knoten entsprechen. Die Gewichtung einer Kante entsteht durch eine Kostenfunktion[42], die möglicherweise auch von der momentanen Iteration abhängig sein kann. Eventuell können auch mehrere Gewichtungen auf einer Kante existieren. In Abbildung 3.6 werden die Kanten hinsichtlich der Distanz und der Zeitdifferenz zwischen zwei Knoten gewichtet. Da die Kanten gerichtet sind, können sich die Kosten von Knoten A zu Knoten B gegenüber den Kosten von Knoten B zu Knoten A unterscheiden. Die Richtung wird dabei durch die Pfeilspitze symbolisiert. Werden zwei Knoten durch eine Linie ohne Pfeilspitzen verbunden, wie in Abbildung 3.3 bei der Darstellung des Problems der Futtersuche, so sind die Kosten von Knoten A zu Knoten B gleich den Kosten von Knoten B zu Knoten A. Des Weiteren können Beschränkungen beispielsweise bezüglich der Start- und Zielknoten oder der Existenz von Kanten definiert werden. Die Darstellung des Start- und Zielknotens erfolgt durch eine explizite Beschriftung des Knotens, wobei auch mehrere Start- und/oder Zielknoten existieren können. Kann jeder Knoten des Graphen als Startknoten bzw. Zielknoten gesehen werden, so erfolgt keine explizite Beschriftung. Eine weitere charakteristische Eigenschaft von Optimierungsproblemen ist, dass das Problem mehrere Zustände aufweisen kann. Diese werden als Sequenzen von Knoten ausgedrückt, wobei die Menge aller gültigen Sequenzen alle Beschränkungen erfüllen muss. Die Lösung eines diskreten Optimierungsproblems ist eine gültige Sequenz, die alle Anforderungen des Problems erfüllt. Die Gesamtkosten der Lösung ergeben sich aus einer bestimmten Funktion, die von den Kosten aller Kanten, die in der Lösung enthalten sind, abhängig ist. Des Weiteren existiert eine geregelte Nachbarschaftsstruktur, die besagt, welche Sequenzen miteinander benachbart sind. Darauf soll an dieser Stelle jedoch nicht näher eingegangen werden.

[42] Der Begriff Kosten hat hier nicht zwingend die Bedeutung von Geldeinheiten, sondern kann auch Entfernungen, Zeit oder andere Optimierungskriterien darstellen.

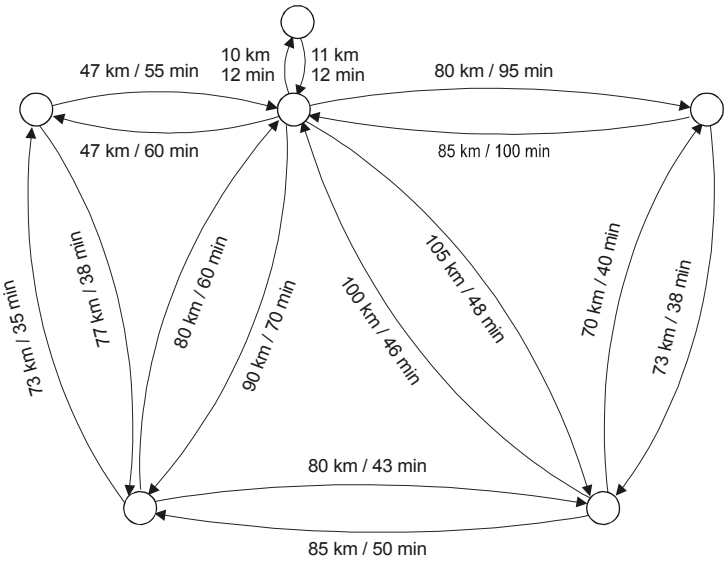

Abbildung 3.6: Darstellung eines zweifach gewichteten, gerichteten Graphen

ACO-Algorithmen eignen sich besonders gut für das Lösen von sogenannten NP-harten Problemen, da sie die hohe Komplexität dieser Probleme in einer annehmbaren Zeit bewältigen können. Dabei kann das Problem nach einem Ziel oder auch nach mehreren Zielen optimiert werden. Auch für dynamische Probleme, bei denen sich die Kosten der verschiedenen Kanten während des Problemlösungsprozesses verändern, lassen sich ACO-Algorithmen mit Erfolg einsetzen. Durch die Unabhängigkeit der einzelnen Ameisen untereinander, lassen sich auch sogenannte verteilte Probleme lösen, deren Informationen über mehrere Rechner verteilt sind.

3.4.2 NP-harte Probleme

Unter NP-harten Problemen versteht man Probleme, deren Komplexität mit der Anzahl an Knoten exponentiell zunimmt. Daher wird meist nicht nach der besten Lösung, sondern nach einer möglichst guten Annäherung gesucht. Dies führt zu einer Suche nach immer besseren Algorithmen, die eine immer genauere Annäherung an die beste Lösung erlangen. Mittels ACO-Algorithmen lassen sich zum jetzigen Zeitpunkt sehr gute Ergebnisse für diverse NP-harte Probleme erzielen. Einige dieser Probleme sollen im Folgenden genauer beschrieben werden.

Travelling Salesman Problem

Das bekannteste Beispiel für ein NP-hartes Problem ist das Travelling Salesman Problem (TSP), in dem ein Handelsreisender die Aufgabe hat, jede Stadt innerhalb eines bestimmten Gebietes genau einmal zu besuchen, und am Schluss seine Tour wieder in der Stadt zu

beenden, von der er gestartet ist. Alle Städte sind miteinander durch Straßen verbunden. Ziel ist es, die Gesamtlänge der Tour des Handlungsreisenden zu minimieren, wobei sich diese aus der Summe der Längen der benutzten Straßen ergibt.

Formal können die Städte als Knoten und die dazwischenliegenden Straßen als Kanten gesehen werden. Gewichtet ist eine Kante durch die Länge der Straße, wobei man beim TSP davon ausgeht, dass der Weg von Knoten A zu Knoten B gleich dem umgekehrten Weg ist. Beim Asymmetric Travelling Salesman Problem (ATSP) wird nicht von diese Annahmen ausgegangen, was bedeutet, dass die Kanten gerichtet sind. Die Bedingungen, dass jede Stadt genau einmal besucht werden muss und dass die Tour in jener Stadt enden muss, in der sie angefangen hat, können als Beschränkungen gesehen werden. Eine Sequenz ist eine beliebige Folge von Knoten. Entspricht der erste Knoten dem letzten, und kommen alle anderen Knoten genau einmal vor, so handelt es sich um eine gültige Sequenz. Gesucht wird nun jene gültige Sequenz mit der kürzesten Gesamtlänge.

Im Jahr 1991 entwickelten Colorni, Dorigo und Maniezzo[43] das Ant System, den ersten ACO-Algorithmus, und wandten ihn auf das TSP an. Stützle und Hoos[44] beschrieben 1996, wie man das MMAS auf das TSP anwendet, und ein Jahr darauf zeigten Dorigo und Gambardella[45], wie das TSP mittels ACS gelöst werden kann.

Quadratic Assignment Problem

Ein weiteres NP-hartes Problem ist das Quadratic Assignment Problem (QAP), in dem es eine Menge von n Einrichtungen, beispielsweise Fabriken, und eine Menge von n Orten gibt, wobei jeweils eine Einrichtung in einem Ort errichtet werden soll. Zwischen den Fabriken müssen beispielsweise Materialien ausgetauscht werden. Die Anzahl dieser und die Distanz bzw. die Zeitdifferenz zwischen den Orten ist bekannt. Ziel ist es, eine Zuordnung aller Fabriken zu jeweils einem Ort zu finden, in der die zurückgelegte Entfernung bzw. aufgewendete Zeit pro Material minimal ist.

Das TSP lässt sich auch als spezielles QAP definieren, wenn man die n Städte als Orte sieht und jeder Stadt eine Nummer von 1 bis n, entsprechend der Reihenfolge in der sie besucht wird, zuordnet. Ziel ist es dann, jeweils einen Ort einer Nummer zuzuordnen, sodass die Tour, bestehend aus den Orten in der Reihenfolge der zugeordneten Nummern, minimal ist.

[43] Vlg. Colorni, A. / Dorigo, M. / Maniezzo, V. (1991).
[44] Vgl. Stützle, T. und Hoos, H. (1996).
[45] Vgl. Dorigo, M. und Gambardella, L. M., *IEEE Transactions on Evolutionary Computation* 1 (1997a) und Dorigo, M. und Gambardella, L. M., *BioSystems 43* (1997b).

Abbildung 3.7 zeigt, wie das QAP als Graph dargestellt werden kann. Dabei werden sowohl Orte als auch Fabriken als Knoten eingezeichnet. Jeder Ort wird mit jeder Fabrik und umgekehrt jede Fabrik mit jedem Ort durch eine Kante verbunden. Die Kosten der Zuordnung einer Fabrik zu einem Ort ergeben sich aus der Summe der zurückgelegten Entfernungen bzw. aufgewendeten Zeit pro Material, das zwischen der ausgewählten Fabrik und anderen Fabriken ausgetauscht werden muss. Diese können aber erst dann vollständig berechnet werden, wenn alle Fabriken, zu denen die Materialien ausgetauscht werden sollen, schon einem Ort zugeordnet worden sind. Ist dies nicht der Fall, so können lediglich Teilkosten errechnet und auf Grund dieser Teilkosten Entscheidungen getroffen werden. Die wichtigsten Beschränkungen für dieses Problem sind, dass die Anzahl an Fabriken gleich der Anzahl an Orten sein muss, dass diese jeweils alternierend gewählt werden müssen und dass kein Ort bzw. keine Fabrik zwei mal zugeordnet werden darf. Die Pfeile in Abbildung 3.7 zeigen eine gültige Sequenz, wobei die erste Zuordnung durch den Pfeil mit einem Stern gekennzeichnet ist. Verfolgt man diesen Pfeil, so erhält man eine Zuordnung von Fabrik 1 zu Ort B. Danach wird Fabrik 4 dem Ort D zugeordnet und die letzten beiden Zuordnungen sind Fabrik 2 zu Ort A und Fabrik 3 zu Ort C. Gesucht ist nun jene gültige Sequenz, die die geringsten Gesamtkosten verursacht.

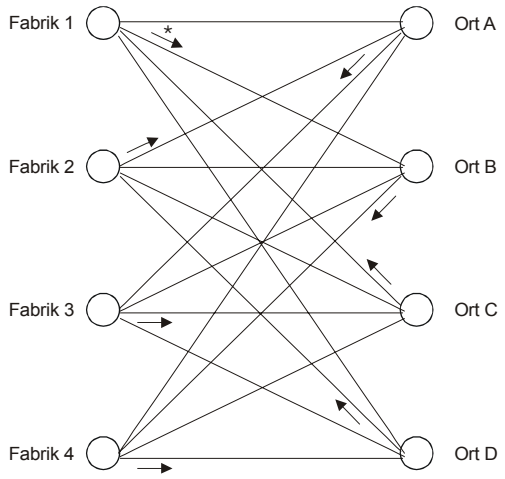

Abbildung 3.7: Darstellung eines QAP als Graph: Dieser Graph stellt ein Quadratic Assignment Problem mit vier Fabriken und vier Orten dar. Die Pfeile kennzeichnen eine mögliche Lösung, wobei die erste Zuordnung mit einem Stern gekennzeichnet ist.

Das QAP ist ein Problem, das im Laufe der Jahre mit Hilfe sehr vieler, verschiedener Algorithmen gelöst worden ist. Einige dieser Algorithmen wurden im vorigen Kapitel ausführlich beschrieben. Beispielsweise schrieb Maniezzo 1994 einen technischen Report[46],

[46] Vgl. Maniezzo, V. et al. (1994).

in dem er zeigte, wie das QAP mittels AS gelöst werden kann. Drei Jahre später wandten Gambardella, Taillard und Dorigo[47] den HAS-Algorithmus auf das QAP an, jedoch wurde der Algorithmus nicht nur auf das Problem adaptiert, sondern die Veränderungen gingen so weit, dass man streng genommen nicht mehr von einem ACO-Algorithmus sprechen kann. Dies beruht darauf, dass die Pheromonspuren nicht dazu verwendet werden, neue Lösungen zu finden, sondern schon vorhandene Lösungen zu verändern. Im gleichen Jahr wurde das QAP mit dem FANT-Algorithmus[48] und ein Jahr später mit dem ANTS-Algorithmus[49] gelöst. Im Jahr 1999 zeigten Stützle und Hoos in ihrem Artikel[50], wie man den MMAS-Algorithmus auf das QAP anwendet.

Vehicle Routing Problem

Das letzte Standardoptimierungsproblem, dass in dieser Arbeit näher beschrieben werden soll, ist das Vehicle Routing Problem (VRP). Das Problem besteht darin, eine bestimmte Anzahl an Kunden mit den jeweils nachgefragten Gütern zu beliefern und nachgefragte Services, die eine bestimmte Zeit in Anspruch nehmen, durchzuführen. Die Touren eines Fahrzeuges starten und enden beim Lager, und sind durch eine bestimmte Kapazität des Fahrzeuges und eine bestimmte Fahrdauer bzw. Tourlänge beschränkt. Wird ein Kunde besucht, so muss dessen Nachfrage mit einem Besuch gedeckt werden. Die Fahrzeit bzw. Distanz zwischen den einzelnen Orten ist bekannt. Das Ziel ist es, sämtliche Nachfragen zu erfüllen, und die Touren hinsichtlich Zeit oder Länge zu minimieren. Auch dieses Problem kann als TSP verstanden werden, sobald die Beschränkungen hinsichtlich der Kapazität und der Fahrdauer bzw. Tourlänge aufgehoben worden sind. Die Darstellung als Graph erfolgt analog zu den bisher beschriebenen Problemen.

Bullnheimer, Hartl und Strauss[51] zeigten 1997 erstmals, dass dieses Problem auch mittels ACO-Algorithmus gelöst werden kann, wobei sie AS, HAS und mehrere Abwandlungen von HAS anwandten. 1999 beschrieben Gambardella, Taillard und Agazzi[52], auf deren Artikel im Folgenden noch näher eingegangen wird, wie sie das VRP durch ein Ant Colony System mit mehreren Kolonien lösten.

[47] Vgl. Gambardella, L. M. / Taillard, E. D. / Dorigo, M., *Journal of the Operational Research Society* 50/2 (1999).
[48] Vgl. Taillard, E. und Gambardella, L. M. (1997).
[49] Vgl. Maniezzo, V., *INFORMS Journal on Computing* 11/4 (1999).
[50] Vgl. Stützle, T. und Hoos, H. (1999).
[51] Vgl. Bullnheimer, B. / Hartl, R. F. / Strauss C. (1997).
[52] Vgl. Gambardella, L. M. / Taillard, E. / Agazzi, G. (1999).

3.4.3 Mehrzieloptimierungsprobleme

Bis jetzt wurde von Problemstellungen mit einem Optimierungskriterium ausgegangen, es existieren jedoch auch Optimierungsprobleme, in denen es zwei oder mehr Kriterien gibt, nach denen optimiert werden soll. Sind diese Kriterien nicht gleichwertig, was bedeutet, dass ein Kriterium einem anderen bevorzugt wird, so wird für jedes Kriterium eine Kolonie eingesetzt. Gambardella, Taillard und Agazzi[53] beschreiben, wie man ein VRP dahingehend löst, dass die Anzahl der Touren und die gesamte Fahrzeit aller Touren minimal ist, wobei das erste Kriterium dem zweiten bevorzugt wird. Die erste Kolonie, der das Minimieren der Anzahl der Touren zugeordnet ist, versucht dabei, eine Lösung mit einer Tour weniger als bei der momentan globalen Lösung zu finden, während die zweite Kolonie die Aufgabe hat, die momentan globale Lösung nach dem zweiten Kriterium zu optimieren. Findet die erste Kolonie eine neue, globale Lösung, so starten beide Kolonien erneut mit der gefundenen Lösung. Das Austauschen von Informationen zwischen den beiden Kolonien basiert dabei ausschließlich auf dem Aktualisieren der Pheromonmengen. Auch Doerner, Hartl und Reimann[54] beschreiben ein ähnliches Problem, in dem jedoch die zweite Kolonie nicht mit der Lösung der ersten Kolonie arbeitet, sondern unabhängig von dieser ihr Kriterium optimiert.

Anders muss vorgegangen werden, wenn alle Kriterien gleichwertig sind. Iredi, Merkle und Middendorf[55] beschreiben zu diesem Problemkreis zwei Methoden, die Single Colony-Methode[56] und die Multi Colony-Methode, die sich zum Bewältigen solcher Probleme eignen. Da Problemstellungen mit gleichwertigen Kriterien für diese Arbeit von größerem Interesse sind, wird im Folgenden näher auf die beiden Methoden eingegangen.

Single Colony-Methode

Bei der Single Colony-Methode wird das Problem durch eine einzelne, jedoch heterogene Ameisenkolonie gelöst. Im Unterschied zu homogenen Kolonien, in denen alle Ameisen die gleiche Prioritätenverteilung hinsichtlich der verschiedenen Optimierungskriterien haben, besteht eine heterogene Kolonie aus Ameisen mit jeweils unterschiedlicher Prioritätenverteilung hinsichtlich dieser Kriterien. Da Iredi, Merkle und Middendorf in ihrem Artikel von zwei Optimierungskriterien ausgehen, kann für jede Ameise k die relative

[53] Vgl. Gambardella, L. M. / Taillard, E. / Agazzi, G. (1999).
[54] Vgl. Doerner, K. / Hartl, R. F. / Reimann, M. (2001).
[55] Vgl. Iredi, S. / Merkle, D. / Middendorf, M. (2001).
[56] Der Name dieser Methode ist nicht allgemein gültig, er wurde zwecks Schlüssigkeit im Rahmen dieser Arbeit so benannt.

Wichtigkeit des ersten Kriteriums λ durch die Formel

$$\lambda = \frac{k-1}{m-1} \tag{3.16}$$

berechnet werden. Auf Grund der Tatsache, dass eine Ameise zwei verschiedene Ziele mit unterschiedlicher Priorität verfolgt, muss die jeweilige Übergangsregel des gewählten Algorithmus' entsprechend angepasst werden. Da Iredi, Merkle und Middendorf den AS-Algorithmus gewählt haben, sieht die Übergangsregel folgender Maßen aus[57]:

$$p_{ij}^{k} = \frac{\left[\tau_{ij}(t)\right]^{\lambda\cdot\alpha} \cdot \left[\tau'_{ij}(t)\right]^{(1-\lambda)\cdot\alpha} \cdot \left[\eta_{ij}\right]^{\lambda\cdot\beta} \cdot \left[\eta'_{ij}\right]^{(1-\lambda)\cdot\beta}}{\sum_{l\in J_i^k}\left(\left[\tau_{il}(t)\right]^{\lambda\cdot\alpha} \cdot \left[\tau'_{il}(t)\right]^{(1-\lambda)\cdot\alpha} \cdot \left[\eta_{il}\right]^{\lambda\cdot\beta} \cdot \left[\eta'_{il}\right]^{(1-\lambda)\cdot\beta}\right)} \tag{3.17}$$

Da die Erkenntnis, wie gut ein bestimmter Weg bezüglich des ersten Ziels ist, mit der Erkenntnis bezüglich des zweiten Zieles nicht vermischt werden soll, müssen die Pheromonmengen hinsichtlich der Ziele getrennt gespeichert werden. $\tau_{ij}(t)$ soll daher im Folgenden die Pheromonmenge bezüglich des ersten Ziels und $\tau'_{ij}(t)$ die Pheromonmenge bezüglich des zweiten Ziels der Kante (i, j) zum Zeitpunkt t darstellen. Auch bei der lokalen Information η wird analog hinsichtlich der Ziele unterschieden.

Haben alle Ameisen eine Lösung gefunden, muss entschieden werden, welche Ameisen die Pheromonspuren aktualisieren dürfen. Besteht eine Kolonie aus relativ vielen Ameisen, so sind alle Ameisen, die eine in der jeweiligen Iteration nicht-dominierte Lösung[58] gefunden haben, zum Aktualisieren der Pheromonmengen berechtigt. Existieren nur relativ wenige Ameisen in einer Kolonie, so sind die λ-Werte stark unterschiedlich. Dies hat zur Folge, dass alle Ameisen in unterschiedlichen Richtungen, abhängig von der Prioritätenverteilung der Kriterien, suchen, und meist jede Ameise eine zwar in dieser Iteration nicht-dominierte Lösung findet, diese aber nicht zwingend auch eine gute Lösung sein muss. Berechtigt man nun alle Ameisen, die eine nicht-dominierte Lösung gefunden haben, zum Aktualisieren der Pheromonspuren, so werden in diesem Fall auch schlechte Lösungen verstärkt. Daher wäre es eine sinnvolle Möglichkeit, nur jene Ameisen aktualisieren zu lassen, die eine insgesamt, also auch von Lösungen aus vorherigen Iterationen, nicht-dominierte Lösung gefunden haben.

[57] Vgl. Formel 2.1.
[58] Eine Lösung ist dann dominiert, wenn zumindest eine andere Lösung bei allen wesentlichen Kriterien mindestens gleich gute und bei mindestens einem sogar bessere Ergebnisse erzielt. Nicht dominierte Lösungen erfüllen folglich diese Bedingungen nicht.

Wird eine Ameise zum Aktualisieren der Pheromonmengen ausgewählt, so muss sie die Pheromonmengen beider Kriterien aktualisieren. Da der AS-Algorithmus gewählt wurde, entspricht die Pheromon-Update-Regel jener in AS[59], jedoch wird $\Delta\tau_{ij}^{k}(t)$ und $\Delta\tau'^{k}_{ij}(t)$ der Wert $1/\omega$ zugewiesen, wenn die Ameise k die Kante (i, j) benutzt und für das Aktualisieren der Pheromonmengen ausgewählt wurde, andernfalls entsprechen beide Wert Null. Durch das einheitliche Erhöhen der Pheromonmengen hat jede Ameise der gleichen Iteration den gleichen Einfluss auf die Pheromonspuren.

Multi Colony-Methode

In gewisser Weise ist die Multi Colony-Methode eine Erweiterung der Single Colony-Methode. Der größte Unterschied besteht darin, dass es in ersterer nicht nur eine Ameisenkolonie gibt, sondern mehrere. Diese Kolonien sind heterogen, wobei jede Kolonie aus gleich vielen Ameisen besteht. Die Pheromonmengen werden bezüglich der Ziele jeweils in einer Pheromonmatrix gespeichert. Da auch diese Methode an Hand des Artikels von Iredi, Merkle und Middendorf beschrieben werden soll und dort von einem Optimierungsproblem mit zwei Zielen ausgegangen wird, verwaltet jede Kolonie zwei Pheromonmatrizen.

Würde man beim Aktualisieren der Pheromonmengen analog zur Single Colony-Methode vorgehen, sodass jede Ameise, die in ihrer Kolonie eine nicht-dominierte Lösung findet, berechtigt ist, die Pheromonmengen zu aktualisieren, dann würde das die gleichen Ergebnisse liefern, als würde man die Single Colony-Methode mehrmals gleichzeitig ausführen. Dabei kommt es zu keiner Kooperation zwischen den Kolonien, jedoch gerade diese Kooperation ist für die Multi Colony-Methode entscheidend. Um eine Zusammenarbeit der Kolonien zu erreichen, muss ein globaler „solution pool“ geschaffen werden, der alle lokalen Lösungen enthält. Zum Aktualisieren der Pheromonmengen werden nur jene Ameisen berechtigt, die eine globale, nicht-dominierte Lösung gefunden haben. Es gibt zwei Möglichkeiten, wo diese Ameisen ihre Tour mittels Pheromonen verstärken sollen. Diese werden in Abbildung 3.8 bildlich dargestellt. Die erste Möglichkeit ist, dass jede Ameise nur in ihrer eigenen Kolonie Pheromonmengen aktualisieren darf. Das hat den Vorteil, dass die Qualität der Lösung global bewertet wird und nicht lokal gute, aber global schlechte Lösungen verstärkt werden. Die zweite Möglichkeit besteht darin, die Lösungen hinsichtlich des Optimierungserfolges eines bestimmten Kriteriums, beispielsweise des ersten, zu sortieren, und in h gleich große Teile aufzuteilen, wobei der i-te Teil der i-ten Kolonie zugeordnet wird und h der Anzahl der Kolonien entspricht. Die globale, nicht-dominierte

[59] Vgl. Formel 2.3.

Lösung einer Ameise kann daher einem bestimmten Teil zugeordnet werden, und dieser Teil ist wiederum einer Kolonie zugeordnet, in welcher die Ameise dann die Pheromonmatrizen aktualisieren soll. Durch diese Methode wird jede Kolonie auf bestimmte Lösungsregionen spezialisiert, beispielsweise werden in der ersten Kolonie immer nur Lösungen verstärkt, die das erste Kriterium stark minimiert haben, jedoch einen schlechten Wert für das zweite Kriterium liefern. Umgekehrt verhält es sich bei der letzten Kolonie.

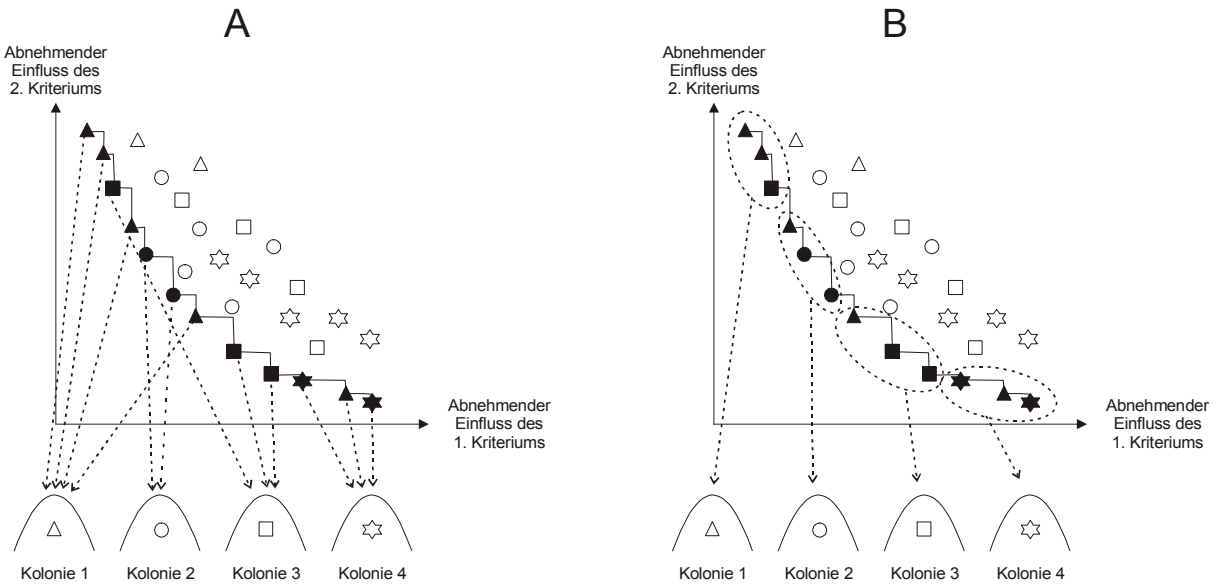

Abbildung 3.8: Mögliche Aktualisierungsstrategien der Ameisen mit nicht-dominierten Lösungen in der Multi Colony-Methode:
A) Jede Ameise, die eine nicht-dominierte Lösung gefunden hat, darf in ihrer eigenen Kolonie die Pheromonmengen verstärken.
B) Die Lösungen werden in gleiche Teile aufgeteilt und jeder Teil einer Kolonie zugeordnet. Ameisen dürfen in jener Kolonie die Pheromonmengen aktualisieren, der dem Teil ihre Lösung zugeordnet ist.
(Quelle: In Anlehnung an Iredi, Merkle und Middendorf, 2001, S. 366)

Da es sich bei den Kolonien um heterogene Kolonien handelt, muss, ähnlich wie bei der Single Colony-Methode, der λ-Wert jeder Ameise errechnet werden. Hierfür gibt es in der Multi Colony-Methode drei verschiedene Methoden. Nach der ersten Methode ergibt sich der λ-Wert für eine Kolonie entsprechend der Berechnung in der Single Colony-Methode, wobei m durch m/h ersetzt werden muss, da m die Gesamtzahl der Ameisen darstellt und m/h die Anzahl der Ameisen in der Kolonie bezeichnet. Diese Berechnung erfolgt in allen Kolonien gleich, was zur Folge hat, dass alle Kolonien hinsichtlich der Prioritätenverteilung der darin existierenden Ameisen gleich sind. In der zweiten Methode werden alle Ameisen geordnet nach Kolonien in einer virtuellen Kolonie zusammengefasst. In dieser virtuellen Kolonie wird dann die Berechnung der λ-Werte analog zur Berechnung in der Single Colony-Methode durchgeführt, was zur Folge hat, dass jeder λ-Wert nur einmal vorkommt und sich die

Kolonien auf bestimmte Lösungsregionen spezialisieren. Die dritte Methode ist eine Mischform der beiden ersten. Hier überlappen die λ-Intervalle die vorgereihte Kolonie und die nachgereihte Kolonie jeweils zu 50 %. Wie diese Methode für vier Kolonien mit jeweils sieben Ameisen aussieht, zeigt Abbildung 3.9.

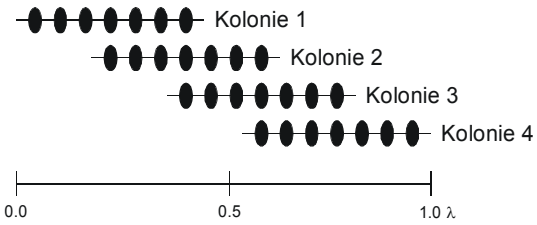

Abbildung 3.9: Überlappende Zuordnung von λ-Werten an Ameisen in der Multi Colony-Methode: Zeigt wie die λ-Werte nach der dritten Methode auf vier Kolonien mit jeweils sieben Ameisen aufgeteilt werden.
(Quelle: In Anlehnung an Iredi, Merkle und Middendorf, 2001, S. 366.)

Der Vollständigkeit halber sei an dieser Stelle erwähnt, dass auch Optimierungsprobleme, die nach nur einem Ziel optimiert werden sollen, mit mehreren Kolonien gelöst werden können. Informationen dazu, im Speziellen zur Kooperation und Kommunikation zwischen den einzelnen Kolonien, kann in dem Artikel „Information Exchange in Multi Colony Ant Algorithms“[60] nachgelesen werden.

3.4.4 Dynamische und verteilte Probleme

Unter dynamischen Problemen versteht man Probleme, deren Struktur sich während des Lösungsprozesses verändern. Beispiele für solche Veränderungen sind das Wegfallen eines Knotens oder einer Kante, das Hinzukommen eines Knotens oder einer Kante und Änderungen der lokalen Informationen. Verteilte Probleme liegen dann vor, wenn das Problem innerhalb eines verteilten Systems[61] auftritt und nur innerhalb dieses Systems gelöst werden kann. Der Prozess der Lösungsfindung ist dadurch auf mehrere Rechner aufgeteilt.

Im Folgenden werden Strategien vorgestellt, die das Lösen von dynamischen, nicht verteilten Problemen ermöglichen. Konkret wird dabei auf das dynamische TSP eingegangen, in welchem nach einer bestimmten Anzahl an Iterationen ein Knoten entweder entfernt oder

[60] Vgl. Middendorf, M. / Reischle, F. / Schmeck, H. (2000).
[61] Unter einem verteilten System versteht man mehrere voneinander unabhängige Computer, die durch ein Netzwerk miteinander verbunden sind und durch Software unterstützt werden, die das Zusammenarbeiten der Computer als integrierte Einheit ermöglicht.

hinzugefügt wird. Daran anschließend soll gezeigt werden, wie man mittels Ameisen das Senden von Datenpaketen über ein Netzwerk optimieren kann. Dieses Problem ist sowohl dynamisch als auch verteilt und wird mittels des AntNet-Algorithmus' gelöst.

Strategien zum Lösen eines dynamischen TSPs

Guntsch und Middendorf beschreiben in einem Artikel[62], auf den im Folgenden Bezug genommen wird, drei Strategien, die zum Lösen eines dynamischen TSP herangezogen werden können. Die Problemstellung wird auf ein dynamisches TSP beschränkt, in welchem während der Lösungssuche entweder ein Knoten hinzugefügt oder gelöscht wird. Auf die Veränderung der Problemstruktur wird sofort mit einer Modifikation der Pheromonmengen reagiert. Dabei ist es wichtig, das richtige Verhältnis zwischen Löschen und Beibehalten der globalen Information, entsprechend der Pheromonspur, zu finden. Handelt es sich nur um eine kleine Veränderung, so liegt das neue Optimum in der Nähe des alten, was dazu führt, dass die alten Informationen zum Teil weiter verwendet werden können. Je größer aber die Veränderung ist, desto mehr Informationen müssen gelöscht werden, da sich andernfalls der Algorithmus von dem alten, nun nur noch lokalen Optimum nicht mehr lösen kann, eventuell dieses noch optimiert, jedoch nicht nach dem globalen Optimum sucht. Das Löschen aller Informationen führt zu einem Performanceverlust, hingegen kann es beim Beibehalten aller Informationen dazu kommen, dass nur ein lokales Optimum gefunden wird. Guntsch und Middendorf haben drei Strategien entwickelt, die einen Kompromiss zwischen den beiden Extrema bilden.

Diese Strategien stellen die einmalige Modifikation der Pheromoninformationen nach der Veränderung der Problemstruktur dar. Dabei wird für jeden Knoten i ein γ-Wert errechnet und die Pheromonmengen aller Kanten (i, j) mit der folgenden Formel verändert:

$$\tau_{ij}(t) \leftarrow (1-\gamma_i)\cdot\tau_{ij}(t) + \gamma_i \frac{1}{n-1}, \tag{3.18}$$

wobei n die Anzahl der Knoten ist. Handelt es sich um ein symmetrisches TSP, so wird der durchschnittliche γ-Wert der Knoten i und j, also $(\gamma_i + \gamma_j) / 2$ statt γ_i verwendet. Für den eingefügten Knoten l entspricht $\gamma_l = 1$, wodurch alle Kanten (l, j) mit $1 / (n\text{-}1)$ initialisiert werden.

[62] Vgl. Guntsch, M. und Middendorf, M. (2001).

Die einzelnen Strategien unterscheiden sich nur in der Art und Weise, wie die γ-Werte berechnet werden. In der ersten Strategie wird jedem γ_i der gleiche Wert ε zugeordnet, wobei ε den Definitionsbereich [0, 1] hat. Dies bedeutet, dass sich alle Pheromonspuren unabhängig von ihrer Nähe zum eingefügten bzw. entfernten Knoten im gleichen Maß verändern. In den anderen beiden Strategien wird auf die Nähe zu diesem Knoten Rücksicht genommen, wobei die Berechnung der Nähe in der zweiten Strategie auf den lokalen Informationen, entsprechend den Entfernungen, und in der dritten Strategie auf den globalen Informationen, entsprechend den Pheromonmengen, basiert. Auf die Vorgehensweise zur Berechnung der γ-Werte dieser beiden Strategien soll hier nicht näher eingegangen werden, diese kann in dem Artikel „Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP" von Guntsch und Middendorf[63] nachgelesen werden.

Zum Testen und Vergleichen der drei Strategien wurden jeweils 1500 Iterationen durchgeführt, wobei nach 250 bzw. 500 Iterationen entweder ein Knoten hinzugefügt oder ein Knoten entfernt wurde. Dabei erwiesen sich die erste und zweite Strategie als etwas effizienter als die dritte Strategie, wobei die guten Resultate der ersten Strategie darauf zurückzuführen sind, dass nur eine einmalige Veränderung der Problemstruktur durchgeführt wurde.

AntNet

AntNet ist ein Routing-Algorithmus, dessen Ziel es ist, ein Netzwerk dahingehend zu kontrollieren, dass alle versendeten Datenpakete den schnellsten Weg vom einem Rechner zu einem anderen unter Berücksichtigung der momentanen Auslastung des Netzwerks gehen. Die Ausgangspunkte, die Ziele und die Anzahl der Datenpakete ist von der Nachfrage im Netzwerk abhängig, und muss für den Algorithmus nicht zuvor bekannt sein. Das Problem ist auf mehrere Rechner, die im Folgenden als Knoten bezeichnet werden, verteilt. Die Dynamik entsteht durch die ständige Veränderung der Auslastung der diversen Knoten.

Für jeden Knoten i existiert eine Routing-Tabelle, in der eingetragen ist, mit welcher Wahrscheinlichkeit $P^i_{j,z}(t)$ ein Paket zu einem bestimmten Nachbarknoten j weitergeleitet wird, unter der Voraussetzung, dass es zu einem bestimmten Zielknoten z gesendet werden soll. Diese Tabelle enthält Werte für alle möglichen Nachbarknoten mit allen möglichen Zielknoten. Des Weiteren existiert pro Knoten eine Matrix Γ_i, die die geschätzte,

[63] Vgl. Guntsch, M. und Middendorf, M. (2001).

durchschnittliche Zeit $\mu_{i \to z}$ von Knoten i zu einem Zielknoten z und die dazugehörige Varianz $\sigma^2_{i \to z}$ enthält.

Der Algorithmus besteht aus zwei Phasen, der Initialisierungsphase und der Hauptphase. In der ersten Phase sind nur Ameisen unterwegs, die die Aufgabe haben, die Routing-Tabellen zu initialisieren. In der darauffolgenden Phase laufen sowohl Ameisen als auch Datenpakete durch das Netzwerk, wobei die Ameisen die Routing-Tabellen ständig aktualisieren und die Datenpakete diese Tabellen zur Wahl des nächsten Knotens heranziehen und diesen mit der jeweiligen Wahrscheinlichkeit wählen. Um den Unterschied zwischen Knoten mit geringer Wahrscheinlichkeit und solchen mit hoher für Datenpakete deutlicher zu machen, wird für diese die Funktion $f(P) = P^{\delta}$ mit $\delta > 1$ angewendet.

In AntNet werden zwei verschiedene Arten von Ameisen eingesetzt, sogenannte „forward ants“ und „backward ants“, wobei letztere gegenüber den ersteren und auch gegenüber Datenpaketen Vorrang haben. Um den Status des Netzwerks kontrollieren zu können, wird in regelmäßigen Zeitintervallen auf jeden Knoten eine „forward ant“ mit einem zufällig bestimmten Ziel gesetzt, wobei die Verteilung der Ziele ungefähr der tatsächlichen Verteilung der Ziele der Datenpakete entsprechen sollte. Die Ameisen laufen dann parallel zueinander durch das Netzwerk, und entscheiden auf Grund der Wahrscheinlichkeit $P^i_{j,z}(t)$ und dem lokalen Auslastungsstatus, zu welchem Knoten sie als nächstes übergehen. Jede „forward ant“ k trägt eine Tabelle $S^k_{s \to z}$ bei sich, in welcher sie pro Knoten i die Nummer des Knotens und die Zeit, die seit dem Verlassen des Startknotens s vergangen ist, einträgt. Wird festgestellt, dass die Ameise im Kreis gegangen ist, also nur noch bereits besuchte Knoten zum Fortsetzen der Tour zur Auswahl stehen, so werden die Informationen von den zuviel gegangenen Knoten aus der Tabelle entfernt. Hat eine „forward ant“ ihren Zielknoten erreicht, so wird eine „backward ant“ generiert, die exakt den umgekehrten Weg vom Zielknoten z zum Startknoten s gehen soll. Um den Weg der „forward ant“ nachvollziehen zu können, wird ihr die Tabelle $S^k_{s \to z}$ übergeben. Bei jedem Zwischenknoten, den die „backward ant“ erreicht, aktualisiert sie die Routing-Tabelle und die Matrix Γ mittels der in der Tabelle enthaltenen Informationen. Durch die Aktualisierungen aller Ameisen wird die Auslastung des Netzwerks indirekt in den Routing-Tabellen festgehalten, nach deren Werten sich die nachkommenden „forward ants“ und die Datenpakete orientieren.

AntNet wurde von Di Caro und Dorigo entwickelt und in einer Reihe von Artikeln[64] vorgestellt. Die erste Präsentation von AntNet erfolgte in dem Artikel „AntNet: A Mobil Agents Approach

[64] Vgl. Di Caro, G. und Dorigo, M. (1997a), (1997b), (1998a), (1998b), (1998c) und (1998d).

to Adaptive Routing"[65], in dem AntNet auch mit anderen Routing-Algorithmen wie zum Beispiel einer einfachen Version des „open short path first"-Algorithmus'[66], dem momentanen Internet-Routing-Algorithmus, oder dem Q-Routing-Algorithmus von Boyan und Littman[67] verglichen wurde und dabei relativ gut abschnitt.

3.5 Zwischenfazit

Abschließend soll hier nochmals auf zwei wichtige Aspekte des zweiten Kapitels näher eingegangen werden. Einerseits sollen die Vor- und Nachteile von Ameisenalgorithmen hervorgehoben werden, und andererseits soll das Adaptieren von ACO-Algorithmen auf andere Problemstellungen veranschaulicht werden.

Ameisenalgorithmen liefern für NP-harte, diskrete Optimierungsprobleme wie zum Beispiel das oben beschriebene „Travelling Salesman Problem" in relativ kurzer Zeit relativ gute Lösungen. Generell lässt sich aber nicht sagen, ob sie hinsichtlich der Performance besser oder schlechter als bestimmte andere Optimierungsalgorithmen wie zum Beispiel Genetische Algorithmen oder Simulated Annealing sind, da die Leistung des jeweiligen Algorithmus' sehr stark von den Gegebenheiten des konkreten Problems abhängig ist. Ein großer Vorteil ist jedoch, dass Ameisenalgorithmen sehr stabil sind, das bedeutet, dass man sie für die verschiedensten Problemstellungen nützen kann. Durch kleine Änderungen kann der Algorithmus auf die jeweilige Problemstruktur angepasst werden. Wie der AntNet-Algorithmus schon gezeigt hat, sind ACO-Algorithmen besonders für dynamische Probleme geeignet, da sie nicht nach einer guten Gesamtlösung suchen, sondern gute Teillösungen markieren und diese dann zu einer guten Gesamtlösung zusammenführen. Auch für verteilte Probleme sind Ameisenalgorithmen gut geeignet, da die Ameisen untereinander nur indirekt kommunizieren. Dies wurde ebenfalls an Hand von AntNet demonstriert. Des Weiteren kann sehr gut auf die Größe eines Problems reagiert werden, indem mehr Ameisen oder sogar mehrere Kolonien von Ameisen eingesetzt werden. Wie in dem Kapitel „Mehrzieloptimierungsprobleme" beschrieben wurde, kann das Einsetzen von mehreren Kolonien die Performance des Algorithmus' verbessern bzw. ermöglicht das Lösen bestimmter Probleme erst. Weitere Vorteile sind, dass Ameisenalgorithmen einfach zu verstehen und leicht zu implementieren sind. Betreffend der Implementierung sollte erwähnt werden, dass die Wahl der Parameter von großer Bedeutung ist, jedoch lassen sich manche

[65] Vgl. Di Caro, G. und Dorigo, M. (1997a).
[66] Vgl. Moy, J. (1995).
[67] Vgl. Boyan, J. A. und Littman, M. L. (1994).

nur sehr schwer oder gar nicht herleiten. In diesem Fall muss auf Erfahrungswerte, und falls diese nicht vorhanden sind, auf Ausprobieren zurückgegriffen werden.

Ein Vorteil, der hier besonders hervorgehoben werden soll, ist das leichte Adaptieren eines ACO-Algorithmus' auf ein bestimmtes Problem. Wie schon in Kapitel 3.4.1 beschrieben, sind Ameisenalgorithmen so ausgelegt, dass die Ameisen durch einen Graphen laufen, was impliziert, dass das Problem als Graph dargestellt werden muss. Die Vorgehensweise dafür wurde an Hand des TSPs, des QAPs und des VRPs in Kapitel 3.4.2 beschrieben. Zuerst muss dabei bestimmt werden, was einem Knoten und was einer Kante entspricht. Danach wird eine Kostenfunktion für die Kanten definiert, aus der die lokalen Informationen der Kanten resultieren. Um den Graphen an das Problem anpassen zu können, besteht die Möglichkeit, Beschränkungen zu definieren. Durch diese Beschränkungen wird festgelegt, welche Sequenzen als gültig gelten. Um die Qualität einer gültigen Sequenz berechnen zu können, muss außerdem eine Gesamtkostenfunktion im Sinne eines quantifizierbaren Ergebnisses, das nicht zwingend auf Geldeinheiten basiert, definiert werden. Ist das Problem als Graph dargestellt, so müssen zur Adaptierung lediglich die definierten Beschränkungen im Algorithmus implementiert werden.

4 Ameisenalgorithmen zum Steuern von Patienten im Planspiel „INVENT“

Im ersten Kapitel dieser Arbeit wurde das von Dipl. oec. Axel Focke entwickelte Planspiel INVENT beschrieben. Dabei blieb die Frage offen, wie die Patienten zu den einzelnen Krankenhäusern zugeordnet werden. Da dieses Problem mit Hilfe eines Ameisenalgorithmus’ gelöst werden soll, wurde im zweiten Kapitel ausführlich auf die verschiedenen Algorithmen eingegangen und beschrieben, wie diese auf verschiedene Probleme angewendet werden können. In diesem Kapitel soll nun gezeigt werden, wie das Problem, die Patienten bestmöglich zu Krankenhäusern zuzuordnen, mit einem Ameisenalgorithmus gelöst werden kann. Dafür wird zuerst das Problem konkret beschrieben und festgelegt, ob es sich um ein statisches oder dynamisches Problem handelt. Ist die Problemstellung klar definiert, so können diverse Realisierungsentscheidungen getroffen werden. Eine wichtige Entscheidung dabei ist, ob das Problem als Mehrzieloptimierungsproblem gelöst werden soll oder ob die im Planspiel vorgegebenen drei Kriterien Entfernung, Eignung und Auslastung in einem Kriterium zusammengefasst werden und lediglich nach diesem einen Kriterium optimiert werden soll. Bei der Zweispielervariante des Planspiels steht zusätzlich noch die Entscheidung offen, ob die Informationen auf beiden zur Eingabe der Spielerdaten verwendeten Rechnern verteilt werden sollen, oder die Daten auf einem Rechner gesammelt werden und der Algorithmus dort mit allen Daten ausgeführt werden soll. Auf Grund dieser Entscheidungen kann die Wahl des Ameisenalgorithmus’ stattfinden. Anschließend wird über die Zweckmäßigkeit des Einsatzes möglicher Erweiterungen, die abhängig vom gewählten Algorithmus sind, entschieden. Steht der Lösungsweg fest, so kann mit der Implementierung begonnen werden. Die Grundlage dafür ist, das vorgegebene Problem als Graph darzustellen. Danach soll an Hand von Pseudocode gezeigt werden, wie der Graph in Quellcode umgesetzt werden kann. Dabei wird explizit auf die Adaptierungen des Algorithmus’ auf das vorliegende Problem eingegangen. Anschließend sollen programmtechnische Details wie die Parametersetzung und das effiziente Speichern von Daten näher erläutert werden. Abschließend wird die Performance des Algorithmus analysiert und auf Grund der Ergebnisse ein geeigneter Abbruchzeitpunkt für den Algorithmus festgelegt.

4.1 Problemstellung

Um ein Problem lösen zu können, muss dieses exakt definiert sein. Dafür soll zunächst das Problem beschrieben und anschließend die Problemstruktur näher betrachtet werden. Aus diesen beiden Unterkapiteln lässt sich dann eine Problemdefinition herleiten.

4.1.1 Problembeschreibung

Wie in Kapitel 2.2.2 schon beschrieben, arbeitet das Programm mit Daten aus der Praxis. Dabei stehen die Patientendaten von sieben Krankenhäusern aus einem Jahr zur Verfügung. Das Problem besteht nun darin, eine gewisse Menge an Patienten bestmöglichst zu einer gewissen Menge an Krankenhäusern zuzuordnen. Dieses Problem kann bis zu drei Mal pro Monat auftreten. Bei der Einspielervariante existiert nur eine Integrierte Versorgungsform, was bedeutet, dass alle Patienten, die der IV beigetreten sind, auf alle sieben Krankenhäuser aufgeteilt werden müssen. Bei der Zweispielervariante existieren hingegen zwei IVs, wodurch zum einen alle Patienten der ersten IV auf die Krankenhäuser der ersten IV und zum anderen alle Patienten der zweiten IV auf die Krankenhäuser der zweiten IV aufgeteilt werden müssen. Unabhängig von der Spielvariante müssen abschließend noch jene Patienten, die keiner IV beigetreten sind, auf alle Krankenhäuser aufgeteilt werden. Für diese drei Probleme soll jeweils der gleiche Algorithmus eingesetzt werden, was bedeutet, dass die Menge der Patienten und die Menge der Krankenhäuser für jeden Aufruf des Algorithmus’ beliebig festgelegt werden kann. Des Weiteren müssen die Parameter des Algorithmus’ für IV-Patienten und Patienten, die keiner IV angehören, unterschiedlich gesetzt werden können.

Die Zuordnung der jeweiligen Patienten zu den jeweiligen Krankenhäusern soll von verschiedenen Kriterien abhängen. Dabei spielt einerseits die Entfernung, die ein Patient von seinem Wohnort zum Krankenhaus zurücklegen muss, eine Rolle. Andererseits soll der Patient in jener Station behandelt werden, die am besten bei seiner Diagnose geeignet ist, und darf auf keinen Fall in ein Krankenhaus eingeliefert werden, in dem keine Station existiert, die ihn adäquat behandeln kann. Des Weiteren muss auf die momentane Auslastung der jeweiligen Stationen Rücksicht genommen werden. Je stärker eine Station ausgelastet ist, desto weniger Patienten sollen dorthin eingewiesen werden. Jedoch soll auch trotz hoher Auslastung immer die Möglichkeit einer Einlieferung offen bleiben. In welchem Ausmaß die jeweiligen Kriterien die Qualität der einzelnen Zuordnung eines Patienten beeinflussen, können die Spieler selbst entscheiden, indem sie die Gewichtungen der Kriterien festlegen.

Ziel ist es jedoch nicht, die bestmögliche Lösung für den einzelnen Patienten zu finden, sondern es wird nach einer globalen Lösung gesucht, in der ein möglichst genau der zuvor von den Spielern vorgegebenen Gewichtung entsprechender Mix aus kurzen Patientenwegen, hohen Eignungskennzahlen und gleichmäßigen Auslastungswerten erreicht wird. Der Unterschied zwischen den beiden Varianten soll an Hand von Abbildung 4.1 in

vereinfachter Weise dargestellt werden. Dabei wird davon ausgegangen, dass es acht Patienten mit der gleichen Diagnose gibt. Des Weiteren existieren drei Krankenhäuser, in denen es jeweils eine Station gibt, in der diese Diagnose behandelt werden kann. In jeder Station sind noch drei Betten frei. Sind diese besetzt, so kann in diesem vereinfachten Beispiel im Gegensatz zum vorliegenden Planspiel kein Patient mehr aufgenommen werden. Die Patienten werden in der Reihenfolge der zugeordneten Nummer eingeliefert. In diesem Beispiel soll nur hinsichtlich der Patientenwege optimiert werden, die Eignung und die Auslastung werden zwecks leichterer Verständlichkeit vernachlässigt. Die Pfeile bezeichnen die Zuordnungen von einem Patienten zu einem Krankenhaus, wobei Pfeile mit punktierter Linie die erste Variante und Pfeile mit strichlierter Linie die zweite Variante darstellen. Pfeile mit einer durchgezogenen Linie zeigen Zuordnungen, die in beiden Varianten gleich sind.

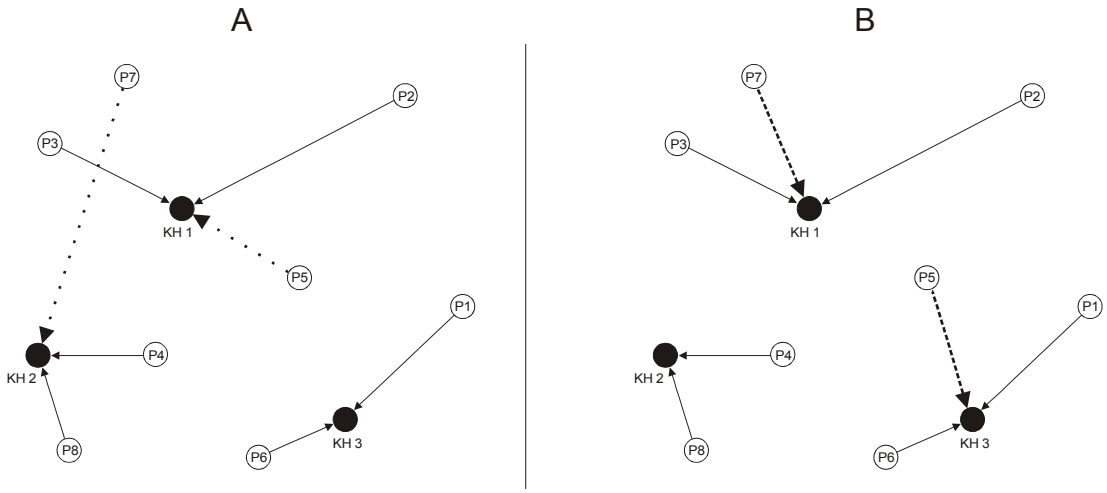

Abbildung 4.1: Darstellung des Unterschieds zwischen
A) dem Optimieren hinsichtlich eines einzelnen Patienten und
B) dem Optimieren hinsichtlich aller Patienten

Wie aus Abbildung 4.1 hervorgeht, führen bis zu dem vierten Patienten beide Varianten zu den gleichen Zuordnungen, jedoch bei dem fünften Patient zeigt sich ein Unterschied. Geht man nach der ersten Variante, entsprechend Abbildung 4.1 A, vor und sucht für jeden Patienten einzeln die bestmögliche Lösung, so muss der fünfte Patient in Krankenhaus 1 eingeliefert werden, da dieses das näherste Krankenhaus ist. Damit ist die Station in diesem Krankenhaus allerdings voll. Beim sechsten Patienten ergeben sich dadurch noch keine Schwierigkeiten, da für diesen Krankenhaus 3 den kürzesten Weg bietet. Jedoch der siebente Patient kann nun nicht mehr in Krankenhaus 1 eingeliefert werden, da die Station dort voll ist. Er muss den mehr als doppelt so langen Weg auf sich nehmen und zu Krankenhaus 2 gehen. Ist es, wie in Abbildung 4.1 B dargestellt, das Ziel, die Patientenwege

ganzheitlich zu minimieren, so wäre es aus globaler Sicht besser, den fünften Patienten in Krankenhaus 3 einzuweisen. Der sechste Patient könnte dann ebenfalls zu Krankenhaus 3 gehen, und der siebente Patient könnte in das für ihn deutlich nähere Krankenhaus 1 eingeliefert werden. In dieser Variante muss der fünfte Patient einen etwas weiteren Weg gehen, jedoch ist dieser kürzer als der zusätzliche Weg, den der siebente Patient in der ersten Variante gehen musste. Daher ist diese Zuordnung aus globaler Sicht besser.

4.1.2 Problemstruktur

Zu einer vollständigen Problemdefinition muss neben der Beschreibung des Problems auch die Problemstruktur feststehen. Diese kann statisch oder dynamisch sein. Um festzustellen, welche dieser beiden Eigenschaften zutrifft, müssen die einzelnen Kriterien näher betrachtet werden. Ist eines der Kriterien dynamisch, was der Fall ist, wenn es sich während des Problemlösungsprozesses verändert, dann handelt es sich um ein dynamisches Problem. Dies soll im Folgenden untersucht werden.

Das Kriterium Entfernung ist in dem behandelten Problem eindeutig statisch, da sich die Entfernung von einem Patienten zu den im Planspiel vorgegebenen Krankenhäusern nicht verändert. Auch die Eignungswerte können als statisch angenommen werden, da diese während des gesamten Spiels gelten. Über das dritte Kriterium, die Auslastung der einzelnen Stationen, kann hier nicht so einfach eine Aussage getroffen werden, da sich dieses nach jeder Zuordnung verändert. Deshalb soll auf dieses Kriterium noch näher eingegangen werden.

In Kapitel 3.4.4 wurden dynamische Probleme beschrieben, darunter das Problem, Datenpakete über ein Netzwerk zu versenden und dabei jeweils den kürzesten Weg hinsichtlich der ständig verändernden Auslastung zu wählen. Diese Problemstellung weist gewisse Ähnlichkeiten mit der in diesem Kapitel behandelten Problemstellung auf, jedoch existieren auch gravierende Unterschiede. Beim Versenden der Datenpakete wird nur der Weg eines einzelnen Datenpakets optimiert. Bei dem Problem, die Patienten den Krankenhäusern zuzuordnen, soll jedoch, wie im vorigen Kapitel beschrieben, auf globaler Ebene optimiert und nicht für jeden Patienten einzeln die beste Lösung gesucht werden. Ein weiterer Unterschied besteht darin, dass vor Beginn des Problemlösungsprozesses nicht bekannt ist, welche Datenpakete von welchem Startknoten zu welchem Zielknoten versendet werden sollen. Die Patientendaten sind jedoch im vorliegenden Fall des Planspiels sehr wohl vorher bekannt, andernfalls könnte man auch nicht aus globaler Sicht optimieren.

Betrachtet man das Kriterium Auslastung näher, so fällt auf, dass sich dieses nur innerhalb einer Lösung verändert. Versucht man, eine neue Lösung zu finden, so geht man wieder von der Anfangsauslastung aus. Würde man die Beschränkung, dass alle Patienten unabhängig von der Auslastung aufgenommen werden müssen, vernachlässigen und nur dahingehend unterscheiden, ob eine Station voll oder noch mindestens ein Bett frei ist, so könnte man die Problemstellung mit dem Travelling Salesman Problem vergleichen, in dem auch jeder Knoten als nächster Aufenthaltsort ausfällt, der schon zuvor besucht worden ist. Das TSP ist jedoch kein dynamisches Problem. Dynamisch wird es erst dann, wenn ein Knoten, wie in Kapitel 3.4.4 beschrieben, während des Lösungsprozesses entfernt bzw. hinzugefügt wird. Legt man diese Erkenntnis auf das hier behandelte Problem um, so bedeutet dies, dass es sich dann um ein dynamisches Problem handeln würde, wenn eine Station oder ein ganzes Krankenhaus während des Problemlösungsprozesses geschlossen bzw. eine neue Station oder auch ein neues Krankenhaus eröffnet werden würde. Auch Veränderungen der Stationsgröße, entsprechend dem Erhöhen oder Senken der Anzahl an Planbetten, lassen auf ein dynamisches Problem schließen.

Auf Grund dieser Überlegungen handelt es sich bei dem Kriterium Auslastung um ein statisches Kriterium. Da damit alle Kriterien des Problems als statisch identifiziert wurden, kann man daraus schließen, dass es sich bei dem hier behandelte Problem um ein statisches Problem handelt.

4.1.3 Resultierende Problemdefinition

Das Problem lässt sich auf Grund der beiden vorhergehenden Kapitel wie folgt definieren:

> Bei dem hier beschriebenen Problem handelt es sich um ein statisches Problem, in dem die bestmögliche Zuordnung einer vorgegebenen Menge von Patienten zu einer vorgegebenen Anzahl an Krankenhäusern gesucht ist, wobei sich bestmöglich auf das globale Optimieren der drei Kriterien Entfernung, Eignung und Auslastung bezieht, die zusätzlich mit einer vom Spieler bestimmten Wertigkeit gewichtet werden.

Auf Grund dieser Definition können nun alle nötigen Realisierungsentscheidungen getroffen und damit der Lösungsweg festgelegt werden.

4.2 Festlegen des Lösungsweges

Die wichtigste Entscheidung beim Festlegen des Lösungsweges ist jene, in der der Ameisenalgorithmus bestimmt wird. Um diese Entscheidung treffen zu können, müssen jedoch zuvor andere Realisierungsentscheidungen getroffen werden. Eine davon ist, ob das Problem als Mehrzieloptimierungsproblem realisiert oder durch das Zusammenfassen der drei Kriterien in ein einzelnes Kriterium gelöst werden soll. Diese Wahlmöglichkeit besteht deshalb, weil einerseits mehrere Kriterien existieren, andererseits aber auch Gewichtungen vorhanden sind, die die Auswirkungen der einzelnen Kriterien auf die Lösung bestimmen. Eine weitere zu treffende Entscheidung, die nur die Zweispielvariante betrifft, ist, ob das Problem als verteiltes Problem realisiert werden soll, oder ob es stets auf einem Rechner gelöst werden soll. Sind diese beiden Entscheidungen getroffen, so kann der Ameisenalgorithmus gewählt werden. Abschließend muss noch entschieden werden, welche Erweiterungen für den ausgewählten Algorithmus möglich sind, und ob der Einsatz dieser in dem beschriebenen Problem sinnvoll ist. Auf all diese Entscheidungen soll im Folgenden detaillierter eingegangen werden.

4.2.1 Entscheidung des Optimierungsweges

Bei dieser Entscheidung geht es darum, ob das Problem nach einem Ziel optimiert oder als Mehrzieloptimierungsproblem gelöst werden soll. Auf den ersten Blick erscheint die Lösung als Mehrzieloptimierungsproblem durchaus sinnvoll, da es drei Kriterien gibt, nach denen optimiert werden soll. Dabei soll kein Kriterium einem anderen bevorzugt werden. Wie in Kapitel 3.4.3 beim Beschreiben von Mehrzieloptimierungsproblemen schon dargestellt worden ist, können Probleme dieser Art mit Hilfe der Single Colony-Methode oder der Multi Colony-Methode gelöst werden. Diese Vorgehensweise wird auch als „Französische Schule“[68] bezeichnet. Im Gegensatz dazu existiert die „Amerikanische Schule“[69], bei der a priori Gewichtungen vorgenommen werden müssen. Durch diese Gewichtungen kann eine Kostenfunktion erstellt werden und nach den daraus resultierenden Kosten wird schließlich optimiert. Im vorliegenden Planspiel ist die Eingabe von a priori Gewichtungen durch die Spieler jedoch ohnehin als fester Bestandteil vorgegeben, so dass das Problem sowohl als Mehrzieloptimierungsproblem als auch als Optimierungsproblem nach einem Ziel realisiert werden kann. Diese beiden Alternativen und auch deren Vor- und Nachteile für das zu lösende Problem sollen im Folgenden näher beschrieben werden.

[68] Vgl. Keeney, R. und Raiffa, H. (1993).
[69] Vgl. Vincke, P. (1992).

Optimieren nach mehreren Zielen

Bei einem Mehrzieloptimierungsproblem lassen sich die Lösungen der Ameisen, wie zuvor schon bei der Beschreibung der Multi Colony-Methode gezeigt, in einem Koordinatensystem darstellen. An Hand der dort beschriebenen Problemstellung, in der allerdings nach nur zwei Kriterien optimiert worden ist, soll im Folgenden die Konsequenzen des Festlegens von Gewichtungen der einzelnen Kriterien gezeigt werden.

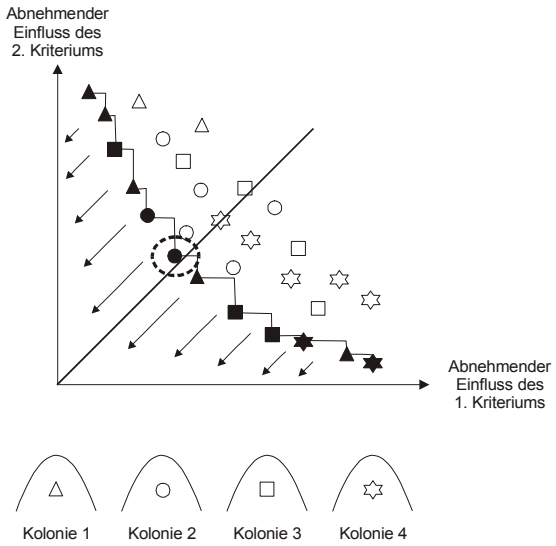

Abbildung 4.2: Darstellung eines Optimierungsproblems mit zwei Kriterien in der Multi Colony-Methode: Die Pfeile zeigen dabei die Richtung, in welche die Lösungen verbessert werden. Durch den strichlierten Bereich wird unter der Voraussetzung, dass für beide Kriterien die gleiche Gewichtung festgelegt wurde, die optimale Lösung markiert.

Da die Ameisen ihre Lösungen im Laufe des Algorithmus’ immer mehr verbessern, verändern sich die nicht-dominierten Lösungen dahingehend, dass der Einfluss zumindest eines der beiden Kriterien steigt. Grafisch gesehen bedeutet dies, wie in Abbildung 4.2 durch die Pfeile verdeutlicht wird, dass sich die nicht-dominierten Lösungen immer mehr den Achsen nähern. Dabei ist zu beachten, dass in viele verschiedene Richtungen optimiert wird. Hat man jedoch eine bestimmte Gewichtung vorgegeben, beispielsweise, dass die Wertigkeit des ersten Kriteriums gleich groß ist wie die des zweiten, so werden am Ende der Berechnungen nur jene Lösungen gewählt, die auf jener Geraden liegen, die durch den Nullpunkt geht und im 45° Winkel ansteigt. Befindet sich keine Lösung exakt auf dieser Geraden, so wird jene nicht-dominierte Lösung gewählt, die diesem am nähersten ist. In Abbildung 4.2 ist die ausgewählte Lösung durch den strichlierten Bereich hervorgehoben. Durch den Winkel von 45° ist der Einfluss des ersten Kriteriums immer gleich groß wie der

Einfluss des zweiten. Liegt eine andere Verteilung vor, so muss auch der Winkel der Gerade entsprechend anders gewählt werden.

Das in diesem Kapitel behandelte Problem wird im Gegensatz zu dem Problem in der Abbildung 4.2 nach drei Kriterien optimiert, die Vorgehensweise zum Auffinden der besten Lösung bei festgelegten Gewichtungen erfolgt jedoch analog. Zur grafischen Darstellung der Lösungen muss jedoch ein dreidimensionales Koordinatensystem herangezogen werden.

Der Nachteil beim Optimieren nach mehreren Zielen bei zuvor festgelegten Gewichtungen ist, dass trotz des Wissens, dass nur bestimmte Lösungen, beispielsweise jene, die auf der zuvor beschriebenen Geraden liegen, gesucht sind, die Lösungen auch in andere Richtungen optimiert werden. Dies kostet erheblich mehr Zeit als das Optimieren einer Lösung in eine bestimmte Richtung, was im Folgenden detaillierter beschrieben wird.

Optimieren nach einem Ziel

Beim Optimieren nach einem Ziel existiert lediglich ein Kriterium, nach dem optimiert wird. In dem hier behandelten Problem soll allerdings nach drei Kriterien optimiert werden. Da aber durch die vorgegebenen Gewichtungen der einzelnen Kriterien feststeht, welchen Einfluss das jeweilige Kriterium auf die Qualität der Lösung hat, können die drei Kriterien multipliziert mit ihrer Gewichtung in einem Wert, der im Folgenden mit dem Begriff „Kosten“ bezeichnet wird, zusammengefasst werden. Formal gesehen lässt sich dies folgender Maßen ausdrücken:

$$K = w_1 \cdot b_1 + w_2 \cdot b_2 + w_3 \cdot b_3 , \tag{4.1}$$

wobei K die Kosten bezeichnet, w den Gewichtungen der Kriterien entspricht und b die Werte der jeweiligen Kriterien darstellt. Die Indizes repräsentieren das entsprechende Kriterium, wobei 1 für die Entfernung, 2 für die Eignung und 3 für die Auslastung steht. Bei den Werten der Kriterien ist außerdem zu beachten, dass diese einheitlich normiert sein müssen. In dem hier behandelten Problem wird dafür ein Wertebereich von 0 bis 100 für jedes Kriterium verwendet, wobei 0 dem besten und 100 dem schlechtesten Wert entspricht.

Da die drei Kriterien Entfernung, Eignung und Auslastung in den Kosten zusammengefasst sind, kann nicht mehr in mehrere, sondern nur noch in eine Richtung optimiert werden. Beide Methoden würden die optimale Lösung finden, ließe man sie lange genug laufen, jedoch würde die hier beschriebene Methode wesentlich schneller zu der Lösung kommen. Die andere Methode ist für Probleme, in denen bereits Gewichtungen vorgegeben sind, daher

schlechter geeignet. Dies führt zu der Entscheidung, dass jene Methode implementiert werden soll, die lediglich nach einem Ziel optimiert.

4.2.2 Entscheidung über die Verteilung des Problems auf mehrere Rechner

In diesem Kapitel soll entschieden werden, ob das Problem als verteiltes Problem realisiert, oder ob es von einem Rechner aus gelöst werden soll. Bei der Einspielervariante stellt sich diese Frage nicht, da nur ein Rechner in das Spiel involviert und das Verteilen des Problems daher unmöglich ist. Für die Zweispielvariante muss diese Entscheidung jedoch getroffen werden.

Geht man davon aus, dass jeder Rechner[70] die Krankenhäuser einer IV verwaltet und eine Liste jener Patienten, die der jeweiligen IV angehören und im momentanen Monat erkrankt sind, an den jeweiligen Rechner übergeben wird, so kann das Problem der Zuordnung der jeweiligen IV-Patienten zu den Krankenhäusern der jeweiligen IV auf dem jeweiligen Rechner gelöst werden. Dies stellt noch kein verteiltes Problem dar, jedoch wenn es um die Zuordnung der restlichen, keiner IV angehörigen Patienten geht, dann sind beide Rechner in den Problemlösungsprozess involviert, da die Patienten auf alle sieben Krankenhäuser aufgeteilt werden müssen.

Ein verteiltes Problem würde jedoch eine Menge an Schwierigkeiten mit sich bringen. Angefangen davon, dass eine ständige Verbindung zwischen den beiden Rechnern bestehen müsste, bis hin zu der Tatsache, dass die Implementierung des Algorithmus’ schwieriger ist, würde außerdem die Berechnungszeit durch das fortwährende Versenden von Daten ansteigen.

Aus diesen Gründen scheint es sinnvoller, das Problem auf einem Rechner zu lösen. Dies wird dadurch ermöglicht, dass ein Rechner bestimmt wird, auf dem das anfallende Problem gelöst werden soll. Alle Daten, die dafür benötigt werden, werden entweder über ein Netzwerk oder aber auch per Diskette an den jeweiligen Rechner übergeben. Jener Rechner, der die Informationen liefert, erstellt eine Datei mit den zu übergebenen Informationen und speichert diese in einem zuvor bestimmten Verzeichnis. Der Rechner, der die Informationen erwartet, fragt in regelmäßigen Intervallen ab, ob die Datei schon im zuvor bestimmten Verzeichnis gespeichert worden ist. Als mögliches Verzeichnis kann auch ein

[70] Genauer betrachtet verwaltet nicht der Rechner sondern das Programm auf dem Rechner die Krankenhäuser, auf diese genaue Betrachtung soll aber der Einfachheit halber verzichtet werden.

Diskettenlaufwerk angegeben werden, wodurch die Übergabe der Informationen per Diskette erfolgen und das Vorhandensein eines Netzwerkes nicht mehr zwingend nötig ist.

4.2.3 Wahl des Ameisenalgorithmus’

Da nun das zu lösende Problem hinsichtlich seiner Eigenschaften vollständig charakterisiert ist, kann die Wahl des Ameisenalgorithmus’ erfolgen. In den vorigen Kapiteln ist bereits hergeleitet worden, dass es sich um ein großdimensioniertes, statisches Problem handelt, welches nach einem Ziel optimiert werden soll. Außerdem soll der Algorithmus unabhängig von der Spielvariante nur jeweils auf einem Rechner ausgeführt werden.

Wie in Kapitel 3.3 beschrieben, gibt es eine Menge von Ameisenalgorithmen, die auf ein solches Problem angewendet werden können. Bei näherer Betrachtung erweisen sich jedoch einige als nicht oder nur schlecht geeignet für das hier zu Grunde liegende Problem. Der Basisalgorithmus Ant System beispielsweise erscheint nicht sinnvoll, da in dem beschriebenen Problem eine große Anzahl von Patienten zu maximal sieben Krankenhäusern zugeordnet werden müssen und, wie schon zuvor beschrieben, AS bei großen Optimierungsproblemen schlechte Ergebnisse liefert. Die Erweiterung AS mit „elitist strategy“ wird in der Literatur lediglich im Zusammenhang mit AS erwähnt. Erweiterte Algorithmen basierend auf AS scheinen daher auch Verbesserungen des ASs mit „elitist strategy“ zu sein. Dies zeigt sich beispielsweise bei dem AS_{rank}-Algorithmus, der sowohl bessere Ergebnisse als AS als auch bessere Ergebnisse als AS mit „elitist strategy“ liefert.[71] AS_{rank} erscheint jedoch auch nicht geeignet für das vorliegende Problem, da dieser Algorithmus versucht, AS dahingehend zu verbessern, dass bei vielen Knoten mit ähnlich hohen Kosten jene Knoten mit deutlich höherer Wahrscheinlichkeit gewählt werden, die den besten Wert aufweisen. Auf Grund der Übergangsregel ist diese Vorgehensweise dann besonders effektiv, wenn es viele mögliche Übergänge gibt, und einige davon sehr gut geeignet sind. Gibt es beispielsweise 100 mögliche Übergänge und 50 Knoten weisen geringe, jedoch ähnliche Kosten auf, so würden laut Übergangsregel in AS alle 50 Knoten mit annähernd der gleichen Wahrscheinlichkeit gewählt werden. In diesem Fall erscheint es sinnvoll, zwischen diesen 50 Knoten genauer zu unterscheiden und diesen Knoten Ränge zuzuordnen. In dem vorliegenden Problem gibt es jedoch nur sieben Übergangsmöglichkeiten, wodurch die Werte der einzelnen Knoten ohnehin schon einen größeren Einfluss auf ihre Wahrscheinlichkeit haben. Aus diesem Grund erscheint das zusätzliche Gewichten der Werte mit den entsprechenden Rängen als wenig sinnvoll und

[71] Vgl. Bullnheimer, B. / Hartl, R. F. / Strauss, C., *Central European Journal for Operations Research and Economics* 7/1 (1999) S. 35.

auch AS_{rank} scheidet aus der Wahl des Algorithmus' aus. Ebenso nicht besonders gut geeignet erscheint der ANTS-Algorithmus, da dieser bis zum momentanen Zeitpunkt nur auf das Quadratic Assignment Problem angewendet wurde und Erfahrungswerte für das Anwenden auf andere Problemstellungen fehlen. Der als letztes beschriebene FANT-Algorithmus stellt sich ebenfalls als ungeeignet für das hier behandelte Problem heraus, da dieser darauf spezialisiert ist, bei kurzer Berechnungsdauer relativ gute Ergebnisse zu liefern. Wie aus den Versuchen von Taillard und Gambardella ersichtlich wird, liefert der FANT-Algorithmus bei 100-maligem Durchlaufen einer Ameise schon ab einer Berechnungsdauer von 10 Sekunden, meist jedoch schon bei 5 Sekunden ein schlechteres Ergebnis als der HAS-Algorithmus, der in diesem Zusammenhang auch repräsentativ für andere auf AS basierenden Erweiterungen ist.[72] Bei 1000-maligem Durchlaufen einer Ameise schneidet der FANT-Algorithmus bei Berechnungen bis zu 13 Sekunden gleich gut, bei höherer Berechnungsdauer allerdings immer schlechter als der HAS-Algorithmus ab.[73] Da das hier zu Grunde liegende Problem relativ groß ist, würde das Ergebnis des FANT-Algorithmus nach wenigen Sekunden wahrscheinlich besser sein als jenes von anderen Ameisenalgorithmen, jedoch wäre die Qualität der Lösung selbst noch relativ schlecht. Um eine halbwegs gute Qualität der Lösung zu erhalten, wurde eine Berechnungsdauer von einigen Minuten einkalkuliert. Bei solch langer Berechnungsdauer liefert jedoch der FANT-Algorithmus schlechte Ergebnisse, was dazu führt, dass auch dieser Algorithmus als nicht geeignet für das vorliegende Problem erscheint.

Auf Grund dieser Überlegungen bleiben ACS, HAS und MMAS über. Der HAS-Algorithmus basiert auf der Idee, eine lokale Suchmethode mit einem ACO-Algorithmus zu kombinieren. Da diese Idee auch in MMAS verwirklicht wird und in ACS einen lokale Suchmethode als Erweiterung implementiert werden kann, scheidet auch dieser Algorithmus aus. Neben ACS und MMAS wäre auch jene Möglichkeit denkbar, in der der ACS-Algorithmus als Basis für MMAS herangezogen wird. Auf Grund der einfacheren Implementierung und der leichteren Verständlichkeit für die Spieler fällt die Wahl jedoch auf den ACS-Algorithmus.

Eine mögliche Erweiterung des ACS-Algorithmus, die speziell für große Optimierungsprobleme geeignet ist, ist das zusätzliche Verwenden von Kandidatenlisten. Außerdem kann der Algorithmus mit einer lokalen Suchmethode verbessert werden. Auf die Frage, ob die Implementierung dieser Erweiterungen sinnvoll ist, soll in den nächsten zwei Kapiteln näher eingegangen werden.

[72] Vgl. Taillard, E. und Gambardella, L. M. (1997) S. 12.
[73] Vgl. Taillard, E. und Gambardella, L. M. (1997) S. 13.

4.2.4 Entscheidung über den Einsatz von Kandidatenlisten

Der Grundgedanke von Kandidatenlisten ist, die Anzahl der Übergangsmöglichkeiten eines Knotens zu anderen Knoten zu beschränken. Wie schon in Kapitel 3.3.2 bei der Beschreibung des ACS-Algorithmus’ erklärt wurde, existiert dabei für jeden Knoten eine Kandidatenliste. In der Liste eines Knotens wird notiert, welche Knoten sich am besten für einen Übergang eignen. Welcher dieser Knoten dann für einen Übergang ausgewählt wird, entscheidet die Übergangsregel. Dies bedeutet, dass die Menge der Knoten, die für einen Übergang gewählt werden können, eingeschränkt wird und vorerst nur jene Knoten zur Auswahl stehen, die am besten für einen Übergang geeignet sind. Darf keiner dieser Knoten auf Grund einer bestimmten Beschränkung ausgewählt werden, so wird jener Knoten aus der Menge der Knoten, die nicht in der Kandidatenliste enthalten sind, gewählt, der die geringsten Kosten aufweist.

Beim Travelling Salesman Problem ist der Einsatz von Kandidatenlisten sinnvoll, da dort jeder Knoten als nächster Knoten gewählt werden kann, sofern er noch nicht besucht worden ist. Schränkt man die Menge auf Grund der Entfernungen ein, so entsteht, sofern die Anzahl der Knoten in einer Kandidatenliste nicht zu gering ist, auch kaum ein Qualitätsverlust, da Städte, die sehr weit entfernt sind, ohnehin kaum ausgewählt werden.

Bei dem in dieser Arbeit behandelten Problem verhält sich dies allerdings anders, da für jeden Patient nur maximal sieben Krankenhäuser zur Auswahl stehen. Das Einschränken dieser sieben Wahlmöglichkeiten macht jedoch wenig Sinn, da dies eher zu einem Qualitätsverlust der Lösung als zu einem Zeitgewinn führen würde. Aus diesem Grund wird diese Möglichkeit der Erweiterung des ACS-Algorithmus nicht implementiert.

In Kapitel 2.2.2 bei der Beschreibung der Datenaufbereitung wurde eine zweite Möglichkeit, die wegen zu hohen Rechenaufwandes nicht realisiert wurde, erwähnt, nach der jedem Patienten alle Stationen eines Krankenhauses zur Auswahl stehen könnten, die zumindest theoretisch die Krankheit des Patienten behandeln können. Bei dieser Variante wäre die Anzahl an möglichen Übergängen dann so groß, dass das Einsetzen von Kandidatenlisten sehr wohl als sinnvoll anzusehen wäre.

4.2.5 Entscheidung über den Einsatz von lokalen Suchmethoden

Bevor Überlegungen hinsichtlich des Einsatzes von lokalen Suchmethoden in dem hier behandelten Problem angestellt werden, soll eine kurze Einführung in das Thema „Lokale Suchmethoden“ gegeben werden.

Allgemeines zum Verständnis von lokalen Suchmethoden

Lokale Suchmethoden basieren darauf, eine vorhandene Lösung durch kleine Veränderungen iterativ zu verbessern. Dabei spielt die Nachbarschaft der momentanen Lösung, entsprechend den möglichen Veränderungen, eine wesentliche Rolle. Des Weiteren ist für die Suchmethode entscheidend, wann eine Veränderung akzeptiert und somit die vorhandene Lösung durch eine neue Lösung ersetzt wird. Beendet wird eine lokale Suchmethode dann, wenn keine Möglichkeit mehr zum Ersetzen der momentanen Lösung gefunden werden kann. Diese Lösung ist dann, wie der Name „lokale Suchmethode“ schon sagt, eine lokale, jedoch nicht zwingend eine globale Lösung.

Die bekanntesten lokalen Suchmethoden, die zum Kombinieren mit Ameisenalgorithmen eingesetzt werden, sind die 2-opt-Methode[74], die 3-opt-Methode[75] und die Lin-Kernighan-Methode[76], bei denen eine Veränderung aus dem Austauchen von zwei, drei oder einer variablen Anzahl von Kanten besteht. Da das Austauschen von drei oder mehreren Kanten aufwendiger ist, als das Austauschen von zwei Kanten und das in diesem Kapitel behandelte Problem relativ groß ist, soll die 3-opt-Methode und die Lin-Kernighan-Methode vernachlässigt und nur die 2-opt-Methode in Betracht gezogen werden.[77]

Beschreibung der 2-opt-Methode

Die 2-opt-Methode beruht auf dem Vertauschen zweier Kanten. Dies wird so lange durchgeführt, bis keine zwei Kanten mehr existieren, deren Austausch eine Verbesserung erzielen würde. Die vorliegende Lösung wird dann 2-optimal genannt, und die lokale Suchmethode wird beendet.

[74] Vgl. Lin, S., *Bell Systems Journal* 44 (1965).
[75] Vgl. Lin, S., *Bell Systems Journal* 44 (1965).
[76] Vgl. Lin, S. und Kernighan, B. W., *Operations Research* 21 (1973).
[77] Vgl. Dorigo, M. und Gambardella, L. M., *IEEE Transactions on Evolutionary Computation* 1 (1997a) S. 61.

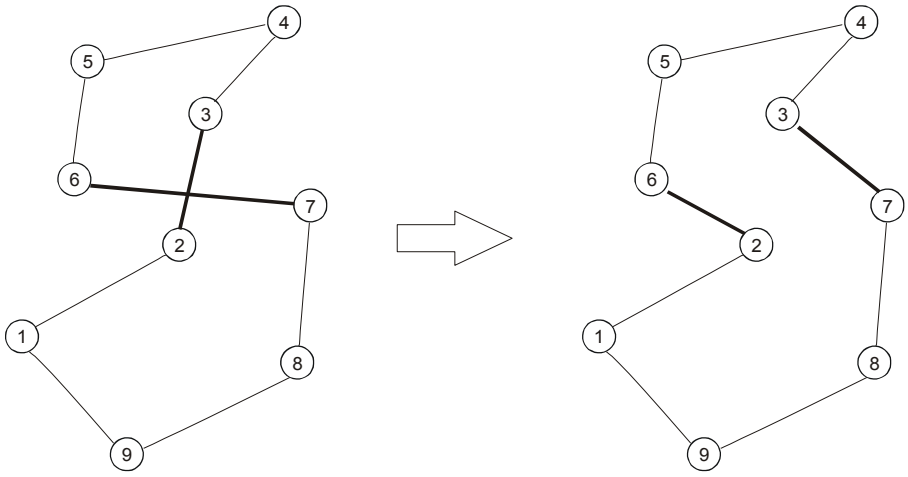

Abbildung 4.3: Austausch zweier Kanten in der 2-opt-Methode

Abbildung 4.3 soll den Austausch zweier Kanten an Hand des Travelling Salesman Problems zeigen. Dabei werden zwei Kanten ausgewählt und gelöscht. Zwischen den verbleibenden vier Knoten werden dann zwei neue Kanten erstellt, wobei es dafür bei der 2-opt-Methode immer nur eine einzige Möglichkeit gibt. Angenommen, es werden wie in Abbildung 4.3 die Kanten zwischen Knoten 2 und Knoten 3 und die Kanten zwischen Knoten 6 und Knoten 7 entfernt, so muss eine Kante zwischen Knoten 2 und Knoten 6 und die zweite Kante zwischen Knoten 3 und Knoten 7 erstellen werden, um eine korrekte Tour zu erzielen. Die zweite Möglichkeit wäre, Knoten 2 mit Knoten 7 und Knoten 3 mit Knoten 6 zu verbinden, dann würden sich jedoch zwei voneinander getrennte Touren ergeben, was beim TSP nicht als gültige Lösung akzeptiert wird.

Im Folgenden soll dargestellt werden, welche Auswirkungen das Einbauen der 2-opt-Methode auf die Lösung des ACS-Algorithmus’ in dem vorgegebenen Problem hat und ob dadurch das Implementieren dieser Methode sinnvoll ist.

Auswirkung der 2-opt-Methode auf das vorgegebene Problem

Wie schon in Kapitel 3.3.3 bei der Beschreibung des Hybrid Ant Systems erläutert wurde, hat es sich als sinnvoll erwiesen, die lokale Suchmethode auf jede Tour, die eine Ameise gefunden hat, anzuwenden. Da der Berechnungsaufwand bei der 2-opt-Methode bei $O(n^2)$ liegt und das hier vorliegende Problem eine hohe Anzahl an Knoten besitzt, würde das Ausführen der 2-opt-Methode bis zu einer 2-optimalen Lösung sehr lange dauern. Eine mögliche Vorgehensweise, die auch Lo[78] angewendet hat, um Genetische Algorithmen mit

[78] Vgl. Lo, C. D., et al. (2001).

der 2-opt-Methode zu verbessern, ist, nur auf einen gewissen Prozentsatz aller Kanten die 2-opt-Methode anzuwenden.

Für das hier zu Grunde liegende Problem bedeutet das Austauschen von Kanten, dass zwei Patienten ihre Stationen tauschen. Voraussetzung dafür ist, dass Patienten mit entsprechenden Diagnosen jeweils in der anderen Station behandelt werden können. Der erste Patient wird dabei jeweils zufällig gewählt, der zweite kann entweder auch zufällig gewählt werden, wobei die zuvor beschriebenen Bedingung eingehalten werden muss, oder es wird jener Patient zum Austausch gewählt, der sich am besten dafür eignet bzw. der die Gesamtkosten am stärksten verringert.

Der große Unterschied zwischen beispielsweise dem TSP, das Lo für seine Versuche herangezogen hat, und dem hier vorliegenden Problem ist die Berechnung der Gesamtkosten nach dem Austausch zweier Kanten. Beim TSP bestehen die Gesamtkosten nur aus der zurückgelegten Entfernung. Um diese nach dem Austausch zweier Kanten zu berechnen, müssen nur die Entfernungen der Kanten, die gelöscht wurden, von den Gesamtkosten subtrahiert und die Entfernungen der Kanten, die neu hinzugefügt wurden, zu den Gesamtkosten addiert werden. Bei dem in diesem Kapitel beschriebenen Problem ist dies jedoch nicht so einfach. Würde es nur die Kriterien Entfernung und Eignung geben, könnte man ähnlich vorgehen, jedoch das Kriterium Auslastung macht ein solches Vorgehen unmöglich. Der Grund dafür liegt darin, dass sich die Auslastung nach jeder Einlieferung eines Patienten für die jeweilige Station verändert. Liefert man nun den als erstes ausgewählten Patienten in eine andere Station ein, so verändern sich die Auslastungswerte für die nachfolgenden Patienten. Erst wenn der als zweites ausgewählte Patient in die Station des ersten eingeliefert wird, stimmen die Auslastungswerte für die nachfolgenden Patienten wieder überein. Dies bedeutet, dass man die Kosten aller Patienten, die zwischen den beiden ausgewählten Patienten eingeliefert werden, neu berechnen muss. Im schlimmsten Fall, wenn der allererste Patient und der allerletzte Patient ausgewählt wird, müssen die Gesamtkosten noch mal von neuem berechnet werden. Aber auch in durchschnittlichen Fällen, in denen nur die Hälfte der Gesamtkosten neu errechnet werden muss, macht eine ganzheitlich neue Berechnung der Gesamtkosten mehr Sinn als jeweils die einzelnen Kosten der vorhandenen Lösung zu subtrahieren und die Kosten der neuen Lösung zu addieren.

Um den Aufwand dieser Berechnungen abschätzen zu können, wurde versuchsweise eine 2-opt-Methode in den ACS-Algorithmus implementiert, in der die Stationen von 100 zufällig

gewählten Patienten mit 100 anderen zufällig gewählten Patienten ausgetauscht wurden, wobei auf die Beschränkung, dass die Krankheiten der Patienten auch in der jeweils anderen Station behandelbar sein müssen, Rücksicht genommen wurde. Diese Zahl an Patienten entspricht ungefähr 1 % aller Patienten im getesteten Monat, hingegen hat Lo in seinem Versuch 20 % aller Knoten ausgewählt. Durch die geringe Anzahl an Vertauschungen und auch wegen der zufälligen Wahl beider Patienten, verbessert sich die Qualität der Lösung nur gering bis kaum, jedoch die Berechnungszeit dauert um das dreizehnfache länger. Um die Qualität der Lösung zu verbessern, müssten vermutlich auch 20 % aller Patienten ausgewählt werden. Alleine dadurch würde die Berechnungsdauer jedoch nochmals 20 mal länger dauern. Das Ausfinden eines Patienten, dessen Station sich am besten für einen Austausch mit der Station des zufällig gewählten Patienten eignet, ist undenkbar, da dies bedeuten würde, dass dabei alle Patienten durchgegangen und bei jedem Patient die Gesamtkosten des Austausches errechnet werden müssten.

Da sich auf Grund dieses Versuchs herausgestellt hat, dass die 2-opt-Methode bei dem vorliegenden Problem für geringe Qualitätsverbesserungen eine sehr hohe Steigerung der Berechnungszeit bewirkt, soll auf den Einsatz dieser Methode verzichtet werden. Auch andere Suchmethoden werden nicht weiter auf die Zweckmäßigkeit in dem vorgegebenen Problem untersucht, da die 2-opt-Methode schon als eine der schnellsten Methoden hinsichtlich der benötigten Berechnungszeit gilt.

Die einzige Möglichkeit, die Methode so umzubauen, dass die Berechnungszeit jener im TSP nahe kommt, wäre, nur jene Patienten in einen Tausch zu involvieren, die direkt nacheinander eingeliefert worden sind. Nur so wäre es möglich, die Gesamtkosten lediglich durch das Subtrahieren der Kosten der alten Kanten und das Addieren der Kosten der neuen Kanten zu berechnen. Da der ACS-Algorithmus aber darauf ausgelegt ist, besonders in der Umgebung einer guten Lösung zu suchen, würde die lokale Suchmethode nur Lösungen liefern, die die Ameisen ohnehin finden würden. Daher erscheint auch diese Methode für das vorliegende Problem wenig sinnvoll.

4.3 Implementierung des Algorithmus’

In diesem Kapitel soll die Implementierung des zuvor ausgewählten Algorithmus’ erfolgen. Dazu wird als erster Schritt, wie schon beim Einsatz von Ameisenalgorithmen im Kapitel 3.4 beschrieben wurde, das Problem als Graph dargestellt. Danach wird die Beschreibung des Quellcodes in Form von Pseudo-Code erläutert und die Unterschiede zu dem in Kapitel 3.3.2 beschriebenen ACS-Algorithmus für die Futtersuche hervorgehoben. Anschließend sollen

programmiertechnische Einzelheiten wie die Parametersetzung und die effiziente Speicherung von Daten erläutert werden. Abschließend wird die Performance des Algorithmus’ analysiert und die Wahl des Abbruchzeitpunktes vorgenommen.

4.3.1 Darstellung des Problems als Graph

Um das hier vorliegende Problem als Graph zu beschreiben, soll auf eine nicht dem Standard entsprechende Art der Darstellung zurückgegriffen werden. Da sich die Entfernungen nur auf die Krankenhäuser, die Eignungen und Auslastungen jedoch auf die Stationen in den jeweiligen Krankenhäusern beziehen, sollen die Krankenhäuser und deren Stationen grafisch als solche erkennbar sein und werden, wie in Abbildung 4.4 ersichtlich, auch als solche dargestellt. Die Patienten werden hingegen wie üblich durch einen Kreis repräsentiert. Kanten existieren von einem Patienten zu der jeweils bestgeeigneten Station in jedem Krankenhaus, wobei sich bestgeeignet ausschließlich auf die Eignung der Station bezieht. Weitere Kanten existieren jeweils von dieser Station zum nächsten Patienten, der durch die Reihenfolge der Aufnahme entsprechend dem Aufnahmedatum und der Aufnahmeuhrzeit vorgegeben ist. Es kann auch vorkommen, dass Kanten von einer Station zu mehreren Patienten führen, jedoch muss in diesem Fall zwingend immer jene Kante gewählt werden, die zu dem in der Reihenfolge nächsten Patienten führt. Aus Gründen der Übersichtlichkeit wurden jedoch die Kanten von Stationen zu Patienten nicht in Abbildung 4.4 eingezeichnet, die Reihenfolge wird auf Grund der Beschriftung und auch der Position der Knoten ersichtlich.

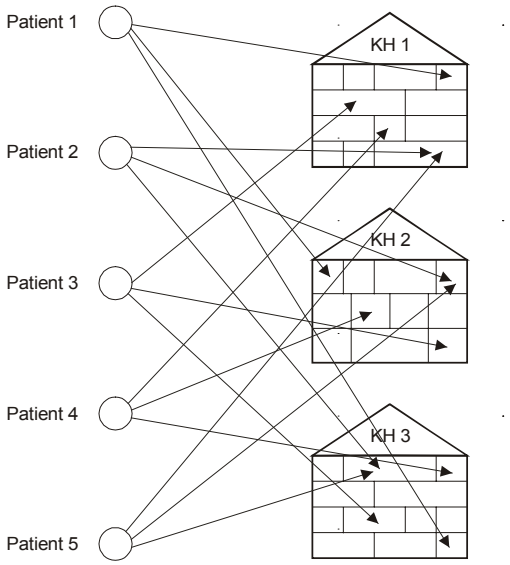

Abbildung 4.4: Darstellung des Problems der Zuordnung von Patienten zu geeigneten Stationen in Krankenhäusern als Graph

Die Kosten der einzelnen Zuordnungen eines Patienten zu einem Krankenhaus bzw. zu der bestgeeigneten Station eines Krankenhauses lassen sich, wie in Kapitel 4.1.3 beim Beschreiben des Optimierens nach einem Ziel dargestellt worden ist, mit der Formel 4.1 berechnen. Aus der Datenlage, die dem Planspiel INVENT zu Grunde liegt, ergibt sich die Beschränkung, dass jeder Patienten in der Reihenfolge seiner Aufnahme zu einer Station, die für die Behandlung der jeweiligen Krankheit geeignet ist, zugewiesen werden muss. Des Weiteren ist zu beachten, dass die Auslastung aller Stationen sowohl beim Einliefern eines Patienten als auch beim Entlassen eines Patienten aktualisiert werden muss. Eine gültige Sequenz, die in späterer Folge einer Tour einer Ameise entspricht, besteht daher aus einer der Reihenfolge entsprechenden Zuordnung jedes Patienten zu einem Krankenhaus, in dem zwingend eine geeignete Station vorhanden sein muss. Die Gesamtkosten dieser Zuordnung errechnet sich aus der Summe der Kosten aller Patienten. Gesucht wird nun jene gültige Sequenz, die die geringsten Gesamtkosten aufweist.

4.3.2 Umsetzen des Graphen in Quellcode

Der Quellcode, der für die Implementierung des Algorithmus’ nötig ist, wird in Abbildung 4.5 in Form von Pseudo-Code dargestellt. Der zuvor beschriebene Graph stellt dabei die Grundlage dar. Im Folgenden soll der Algorithmus beschrieben und danach die Unterschiede zu dem in Kapitel 3.3.2 dargestellten ACS-Algorithmus für die Futtersuche deutlich gemacht werden.

Beschreibung des Algorithmus’

Der Algorithmus beginnt, wie die beiden zuvor beschriebenen Algorithmen auch, mit der Initialisierung der Pheromonmengen aller Kanten mit τ_0. Des Weiteren wird L^+ auf -1 gesetzt, um zu signalisieren, dass noch keine Tour berechnet wurde.

In der Hauptschleife werden entweder t_{max} Iterationen durchgeführt oder die Schleife wird abgebrochen, wenn die zuvor festgelegte Berechnungszeit überschritten ist. In jedem Schleifendurchgang bilden k Ameisen eine Tour. Fängt eine Ameise an, eine Tour zu bilden, so wird zuerst die Länge L ihrer Tour auf 0 gesetzt und alle Einträge aus der Variable T, die die Tour der Ameise speichert, gelöscht. Danach erfolgt die Initialisierung der Auslastungen aller Stationen. Da sich die Auslastungen während einer Tour verändern, bei der nächsten Tour allerdings wieder von den Anfangsauslastungen ausgegangen werden muss, werden sowohl die Anfangsauslastungen als auch die momentanen Auslastungen gespeichert. Vor jedem Tourbeginn muss daher die momentane Auslastung jeder Station auf die Anfangsauslastung der jeweiligen Station zurückgesetzt werden. Ähnlich verhält es sich mit

der Liste des momentanen Patientenbestandes. Diese enthält alle Patienten, die momentan in einem der Krankenhäuser liegen, aufsteigend sortiert nach deren Entlassungstermin. Wird ein weiterer Patient eingeliefert oder ein Patient entlassen, so muss die Liste aktualisiert werden. Durch diese Veränderung während einer Tour muss die Liste am Anfang jeder Tour, ähnlich wie die Auslastung, mit jenen Patienten, die in der Liste des Anfangspatientenbestandes enthalten sind, initialisiert werden.

Anschließend wird die Liste aller Patienten, die eingeliefert werden sollen, durchgegangen. Bevor ein Patient eingeliefert wird, wird überprüft, ob der erste Patient, der in der Patientenbestandsliste eingetragen ist, vor dem Patienten, der gerade eingeliefert werden soll, entlassen worden ist. Da die Liste nach Entlassungsterminen sortiert ist, ist der Patient an erster Stelle immer jener, der vor allen anderen Patienten in der Liste entlassen wird. Ist der Patient vor dem momentan einzuliefernden Patienten entlassen worden, so wird er aus der Patientenbestandsliste gelöscht, wodurch ein anderer Patient an die erste Stelle vorrückt. Außerdem muss die Auslastung jener Station, auf der der Patient gelegen ist, aktualisiert werden. Danach wird erneut geprüft, ob der nun an erster Stelle stehende Patient vor dem momentan einzuliefernden Patienten entlassen wurde. Dieser Vorgang wird solange wiederholt, bis der erste Patient in der Liste einen Entlassungstermin nach der Einlieferung des momentan bearbeiteten Patienten aufweist.

Als nächstes erfolgt die Auswahl des Krankenhauses für den einzuliefernden Patienten. Um zu entscheiden, ob die Ameise jenes Krankenhaus wählt, welches sich für diesen Patienten als das beste hinsichtlich Kosten und momentaner Pheromonspur erweist, oder das nächste Krankenhaus mittels der Übergangsregel bestimmt, wird mittels eines Zufallsgenerators eine Zahl q erzeugt. Ist q kleiner oder gleich dem bereits zuvor festgelegten Parameter q_0, so wird die erste Vorgehensweise durchgeführt, ist q hingegen größer als q_0, so wird das Krankenhaus nach der zweiten Vorgehensweise bestimmt. Die Liste J_i^k existiert bei der Implementierung des vorgegebenen Problems nicht. Stattdessen wird die Funktion $Eignung$ (u, i) aufgerufen, die die Eignung der bestgeeigneten Station im Krankenhaus u für die Krankheit des Patienten i liefert. Ist diese gleich Null, so darf der Patient nicht in das Krankenhaus u eingeliefert werden. Wie bereits bei der Problembeschreibung erwähnt, ist die Auslastung der Station hingegen kein Kriterium, nach dem ein Patient nicht in ein Krankenhaus eingeliefert werden darf. Durch eine hohe Auslastung einer Station wird die Wahrscheinlichkeit, dass der Patient in diese Station eingeliefert wird, lediglich verkleinert. Der Wert η_{ij} ergibt sich, ähnlich wie beim Problem der Futtersuche, aus dem inversen Wert der Kosten.

Ist ein Krankenhaus für den momentan einzuliefernden Patienten gefunden, so wird dieses in die Tour T eingetragen und die Kosten, die die Einlieferung des Patienten laut der Formel 4.1 verursacht, werden zu den Gesamtkosten addiert. Da der Patient in der für ihn bestgeeigneten Station aufgenommen wird, muss dort auch die Auslastung entsprechend erhöht werden. Außerdem muss der Patient entsprechend seines Entlassungstermins in die Liste des Patientenbestandes eingefügt werden. Um die nachfolgenden Ameisen zur Erforschung neuer Wege zu animieren, wird die Menge an Pheromonen der soeben genutzten Kante (i, j) durch das Anwenden der lokalen Pheromon-Update-Regel verringert.

Hat eine Ameise ihre Tour beendet bzw. hat sie alle Patienten einem Krankenhaus zugeordnet, so werden die Gesamtkosten L mit den bisher geringsten Gesamtkosten L^+ verglichen. Ist die gefundene Lösung besser als die bisherigen Lösungen oder hat es zuvor noch keine Lösungen gegeben, was zutrifft, wenn L^+ dem Wert -1 entspricht, so wird die gefundene Tour T als bisher beste Tour T^+ und deren Gesamtkosten L als bisher geringste Gesamtkosten L^+ gespeichert.

Haben alle Ameisen ihre Tour beendet, dann darf jene Ameise die Kanten ihrer Tour verstärken, die die bisher beste Tour T^+ gefunden hat. Die Verstärkung erfolgt durch das Anwenden der Pheromon-Update-Regel, die nicht mit der zuvor schon verwendeten lokalen Pheromon-Update-Regel verwechseln ist, auf alle Kanten der besten Tour.

Sind t_{max} Iterationen berechnet oder ist die zuvor festgelegte Berechnungszeit überschritten, so wird der Algorithmus beendet. Da, wie schon in Kapitel 2.2.3 erwähnt wurde, der Algorithmus für jedes Monat aufgerufen wird, müssen danach noch sowohl die Auslastungen aller Stationen als auch die Liste des Patientenbestandes nach dem Zuordnen der Patienten entsprechend der besten Tour gespeichert werden. Da diese Informationen auf Grund des hohen Speicherplatzbedarfs und auch auf Grund des Zeitaufwandes, der beim ständigen Aktualisieren entstehen würde, nicht während des Algorithmus’ gespeichert wurden, muss die beste Tour nochmals rekonstruiert werden. Die Endauslastung des momentan bearbeiteten Monats stellt im nächsten Monat die Anfangsauslastung dar, analog verhält es sich mit der Liste des Patientenbestandes. Sind diese Informationen global gespeichert, so wird die beste Tour an die aufrufende Funktion zurückgegeben.

Wie in den anderen Darstellungen von Algorithmen ist auch hier aus den gleichen Gründen wieder auf den Zusatz der Iteration t verzichtet worden. Des Weiteren wurde auch der Zusatz der Ameise k bei der Bezeichnung der Tour T und deren Länge L vernachlässigt. Dies ist

zulässig, da jede Ameise schon nach dem Beenden ihrer Tour überprüft, ob die gefundene Tour bessere Werte erzielt als die bisher beste Tour. Ist dies der Fall, so wird die gefundene Tour als bisher beste Tour T^+ gespeichert und deren Länge als L^+. Andernfalls wird die gefundene Tour und deren Länge im Algorithmus nicht mehr verwendet und kann mit den Ergebnissen der nächsten Ameise überschrieben werden. Beim AS-Algorithmus wäre diese Vorgehensweise nicht zulässig, da dort die Touren der Ameisen zum Berechnen der Pheromon-Update-Regel herangezogen werden. Bei dem in Kapitel 3.3.2 beschriebenen ACS-Algorithmus könnte diese Veränderung auch durchgeführt werden, jedoch erschien sie dort nicht als sinnvoll, da das Sparen von Speicherplatz dort nicht im Vordergrund stand.

```
/* Initialisierung */
For each Kante (i, j)
    τij = τ0
End For
L+ = -1              //es gibt noch keine minimale Tourlänge

/* Hauptschleife */
t = 1
While t <= tmax und zuvor festgelegte Berechnungszeit nicht überschritten
    /* Für jede Ameise eine Tour finden */
    For k = 1 to m
        L = 0
        Lösche alle Einträge aus T

        For each Station
            Auslastung (Station) = Anfangsauslastung (Station)
        End For

        Lösche alle Einträge aus der Liste des Patientenbestandes
        For each Patient in Liste des Anfangspatientenbestandes
            Füge den Patient in die Liste des Patientenbestandes ein
        End For

        For i = erster Patient in der Liste der einzuliefernden Patienten to letzter
        Patient in der Liste der einzuliefernden Patienten
            While erster Patient in der Liste des Patientenbestandes vor dem
            Einliefern von Patient i entlassen wird
                Lösche den ersten Patienten aus der Liste des
                Patientenbestandes
                Aktualisiere die Auslastung der jeweiligen Station
            End While

            Errechne eine Zufallszahl q

            Wähle das Krankenhaus j mittels der Formel:
```

$$j = \begin{cases} \arg\max_{Eignung(u,i)\,\text{not}=0} \left\{ [\tau_{iu}] \cdot [\eta_{iu}]^{\beta} \right\} & \text{wenn } q \le q_0 \\ \Psi & \text{wenn } q > q_0 \end{cases}$$

wobei $Eignung(\Psi, i)$ *not* = 0 und Ψ mittels der Wahrscheinlichkeit

$$p_{i\Psi} = \frac{[\tau_{i\Psi}] \cdot [\eta_{i\Psi}]^{\beta}}{\sum_{l \in J_i} ([\tau_{il}] \cdot [\eta_{il}]^{\beta})}$$

gewählt wird.

//Patient i wird in die für seine Krankheit bestgeeignetste Station in //Krankenhaus j eingeliefert

Füge Krankenhaus j in T ein
$L = L + 1/\eta_{ij}$

Aktualisiere die Auslastung der jeweiligen Station

Füge den Patienten i in die Liste des Patientenbestandes, die nach dem Entlassungsdatum sortiert ist, an der richtigen Position ein

Berechne die neue Menge an Pheromonen laut der lokalen Pheromon-Update-Regel:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \tau_0$$

End For
//Ameise hat ihre Tour beendet

If $L < L^+$ oder $L^+ = -1$ **Then**
Aktualisiere T^+ und L^+
End If

End For
//Alle Ameisen sind durchgelaufen

/* Pheromonspur updaten */
For each Kante (i, j) $\in T^+$
Berechne die neue Menge an Pheromonen laut der Pheromon-Update-Regel:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij},$$

wobei $\Delta\tau_{ij} = 1/L^+$
End For
End While

Konstruiere die Zuordnung der Patienten laut T^+ nochmals und speichere nach dem letzten eingelieferten Patienten die Auslastung und die Liste des Patientenbestandes für den nächsten Aufruf des Algorithmus’

Return T^+

Abbildung 4.5: Darstellung des ACS-Algorithmus für das Steuern der Patienten im Planspiel INVENT

Unterschiede zum ACS-Algorithmus bei der Futtersuche

Die im Zwischenfazit getätigte Aussage, dass für das Adaptieren des Algorithmus’ nach der Darstellung des Problems als Graphen lediglich die definierten Beschränkungen im Algorithmus implementiert werden müssten, bestätigt sich hier. Es existieren drei Unterschiede zwischen den Beschränkungen für die Futtersuche und jenen des vorliegenden Problems. Die erste ist, dass auf die Auslastung besonders Rücksicht genommen werden muss, da sich diese während des Auffindens einer Tour ständig verändert. Des Weiteren verändert sich mit der Auslastung auch der Patientenbestand. Hierbei ist es wichtig, die Patienten sortiert nach ihrem Entlassungstermin zu speichern und rechtzeitig jene aus der Liste des Patientenbestandes zu entfernen, die aus der Station entlassen wurden. Die dritte unterschiedliche Beschränkung bezieht sich auf die Eignung, die als einziges Kriterium herangezogen wird, um zu bestimmen, ob ein Patient in ein Krankenhaus eingewiesen werden darf oder nicht. Diese ist vergleichbar mit der Liste J_i^k, welche sich aber im Gegensatz zur Eignung während des Auffindens der Tour verändert.

Abgesehen von diesen drei Änderungen entspricht der hier behandelte Algorithmus im Wesentlichen dem Algorithmus der Futtersuche. Ein weiterer Unterschied zwischen dem hier dargestellten Algorithmus und dem ACS-Algorithmus für die Futtersuche ist darauf zurückzuführen, dass im letzteren Kandidatenlisten verwendet wurden. Diese stellen jedoch eine Erweiterung des Algorithmus’ der Futtersuche dar, die im hier behandelten Algorithmus aus zuvor schon beschriebenen Gründen nicht implementiert werden sollten. Um Speicherplatz zu sparen, wurde hingegen im hier beschriebenen Algorithmus auf das Speichern der Touren aller Ameisen und deren Längen verzichtet. Dadurch musste der Vergleich, ob die gefundene Tour besser ist als die bisher beste, sofort, nachdem eine Ameise ihre Tour beendet hat, getätigt werden. Ein weitere Unterschied ist entstanden, da beim hier behandelten Algorithmus schon bekannt ist, wie er in das gesamte Programm eingebunden werden soll. Deshalb werden bestimmte Daten, wie die Endauslastung der besten Tour und der Endbestand an Patienten in den Krankenhäusern nach der Zuordnung entsprechend der besten Tour, global gespeichert. Anstatt die beste Tour und deren Länge auszugeben, wie das bei der Futtersuche gemacht wird, wird die beste Tour an die übergeordnete Funktion übergeben. Ein letzter Unterschied bezieht sich auf den Abbruchzeitpunkt, der bei dem Algorithmus für die Futtersuche nur durch die Anzahl an Iterationen festgelegt ist. Bei dem hier beschriebenen Algorithmus wird allerdings abgebrochen, wenn entweder die Anzahl an festgelegten Iterationen oder die festgelegt Berechnungszeit überschritten ist.

Diese Veränderungen beruhen alle auf möglichen Erweiterungen bzw. Verbesserungen oder auf der Tatsache, das der hier behandelte Algorithmus schon in ein Programm eingebettet ist. Würde man diese Veränderungen weglassen und lediglich die unterschiedlichen Beschränkungen implementieren, so würde der Algorithmus auch funktionieren, was die am Anfang beschriebene Aussage bestätigt.

4.3.3 Programmtechnische Details

In diesem Kapitel soll auf die Setzung der Parameter und auf das Speichern von Daten und der damit verbundenen Probleme eingegangen werden.

Parametersetzung

Beim Setzen der Parameter wurde sowohl auf die Literatur[79], in der die Implementierung eines ACS-Algorithmus' beschrieben wurde, als auch auf Erfahrungswerte zurückgegriffen. Dabei hat sich die Anzahl von 10 Ameisen in der Literatur einheitlich durchgesetzt und soll auch hier entsprechend übernommen werden. Für die Herleitung dieser Anzahl soll hier auf Dorigo und Gambardella[80] verwiesen werden. Der Parameter β, der in der Übergangsregel die Wertigkeit der Pheromonspur darstellt, wird in der Literatur entweder mit dem Wert 1, entsprechend der Wertigkeit der Kosten, oder mit dem Wert 2 besetzt. An Hand einiger Tests hat sich herausgestellt, dass der Algorithmus für das vorliegende Problem für den Wert 2 durchschnittlich bessere Ergebnisse liefert. Deshalb wurde entschieden, den Parameter β mit dem Wert 2 zu belegen. Bei dem Verdampfungsfaktor ρ und dem Parameter q_0, der, wie zuvor beschrieben, für die Entscheidung verantwortlich ist, nach welcher Strategie der nächste Knoten bzw. das nächste Krankenhaus ausgewählt wird, ist sich die Literatur ebenfalls einig. Der Verdampfungsfaktor ρ wird auf den Wert 0,1 gesetzt, was bedeutet, dass 10 % der Pheromone pro Iteration verdampfen. Für den Parameter q_0 hat sich der Wert 0,9 als gut geeignet erwiesen. Dies bedeutet, dass nur 10 % aller Ameisen das Krankenhaus auf Grund der Übergangsregel bestimmen, die restlichen 90 % wählen jenes Krankenhaus, das hinsichtlich der Kosten und der bisher gelegten Pheromonmengen am besten geeignet erscheint. Auch für die Anfangsmenge an Pheromonen τ_0 auf jeder Kante haben sich in der

[79] Vgl. Dorigo, M. und Gambardella, L. M., *IEEE Transactions on Evolutionary Computation* 1 (1997a) S. 59 ff, Dorigo, M. und Gambardella, L. M., *BioSystems* 43 (1997b) S. 77, Randell, M. und Tonkes, E. (2001) S. 8, Gambardella, L. M. / Taillard, E. / Agazzi, G. (1999) S. 74 und Bonabeau, E. / Dorigo, M. / Theraulaz, G. (1999) S. 51.
[80] Vgl. Dorigo, M. und Gambardella, L. M., *IEEE Transactions on Evolutionary Computation* 1 (1997a) S. 58.

Literatur schon Werte bzw. in diesem Fall eine Formel durchgesetzt. Diese lautet:

$$\tau_0 = \frac{1}{n \cdot L_{nn}} \quad , \tag{4.2}$$

wobei n die Anzahl der Patienten darstellt und L_{nn} der Länge jener Tour entspricht, die zuvor mittels der von Dorigo und Gambardella[81] ebenfalls verwendeten „Nearest Neighbour“-Heuristik[82] errechnet werden muss. Bei diesem Algorithmus wird immer jenes Krankenhaus gewählt, welches sich hinsichtlich der Kosten als bestes erweist.

Auf das Festlegen der Abbruchbedingungen soll an dieser Stelle nicht näher eingegangen werden, da diese von den Ergebnissen des Algorithmus’ abhängen und daher in dem Kapitel „Ergebnisse“ behandelt werden. Jedoch kann bereits vorweggenommen werden, dass sich laut Literatur Werte von 100 bis ungefähr 10.000 Iterationen als geeignet erwiesen haben.

Effizienz der Speicherung von Daten

Bei Optimierungsalgorithmen oder allgemein bei Algorithmen, die schnell durchgeführt werden müssen, ist es besonders wichtig, dass der Algorithmus effizient programmiert ist. Dabei spielt das Speichern von Daten eine wesentliche Rolle. Wird ein Wert öfters verwendet, so sollte er lokal gespeichert werden. Bei großen Datenmengen entsteht dadurch oft ein Konflikt, da einerseits auf lokal gespeicherte Daten schneller zugegriffen werden kann, andererseits der Hauptspeicher nicht unnötig belegt werden soll. Beispielsweise sollte nicht die gesamte Datenbank in einem Programm gespeichert werden sondern immer nur jene Tabellen, die gerade verwendet werden.

Gerade bei großdimensionierten Problemen, in denen viele Daten gespeichert werden müssen, kann durch die richtige Wahl der Datenstruktur und durch das effiziente Verwenden dieser viel Rechenzeit gespart werden. Dies soll an Hand der Liste des Patientenbestandes gezeigt werden. Da die Menge an Patienten, die momentan in den Krankenhäusern liegen, ständig schwankt, aber vor allem, weil bei Einlieferungen Patienten an einer bestimmten Position in der Liste, entsprechend dem Einlieferungszeitpunktes, eingefügt werden müssen, eignet sich eine statische Datenstruktur, deren Größe schon vor der Laufzeit angegeben werden muss und deren Felder nicht verschoben werden können, nicht. Stattdessen sollen die Patienten in einer „Liste“ gespeichert werden. Eine Liste stellt im programmiertechnischen Kontext eine dynamische Datenstruktur dar, die aus mehreren Elementen besteht, wobei in diesem Fall jedes Element die Daten eines Patienten speichert.

[81] Vgl. Dorigo, M. und Gambardella, L. M., *BioSystems* 43 (1997b) S. 77.
[82] Vgl. Rosenkrantz, D. J. / Stearns, R. E. / Lewis, P. M., *SIAM Journal on Computing* 6 (1977).

Des Weiteren wird in einem Listenelement vermerkt, welches Element das nächste ist bzw. ob es sich bei dem Element um das letzte Element der Liste handelt. Dynamisch bedeutet im Gegensatz zu statisch, dass jederzeit Listenelemente eingefügt, verschoben und gelöscht werden können.

Eine mögliche Methode für die Implementierung der Patientenbestandsliste ist, jedes Mal, wenn ein Patient eingeliefert wird, ein Element hinzuzufügen und jedes Mal, wenn ein Patient entlassen wird, dessen Element wieder zu löschen. Außerdem müssen nach dem Beenden einer Tour alle Patienten aus der Liste entfernt werden und die Liste mit den Patienten des Anfangspatientenbestandes initialisiert werden. Das Problem bei Listen ist jedoch, dass das Hinzufügen und Löschen von Listenelementen relativ viel Zeit kostet. Daher ist es in diesem Fall sinnvoller, beim ersten Zuordnen aller einzuliefernden Patienten jeweils ein Listenelement für jeden Patienten in die Liste einzufügen, aber beim Entlassen eines Patienten, das Element nicht zu löschen, sondern an die letzte Stelle der Liste zu verschieben und es als momentan nicht verwendetes Element zu markieren. Steht ein nicht verwendetes Listenelement an letzter Stelle, so kann dieses beim Einliefern eines neuen Patienten verwendet werden, anstatt ein neues Listenelement einzufügen. Am Ende dieser Zuordnung besteht die Liste dann aus der maximalen Anzahl an Elementen, die für die Zuordnung der einzuliefernden Patienten benötigt werden. Daher sollen im weiteren Verlauf keine Elemente mehr hinzugefügt oder gelöscht werden, sondern anstatt des Hinzufügens soll ein nicht verwendetes Element, das an der letzten Stelle der Liste steht, mit den Daten gefüllt und an die richtige Stelle verschoben werden, und anstatt des Löschens soll das Element an die letzte Stelle der Liste verschoben und als nicht verwendetes Element markiert werden.

Des Weiteren ist gerade bei solchen Listen wichtig, dass nur jene Informationen in einem Element enthalten sind, die auch wirklich verwendet werden. Da es auch Listenelemente im Programm gibt, in denen die Daten eines einzuliefernden Patienten gespeichert werden, muss für die Patienten in der Patientenbestandsliste ein anderes Listenelement definiert werden. Dies ist besonders wichtig, da ein Element eines einzuliefernden Patienten den zehnfachen Speicherplatz besetzt wie ein Element eines Patienten der Patientenbestandsliste. Dies beruht darauf, dass bei dem einzuliefernden Patienten alle ihn betreffenden Daten beispielsweise seine Diagnose, sein Aufnahme- und Entlassungsdatum, die Entfernungen, die er zu jedem Krankenhäusern zurücklegen muss, und auch die Station und das Krankenhaus, in das er schließlich eingewiesen wird, gespeichert werden müssen. Beim Patientenbestand geht es nicht mehr darum, den Patienten genau identifizieren zu

können, sondern es reicht aus, zu wissen, wann ein Patient welche Station in welchem Krankenhaus verlässt. Deshalb müssen auch nur diese Daten gespeichert werden.

Beachtet man diese beiden Maßnahmen, so kann die Berechnungszeit rapide gesenkt werden. Aus selbst durchgeführten Versuchen geht hervor, dass bei einer großen Menge von einzuliefernden Patienten die Berechnungszeit um bis zu 70 % gesenkt werden kann.

Weitere 10 % können eingespart werden, wenn man als Datenstruktur statt einer herkömmlichen Liste eine Ringliste verwendet. Diese Datenstruktur entspricht einer Liste, in deren letztem Element vermerkt ist, dass das ersten Element der Liste als nächstes Element gilt. Durch diesen Eintrag schließt sich so zu sagen die Kette der Elemente und bildet einen Ring. Will man ein Element einfügen, so existieren bei der herkömmlichen Liste zwei Ausnahmefälle. Diese treten ein, wenn das einzufügende Element entweder an erster oder an letzter Stelle eingefügt werden soll, und beruhen darauf, dass im einen Fall kein Element vor dem einzufügenden Element existiert und im anderen keines danach. Bei der Ringliste hingegen ist immer ein Element davor und ein Element danach vorhanden, da die Kette der Elemente geschlossen ist. Die auftretenden Fälle alleine würden keine so große Zeitersparnis bringen, jedoch kann dadurch, dass alle Elemente gleich behandelt werden können, auf die Abfragen, ob es sich um einen Ausnahmefall handelt, verzichtet werden. Da wie zuvor beschrieben die Liste nicht nur gerade in Verwendung befindende Elemente enthält, sondern auch jene Elemente speichert, die nicht mehr benötigt werden, ist das Umwandeln dieser Liste in eine Ringliste etwas komplexer, und es müssen doch Unterscheidungen zwischen speziellen Fällen getroffen werden, jedoch zeigt sich auf Grund der Zeitersparnis, dass eine Ringliste für das hier vorliegende Problem trotzdem besser geeignet ist.

4.3.4 Ergebnisse

Die Qualität der Lösung, die mittels des implementierten Algorithmus gefunden wird, hängt von der Anzahl der durchlaufenen Iterationen und der Anzahl der Patienten, die einem Krankenhaus zugeteilt werden sollen, ab. Des Weiteren wird das Ergebnis von den jeweiligen Patienten, die eingewiesen werden sollen, beeinflusst. Dies bedeutet, dass das Ergebnis leicht schwankt, wenn zwar die Anzahl der Patienten gleich bleibt, jedoch andere Patienten mit anderen Diagnose und anderen Verweildauern den Krankenhäusern zugeordnet werden. Auch das Monat bzw. die Periode, in dem der Algorithmus angewendet wird, beeinflusst die Qualität der Lösung, da die Anfangsauslastungen der Stationen am Beginn jedes Monats unterschiedlich sind.

Im Folgenden sollen gezeigt werden, in welchem Maß sich die Qualität der Lösung verändert, wenn die Anzahl der zuzuordnenden Patienten schwankt. Dies soll an Hand von drei Testtabellen demonstriert werden. Diese Tabellen unterscheiden sich hinsichtlich der Anzahl und Zusammensetzung der Patienten, und werden jeweils aus den Realdaten der Patienten aller Krankenhäuser eines Monats, entsprechend einer Anzahl von ungefähr 15.000 Patienten, errechnet.

Eine Testtabelle besteht aus einer gewissen Menge von Patienten, die mittels des Algorithmus' in eines der Krankenhäuser eingewiesen werden sollen. Sie stellen also jene Menge an Patienten dar, die bereits der IV angehören.[83] Für die Entscheidung, welche Patienten zum Zeitpunkt des Testens zu der IV gewechselt sind, wurde hier beispielhaft eine Unterteilung von zwei Kategorien vorgenommen. In die erste Kategorie fallen Patienten, die zwischen 25 und 45 Jahren alt sind, die restlichen Patienten werden der zweiten Kategorie zugeteilt. Diese Unterscheidung beruht auf der Tatsache, dass Versicherungsnehmer der ersten Kategorie in der Realität von Integrierten Versorgungsformen besonders angesprochen werden und daher mit höherer Wahrscheinlichkeit zu einer IV wechseln. Besonders ältere Patienten weisen einerseits eine geringe Wechselbereitschaft auf und gehören, erste Erfahrungen aus der Schweiz belegen dies, andererseits auch nicht zu den bevorzugten Kunden der IV.

Die erste Testtabelle enthält Daten, wie sie eher in der Anfangsphase des Planspiels zu erwarten sind, wobei diese Daten aus 10 % der ersten Kategorie von Patienten und 1 % von Patienten der zweiten Kategorie gewählt wurden. Dies entspricht ungefähr 3 % aller Patienten, die in die Krankenhäuser eingeliefert wurden. Die zweite Tabelle setzt sich aus 25 % der ersten und 2,5 % der zweiten Kategorie von Patienten zusammen und stellt eine fortgeschrittene Phase des Etablierungsprozesses einer IV dar, in der schon 7 % aller Patienten der Krankenhäuser IV-Patienten sind. Um die maximale Zahl an Patienten, die im Laufe des Planspiels zu einer IV wechseln, festzulegen, erscheint eine Patientenzahl von 50 % der ersten und 5 % der zweiten Altersgruppe als sinnvoll.[84] Dies entspricht einem Anteil von 15 % aller Patienten, die in die Krankenhäuser eingeliefert wurden. Hierbei darf nicht vergessen werden, dass das Planspiel nicht über die gesamte mögliche Lebensdauer einer Integrierten Versorgungsform gespielt wird, sondern nur deren Etablierung aufgezeigt werden soll.

[83] Die Beschreibung der Ergebnisse bezieht sich auf die Einspielervariante, ähnlich kann jedoch auch für die Zweispielervariante vorgegangen werden.
[84] Vgl. Baur, R./ Stock, J. (2002).

Der Algorithmus wurde auf alle drei Testtabellen angewendet, wobei die beste Lösung nach jeweils 10 Iterationen gespeichert wurde. Insgesamt wurden 5.000 Iterationen berechnet, wobei sich die Lösung bei der ersten Tabelle schon nach 400 Iterationen nicht mehr veränderte. In der zweiten Tabelle stellte sich erst nach 1.500 Iterationen eine konstant bleibende Lösung ein. Für die dritte Tabelle reichte eine Berechnung von 5.000 Iterationen nicht aus, um eine konstante Lösung zu erhalten, daher wurden 40.000 Iterationen berechnet. Die daraus resultierende Lösung scheint konstant zu sein und ist lediglich um 0,001 % besser als jene nach 5.000 Iterationen.

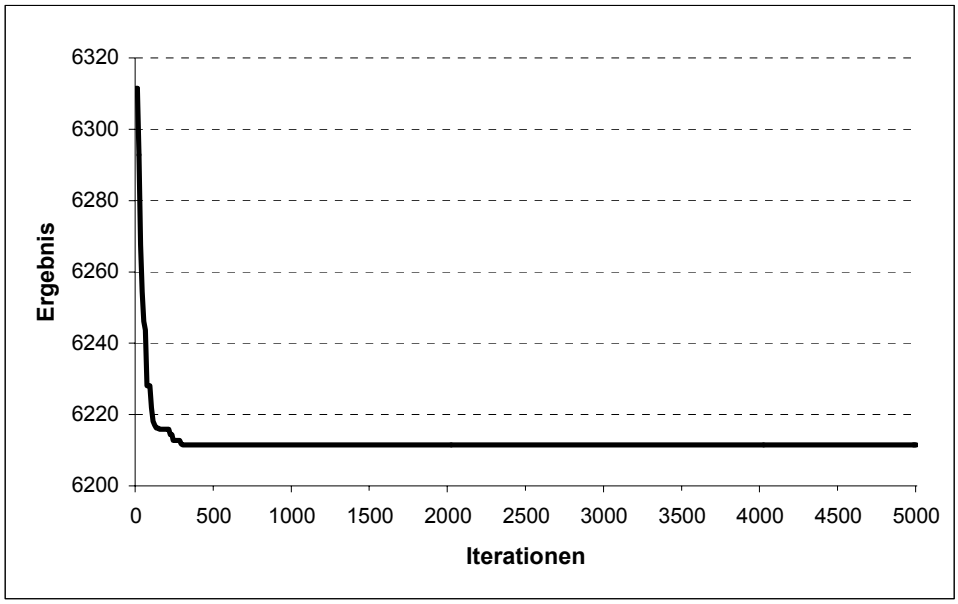

Abbildung 4.6: Darstellung der Qualitätsveränderung der Lösung über 5.000 Iterationen bei 3 % IV-Patienten

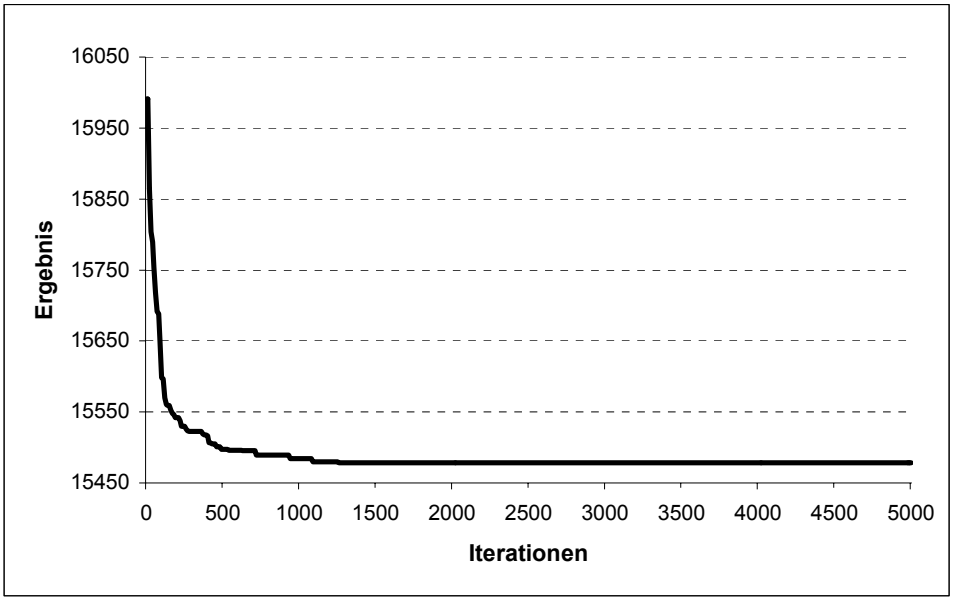

Abbildung 4.7: Darstellung der Qualitätsveränderung der Lösung über 5.000 Iterationen bei 7 % IV-Patienten

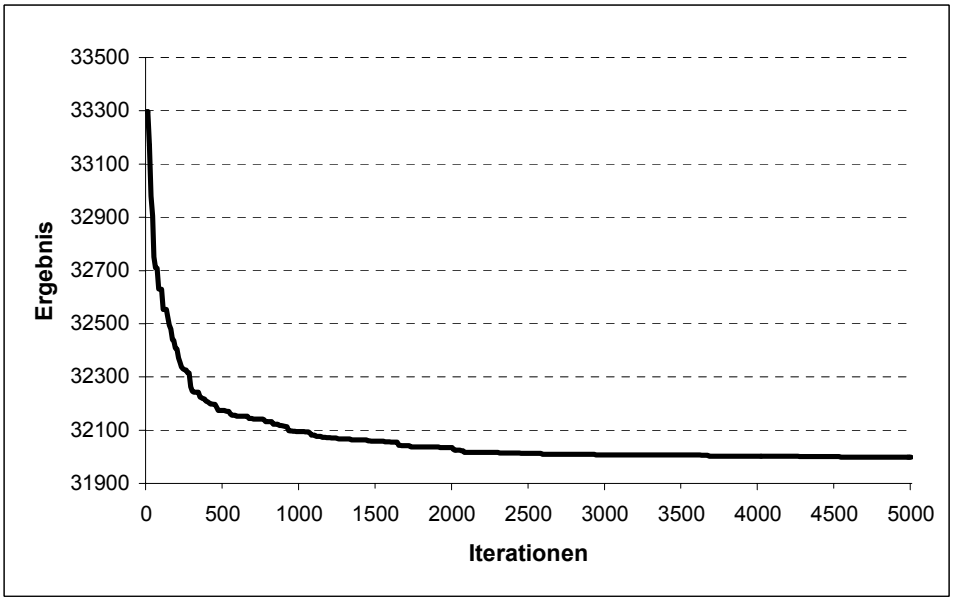

Abbildung 4.8: Darstellung der Qualitätsveränderung der Lösung über 5.000 Iterationen bei 15 % IV-Patienten

Um die Leistungskurven des Algorithmus’ genauer darstellen zu können, sollen diese nochmals mit einer geringeren Anzahl an Iterationen abgebildet werden.

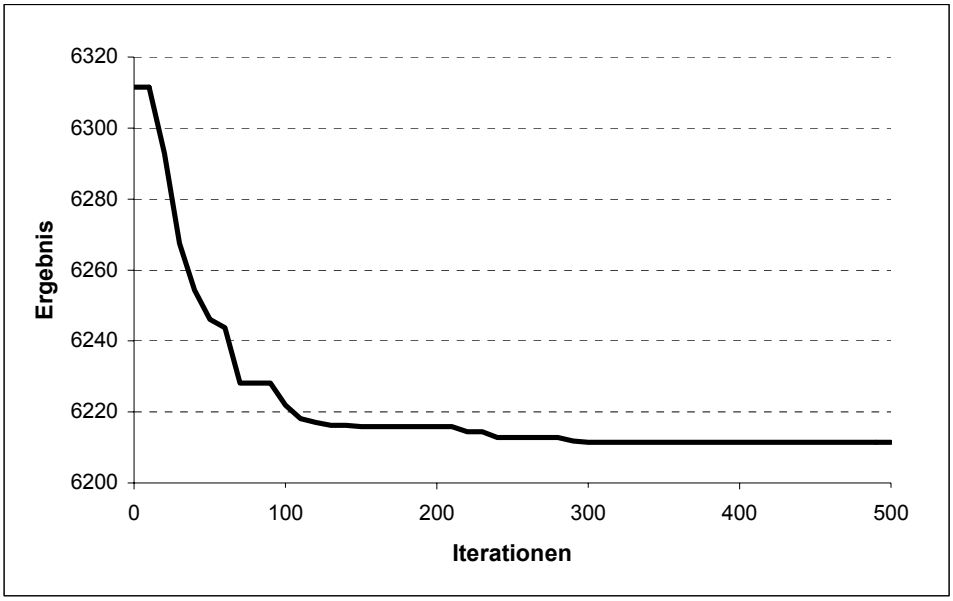

Abbildung 4.9: Darstellung der Qualitätsveränderung der Lösung über 500 Iterationen bei 3 % IV-Patienten

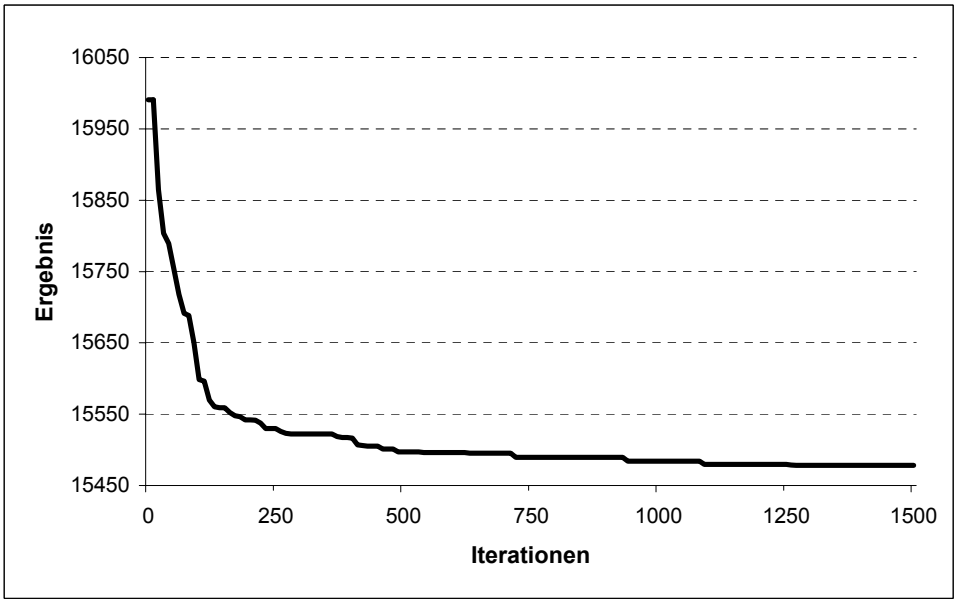

Abbildung 4.10: Darstellung der Qualitätsveränderung der Lösung über 1.500 Iterationen bei 7 % IV-Patienten

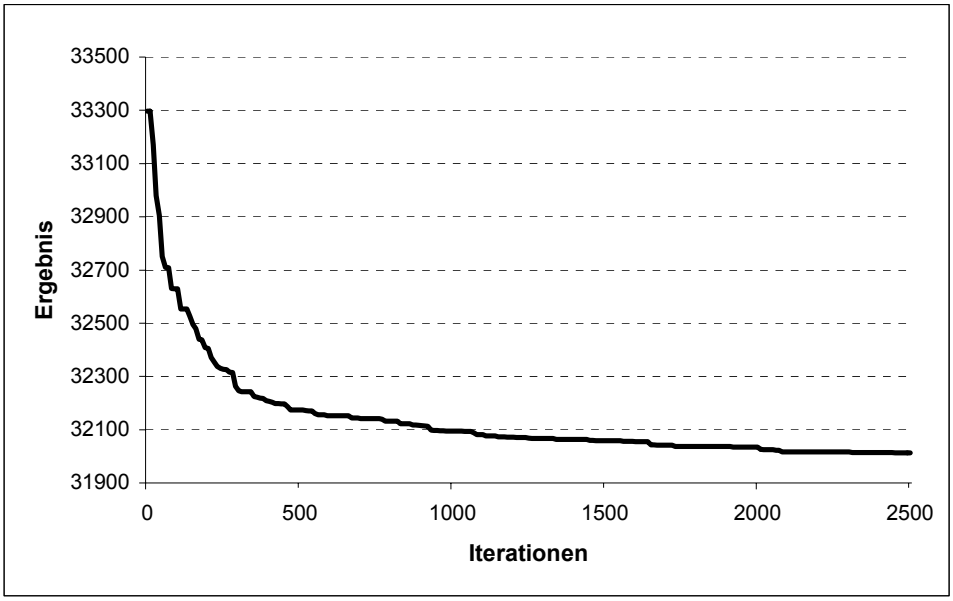

Abbildung 4.11: Darstellung der Qualitätsveränderung der Lösung über 2.500 Iterationen bei 15 % IV-Patienten

Auf Grund dieser Abbildungen zeigt sich, dass die Anzahl an Iterationen, mit der eine relativ gute Qualität der Lösung erzielt wird, von der Anzahl an Patienten abhängig ist. Scheint im letzten Fall eine Anzahl von 1.000 – 1.500 Iterationen als sinnvoll, so reichen im zweiten Fall 500 – 1.000 aus, um eine ähnliche Effizienz zu erreichen. Im ersten Fall erscheint es auf Grund des schnellen Auffindens der besten Lösung sinnvoll, bis zu dieser besten Lösung zu rechnen.

Da einerseits das Planspiel nicht nur für die momentan zu Grunde liegenden Daten konfiguriert ist, sondern auch mit anderen Daten gespielt werden kann, und andererseits die Dauer der Berechnung einer Periode vom Zweck des jeweiligen Spiels abhängig ist, soll die Spielleitung den Abbruchzeitpunkt bestimmen. Auf Grund der Tatsache, dass das Berechnen von Iterationen nicht auf jedem Rechner gleich lange dauert, sollen, wie zuvor bei der Beschreibung des Algorithmus’ schon hervorgegangen ist, zwei Abbruchbedingungen festgelegt werden. Eine bezieht sich dabei auf die maximale Anzahl an Iterationen, die berechnet werden sollen, die andere legt die maximale Berechnungsdauer fest. Ist eine dieser Grenzen überschritten, so wird der Algorithmus beendet. Die Angabe der Zeit bezieht sich dabei auf die maximale Zeit der Berechnung einer Periode, die Angabe der Anzahl an Iterationen wird hingegen pro Patient festgelegt, da, wie aus den zuvor präsentierten Abbildungen hervorgeht, die Anzahl an benötigten Iterationen für eine gute Qualität der Lösung von der Anzahl an zuzuordnenden Patienten abhängt. Durch diese beiden Grenzen muss die Performance des verwendeten Rechners nicht berücksichtigt werden, da bei

Spielen, in denen die Teilnehmer nur maximal eine bestimmte Zeit lang warten sollen, diese Zeit angegeben werden kann, bei Spielen, in denen die Qualität der Lösung im Vordergrund steht, die Anzahl der Iterationen festgelegt werden kann.

Abschließend sollen noch Richtwerte für die Zeit der Berechnung einer Lösung mit relativ guter Qualität angegeben werden. Für die Berechnungen wurde ein Rechner mit Intel Pentium III Prozessor, 933 MHz und 256 MB RAM verwendet. Das Berechnen von 400 Iterationen auf Grund der Daten der ersten Testtabelle dauert mit dem zuvor beschriebenen Rechner 1 Minute und 13 Sekunden. Sollen für die Daten der zweiten Tabelle beispielsweise 750 Iterationen berechnet werden, so dauert diese 4 Minuten und 38 Sekunden. Die maximale Berechnungsdauer mit den Daten der dritten Tabelle und 1.250 Iterationen dauert ungefähr 15 Minuten.

5 Fazit

Die zentrale Aufgabe dieser Arbeit war es, einen geeigneten Ameisenalgorithmus für das im Planspiel INVENT vorliegende Problem der Zuordnung von Patienten zu Krankenhäusern zu ermitteln und diesen geeignet zu implementieren. Dieses Ziel kann als erreicht angesehen werden, da nunmehr ein lauffähiger Algorithmus zur Verfügung steht, der zudem auch noch in angemessener Zeit für die Problemstellung ausreichend gute Lösungen ermittelt.

Als besonders aufwendig hat sich in diesem Zusammenhang die Analyse der vorhandenen Ameisenalgorithmen erwiesen, da es mittlerweile eine recht große Zahl an Variationen von Ameisenalgorithmen gibt. Nicht zuletzt deshalb musste diesem Teil der Arbeit auch besonders viel Raum gegeben werden.

Dies hat sich letztlich jedoch als sinnvoll erwiesen, da die Berechnungen mit Hilfe solcher Algorithmen durchaus aufwendig sind. Wie auch schon in der Arbeit beschrieben, ist es nur zum Teil möglich, die in der Literatur verwendeten oder empfohlenen Parameterwerte 1:1 zu übernehmen. Vieles muss daher experimentell herausgefunden werden. Das versuchsweise Implementieren von ungeeigneten Algorithmen hätte daher unangemessen viel Zeit in Anspruch genommen, ohne jedoch für das vorgegebene Problem zu zufriedenstellenden Lösungen zu kommen.

Darüber hinaus kann jedoch die Art und Weise, in der zunächst das Problem in seiner Struktur sehr genau analysiert und dann erst mit der Implementierung und ersten Berechnungen begonnen wurde, auch für andere Probleme übernommen werden.

Das Ergebnis selbst jedoch ist anders ausgefallen, als dies anfänglich erhofft wurde. Zwar zeigten die Ergebniswerte die bei Ameisen zu erwartende Konvergenz gegen eine als optimal vermutete Lösung, die erreichte Berechnungszeit für alle Patienten eines Monats liegt jedoch bei mehreren Stunden. Dies führte dazu, dass eine Differenzierung zwischen IV-Patienten und Patienten, die keiner IV angehören, im Algorithmus implementiert wurde. Dadurch liegt das Berechnen der IV-Patienten wieder im zeitlichen Rahmen und die Parameter für das Berechnen der Patienten, die keiner IV zugehören, können entsprechend der zur Verfügung stehenden Zeit gesetzt werden.

Besonders überrascht hat letztlich der Einfluss der konkreten Implementierung auf die Performance des Algorithmus‘. Dabei zeigt sich, dass nicht nur die Wahl des Algorithmus’

entscheidend ist, sondern auch die Art der Implementierung und damit unter anderem auch die Verwaltung der zu Grunde liegenden Daten. Wird der Algorithmus ineffizient implementiert, so kann trotz richtiger Wahl des Algorithmus' dieser eine schlechte Performance aufweisen und sich als ungeeignet für das zu Grunde liegende Problem herausstellen. Der in dieser Arbeit angesprochenen Anpassung des Grundalgorithmus‘ an die konkreten Erfordernisse des Problems kommt demnach eine sehr große Bedeutung zu.

Die Wahl des „optimalen“ Algorithmus für das vorliegende Problem musste sich außerdem, anders als das zumeist in der Literatur der Fall ist, nicht nur an der optimalen Lösung selbst, sondern auch an den äußeren Gegebenheiten des Planspiels INVENT orientieren. Es ist den Spielern beispielsweise nicht zuzumuten, stundenlang, was in diesem Fall wirklich mehrere Stunden bedeuten könnte, auf Ergebnisse zu warten. Im Gegensatz dazu ist es bei der Verwendung des Algorithmus für ein Planspiel nicht entscheidend, ob die Lösung um einen geringen Prozentsatz des Optimums besser oder schlechter ist. Ziel des Planspiels ist es, den Spielern Eindrücke zu vermitteln, wie sich die an einer Integrierten Versorgungsform beteiligten Versicherten und Leistungsanbieter verhalten könnten.

Abschließend kann daher noch einmal festgestellt werden, dass es „den“ besten Ameisenalgorithmus nicht wirklich gibt. Für das vorgegebene Problem im Planspiel INVENT ist jedoch im Rahmen dieser Arbeit eine praktikable Lösung gefunden worden.

Literaturverzeichnis

Aron, S. / Deneubourg, J.-L. / Goss, S. / Pasteels, J. M.: „Functional self-organisation illustrated by inter-nest traffic in the Argentine ant Iridomyrmex humilis". In Alt, W. und Hoffman G. (Hrsg.): Biological Motion, Springer-Verlag, Berlin (1990) 533-547.

Baur, R. und Stock, J.: „Neue Formen der Krankenversicherung in der Schweiz - zur Evaluation der ersten HMOs in Europa". In Preuß, K.-J. / Räbiger, J. / Sommer, J. H. (Hrsg): Managed Care - Evaluation und Performance-Measurement integrierter Versorgungsmodelle - Stand der Entwicklung in der EU, der Schweiz und den USA, Stuttgart, New York (2002)135-152.

Bonabeau, E. / Dorigo, M. / Theraulaz, G.: Swarm Intelligence: From Nature to Artificial Systems, Oxford University Press, New York, 1999.

Boyan, J. A. und Littman, M. L.: „Packet routing in dynamically changing networks: A reinforcement learning approach". In Cowan, J. D. / Teauso, G. / Alspector, J. (Hrsg): Advances in Neural Information Processing Systems, Band 6, Morgan-Kauffman, San Francisco (1994) 671-678.

Bullnheimer, B. / Hartl, R. F. / Strauss, C.: „Applying the Ant System to the vehicle routing problem", Paper presented at Second International Conference on Metaheuristics, Sophia-Antipolis, France, 1997. (Veröffentlicht in Voss, S. / Martello, S. / Osman, I. H. / Roucairol, C. (Hrsg): Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer, Boston (1999), 285-296.)

Bullnheimer, B. / Hartl, R. F. / Strauss, C.: „A new rank based version of the ant system - A computational study". Central European Journal for Operations Research and Economics, Band 7, Nr. 1 (1999) 25-38.

Colorni, A. / Dorigo, M. / Maniezzo, V.: „Distributed optimization by ant colonies". In Varela, F. und Bourgine, P.: Proceedings First European Conference on Artificial Life, MIT Press, Cambridge (1991) 134-142.

Deneubourg, J.-L. / Aron, S. / Goss, S. / Pasteels, J. M.: „The self-organizing exploratory pattern of the Argentine ant", Journal of Insect Behavior, Band 3 (1990) 159-168.

Di Caro, G. und Dorigo, M.: „AntNet: A mobil agents approach to adaptive routing", Technical Report IRIDA/97-12, Université Libre de Bruxelles, Belgium, 1997a.

Di Caro, G. und Dorigo, M.: „A study of distributed stigmergetic control for racket-switched communications networks", Technical Report IRIDIA/97-12, Université Libre de Bruxelles, Belgium, 1997b.

Di Caro, G. und Dorigo, M.: „Mobile agents for adaptive routing". In El-Rewini H. (Hrsg): Proceedings of the 31st Hawaii International Conference on System Sciences, IEEE Computer Society Press, Los Alamitos (1998a) 74-83.

Di Caro, G. und Dorigo, M.: „AntNet: Distributed stigmergetic control for communications networks", Journal of Artificial Intelligence Research, Band 9 (1998b) 317-365.

Di Caro, G. und Dorigo, M.: „Ant colonies for adaptive routing in packet-switched communications networks". In Eiben, A. E. / Back, T. / Schoenauer, M. / Schwefel, H.-P. (Hrsg): Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature, Springer-Verlag, Berlin/Heidelberg (1998c) 673-682.

Di Caro, G. und Dorigo, M.: „An adaptive multi-agent routing algorithm inspired by ants behavior". In Hawick, K. A. und James, H. A. (Hrsg): Proceedings of PART98 – Fifth Annual Australiasian Conference on Parallel and Real-Time Systems, Springer, Singapore (1998d) 261-272.

Doerner, K. / Hartl, R. F. / Reimann, M.: „Cooperative ant colonies for optimizing resource allocation in transportation". In Boers, E. J. W. et al. (Hrsg): Applications of Evolutionary Computing, Springer, Berlin/Heidelberg (2001) 70-79.

Dorigo, M. und Di Caro, G.: „The Ant Colony Optimization Meta-Heuristic". In Corne, D. / Dorigo, M. / Glover, F. (Hrsg): New Ideas in Optimization, McGraw-Hill, London (1999) 11-32.

Dorigo, M. und Gambardella, L. M.: „Ant Colony System: A cooperative learning approach to the traveling salesman problem", IEEE Transactions on Evolutionary Computation, Band 1 (1997a) 53-66.

Dorigo, M. und Gambardella, L. M.: „Ant colonies for the traveling salesman problem", BioSystems, Band 43 (1997b) 73-81.

Dorigo, M. / Maniezzo, V. / Colorni, A.: „Ant System: Optimization by a colony of cooperation agents", IEEE Transactions on Systems, Man, and Cybernetics - Part B, Band 26, Nr. 1 (1996) 29-41.

Gambardella, L. M. und Dorigo, M.: „An Ant Colony System hybridized with a new local search for the sequential ordering problem", INFORMS Journal on Computing, Band 12, Nr. 3 (2000) 237-255.

Gambardella, L. M. / Taillard, E. / Agazzi, G.: „MACS-VRPTW: A multiple Ant Colony System for vehicle routing problems with time windows". In Corne, D. / Dorigo, M. / Glover, F. (Hrsg): New Ideas in Optimization, McGraw-Hill, London (1999) 63-76.

Gambardella, L. M. / Taillard, E. D. / Dorigo, M.: „Ant colonies for the QAP" (Technical Report IDSIA 4-97), Journal of the Operational Research Society, Band 50, Nr. 2 (1999) 167-176.

Glaeske, G.: „Integrierte Versorgung in Deutschland – Rahmenbedingungen für mehr Effektivität und Effizienz?". In Preuß, K.-J. / Räbiger, J. / Sommer, J. H. (Hrsg): Managed Care - Evaluation und Performance-Measurement integrierter Versorgungsmodelle - Stand der Entwicklung in der EU, der Schweiz und den USA, Stuttgart, New York (2002) 4-19.

Goss, S. / Aron, S. / Deneubourg, J.-L. / Pasteels, J. M.: „Selforganized shortcuts in the Argentine ant", Naturwissenschaften, Band 76 (1989) 579-581.

Guntsch, M. und Middendorf, M.: „Pheromone modification strategies for ant algorithms applied to dynamic TSP". In Boers, E. J. W. et al. (Hrsg): Applications of Evolutionary Computing, Springer, Berlin/Heidelberg (2001) 213-222.

Hölldobler, B. und Willson, E. O.: The Ants, Harvard University Press, Cambridge, 1990.

Iredi, S. / Merkle, D. / Middendorf, M.: „Bi-Criterion optimization with multi colony ant algorithms“. In Zitzler, E. et al. (Hrsg): Proceedings on the First International Conference on Evolutionary Multi-Criterion Optimization, Springer, Zürich (2001) 359-372.

Keeney, R. und Raiffa, H.: Decisions with Multiple Objectives, Cambridge 1993

Lin, S.: “Computer solutions of the traveling salesman problem”, Bell Systems Journal, Band 44 (1965) 2245-2269.

Lin, S. und Kernighan B. W.: „An effective heuristic algorithm for the traveling salesman problem“, Operations Research, Band 21 (1973) 498-516.

Lo, C. D. / Srisa-an, W. / Chang, J. M. / Chern, J. C.: „The effect of 2-opt and initial population generation on solving the traveling salesman problem unsing genetic algorithms”, Proceedings of 5th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida (2001) 282-287.

Maniezzo, V.: „Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem“ (Technical Report CSR 98-1), INFORMS Journal on Computing, Band 11, Nr. 4 (1999) 358-369.

Maniezzo, V. / Muzio, L. / Colorni, A. / Dorigo, M.: „Il sistema formiche applicato al problema dell’assegnamento quadratico“, Technical Report 94-058, Politecnico di Milano, Italy, 1994.

Middendorf, M. / Reischle, F. / Schmeck, H.: „Information exchange in multi colony ant algorithms“. In Rolim, J. (Hrsg.): Parallel and Distributed Computing, Proceedings of the Fifteenth IPDPS 2000 Workshops, Third Workshop on Biologically Inspired Solutions to Parallel Processing Problems, Cancun, Mexico, Springer (2000) 645-652.

Monmarché, N. / Venturini, G. / Slimane M.: „On how Pachycondyla apicalis ants suggest a new search algorithm“, Future Generation Computer Systems, Band 16 (2000) 937-946.

Moy, J.: „Link-state routing”. In Steenstrup, M. E. (Hrsg): Routing in Communications Networks, Englewood Cliffs, Prentice-Hall (1995) 137-157.

Orth, C: Unternehmensplanspiele in der betriebswirtschaftlichen Aus- und Weiterbildung, Reihe: Personal-Management, Band 18, 1. Auflage, Josef Eul Verlag, Lohmar – Köln, 1999.

Preuß, K.-J. / Räbiger, J. / Sommer, J. H. (Hrsg), Managed Care - Evaluation und Performance-Measurement integrierter Versorgungsmodelle - Stand der Entwicklung in der EU, der Schweiz und den USA, Stuttgart, New York, 2002.

Randall, M. und Tonkes E.: „Solving network synthesis problems using ant colony optimization”. In Monostori, L. / Vancza, J. / Ali, M. (Hrsg): Proceedings of the 14th International Conference on Industrial and Engineering Applicatins of Artificial Intelligence and Expert Systems, Budapest, Springer (2001) 1-10.

Rosenkrantz, D. J. / Stearns, R. E. / Lewis, P. M.: „An analysis of several heuristics for the travelling salesman problem”, SIAM Journal on Computing, Band 6 (1997) 563-581.

Stützle, T. und Hoos, H.: „Improving the Ant-System: A detailed report on the MAX-MIN Ant System” Technical Report AIDA-96-12, FG Intellektik, TH Darmstadt, 1996. (http://www.cs.ubc.ca/spider/hoos/ps/aida-96-12r.ps, Zugriff am 8. Dez. 2002)

Stützle, T. und Hoos, H.: „Improvements on the Ant-System: Introducing the MAX-MIN Ant System“. In Smith, G. D. / Steele, N. C. / Albrecht, R. F. (Hrsg): Proceedings of the International Conference of Artificial Neural Networks and Genetic Algorithms, Springer, Wien (1997a) 245-249.

Stützle, T. und Hoos, H.: „MAX-MIN Ant System and local search for the travelling salesman problem“. In Baeck, T. / Michalewicz, Z. / Yao, X. (Hrsg): Proceedings of International Conference on Evolutionary Computation (1997b) 309-314.

Stützle, T. und Hoos, H.: „MAX-MIN Ant System and local search for combinatorial optimization problems“. In Voss, S. / Martello, S. / Osman, I. H. / Roucairol, C. (Hrsg): Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer, Boston (1999) 313-329.

Taillard, E. und Gambardella, L. M.: „Adaptive memories for the quadratic assignment problem” Technical Report IDSIA-87-97, IDSIA, Lugano, Schweiz, 1997. (http://ina.eivd.ch/collaborateurs/etd/articles.dir/IDSIA-87-97.ps, Zugriff am 8. Dez. 2002)

Technische Universität Berlin (Hrsg): http://www.bionik.tu-berlin.de/kompetenznetz/bionik/bionik.html, Zugriff am 10. Jan. 2003.

Vincke, P., Multicriteria Decision-Aid, Wiley, Chichester 1992.