Ein Hybrid-Verfahren zur Bearbeitung Kombinatorischer Optimierungsprobleme

Die Integration von Multiagentensystemen und Genetischen Algorithmen
in einem Framework zur Behandlung von Optimierungsproblemen aus der Praxis

Roland Hesse

Technische Universität München
Institut für Informatik

Ein Hybrid-Verfahren zur Bearbeitung Kombinatorischer Optimierungsprobleme

Die Integration von Multiagentensystemen und Genetischen Algorithmen
in einem Framework zur Behandlung von Optimierungsproblemen aus der Praxis

Roland Hesse

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. B. Radig

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Dr. h.c. mult. W. Brauer, em.
2. Univ.-Prof. Dr. A. Koch, Universität Salzburg (Österreich)
3. Univ. Prof. Dr. P. Struss

Die Dissertation wurde am 11.05.2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 23.10.2007 angenommen.

Kurzfassung

Die vorliegende Arbeit beschreibt die Entwicklung eines Hybrid-Verfahrens zur Bearbeitung kombinatorischer Optimierungsprobleme (KOP), seine Implementierung als Softwaresystem in Form eines Framework sowie dessen Anpassung und Anwendung auf konkrete Probleme. Einsatzbereich des Framework sind partiell evaluierbare KOP, insbesondere Probleme mit Raumbezug, wie sie beispielsweise in der Wirtschafts- und Verkehrsgeographie auftreten. Ziel ist es, vorhandenes Anwenderwissen auszunutzen, um den Optimierungsvorgang gegenüber rein zufallsbasierten Verfahren zu beschleunigen. Das Wissen betrifft Methoden zur Bewertung von Teillösungen und Kriterien für die Auswahl erfolgversprechender Variationsoperatoren.

Die Kernidee besteht in der Verknüpfung von Genetischen Algorithmen (GA) mit einem Multiagentensystem (MAS) zu einem Hybridverfahren, genannt *Genetische Konferenz (GK)*. Die Agenten des MAS erweitern einen zunächst universell einsetzbaren GA dabei um problemspezifisch definierte Operationen zur Bewertung, Selektion und Variation. Jeder Agent verwaltet *eines* der dem KOP zugrunde liegenden Objekte und versucht, dessen Konfiguration „lokal“ zu optimieren, indem er die entsprechenden Entscheidungsvariablen überwacht und einstellt.

Während ein herkömmlicher GA lediglich eine „globale“ Bewertung von Lösungen vorsieht, bewerten die Agenten der GK Lösungen zusätzlich *partiell* hinsichtlich problemspezifisch festgelegter Kriterien. Die so gesammelten Informationen werden zur zielgerichteten Variation von Lösungen eingesetzt: In einer Lösung wird vorzugsweise die Konfiguration eines Objekts variiert, dessen Agent „Verbesserungspotential“ meldet, weil er die derzeitige Konfiguration als ungünstig einstuft. Die Auswahl eines Variationsoperators erfolgt anhand von Koeffizienten, mit denen der Agent Lösungen hinsichtlich unterschiedlicher Kriterien bewertet. Die vom Anwender vorgegebene Wahrscheinlichkeit, mit der ein Operator in einer bestimmten Situation angewendet wird, kann im Laufe des Verfahrens von den Agenten angepasst werden – die Agenten können so aus dem Erfolg früherer Aktionen lernen.

Die Agenten arbeiten nicht nur auf dem zentralen „Lösungspool“ eines Standard-GA, sondern verfügen zusätzlich über individuelle Gedächtnisse, in denen sie partiell besonders günstige Vorschläge ablegen können. So wird deren Verdrängung vermieden und die enthaltenen Informationen können später zur Variation anderer Vorschläge wieder abgerufen werden. Des Weiteren können auf Grundlage eines Vorschlags auch Koalitionen unter mehreren mit der Lösung zufriedenen Agenten gebildet werden, um die Kombination der entsprechenden Konfigurationen vor dem „Vergessen“ zu bewahren.

Das Framework ist zwar universell einsetzbar, muss aber für die Bearbeitung konkreter Problem(exemplar)e durch Implementierung abstrakter Methoden angepasst werden. Dafür wird ein Vorgehensmodell mit einer exemplarischen Anpassung auf das *Traveling Salesman Problem* geliefert. Zum Test wurden Optimierungsläufe mit unterschiedlichen Varianten dieser Implementierung durchgeführt. Dabei zeigte das integrierte Verfahren eine deutlich höhere Performanz als Varianten, die die Basisverfahren MAS und GA jeweils gesondert simulieren.

Zuletzt wird die Anpassung auf drei Optimierungsprobleme aus der Praxis beschrieben. Dabei handelt es sich um ein Standortplanungsproblem, die Optimierung Integraler Taktfahrpläne sowie ein Zuordnungsproblem aus dem universitären Bereich. Die drei Implementierungen liefern überzeugende Resultate und zeigen, dass entsprechende Anpassungen des Framework sich in ganz unterschiedlichen Anwendungsgebieten als wertvolle Planungsinstrumente erweisen können.

Abstract

This thesis outlines the development of a hybrid algorithm for the processing of Combinatorial Optimization Problems (COP) and its implementation in the form of a framework. The framework is then adapted and applied to concrete problems. The scope of the framework envelops COP that allow the evaluation of partial solutions; in particular it addresses COP with geographic context. Such problems occur in the fields of economic or transportation geography, for instance. The goal is to exploit available user knowledge in order to speed up the optimization process. This knowledge concerns the fitness of partial solutions as well as choice criteria for suitable variation operators.

A central idea of the thesis is to combine Genetic Algorithms (GA) with a Multi-Agent-System (MAS) to a hybrid algorithm, called *Genetic Conference (GC)*. The agents of the MAS increase a universal GA with problem-specific methods to evaluate and modify solutions to the problem. Each agent is responsible for one of the objects forming the COP and attempts to optimize its configuration locally by monitoring and adjusting the corresponding decision variables.

A conventional GA provides only a global evaluation of solutions. However, the agents of the GC additionally measure partial solutions, relating to specific criteria. This information is used for a goal-oriented variation of solution candidates. In a given solution, preferably those configurations of objects are modified, whose agents claim room for improvement based on their dissatisfaction about the present configuration. Variation operators are chosen on the basis of coefficients used by the agents for evaluating solutions with regard to various criteria. The probability or the choice of an operator in a specific situation is provided by the user. The agents may adapt these probabilities during the optimization process. Thus, they can learn by past experiences.

The algorithm works on more than just a common chromosome-pool as a standard-GA would do. In addition, agents have individual memories and use them to store those solutions, which are favored by partial evaluation. The removal of a solution from the central pool does not result in a total loss, since the memorized information can later be used for the variation of other solutions. The agents are also able to build coalitions based on particular solutions that satisfy their members most widely. This way they can save the combination of the corresponding configurations from oblivion.

The Genetic Conference is defined as an all-purpose algorithm that has to be adapted to specific problems by implementing abstract methods. A process model and a prototypic adaptation for the treatment of *Traveling Salesman Problems* is provided. Tests of this implementation show a significant speed-up of the integrated MAS & GA algorithm compared to versions that simulate the base algorithms in an isolated way.

The framework is then adapted to three real-life optimization problems. These are: a location planning problem; the optimization of integrated periodic timetables; and an assignment problem from the university domain. The three case studies yield most encouraging results. These indicate that the framework can be tailored to be a most useful planning tool for a wide range of applications.

Danksagung

Mein tiefster Dank gilt Herrn Prof. Dr. Dr. h.c Wilfried Brauer zunächst dafür, dass er sich bereit erklärte, die Betreuung meiner externen Promotion zu übernehmen. Besonders danken möchte ich ihm für die vielen Gespräche und Anregungen, aber auch für die kreative Freiheit, die er mir bei der Erstellung der Arbeit gelassen hat.

Herrn Prof. Dr. Andreas Koch danke ich für seine Bereitschaft, die Arbeit von Seiten der Geographie aus mit zu betreuen und für seine konstruktiven Ratschläge hinsichtlich des Aufbaus der Arbeit.

Die Hinweise von Herrn Prof. Dr. Struss haben am Ende wesentlich zur Lesbarkeit der Arbeit und zur Verdeutlichung der Ziele beigetragen. Dafür gilt ihm mein besonderer Dank, ebenso wie für seine kurzfristige Zusage, das Promotionsverfahren als Gutachter und Prüfer zu unterstützen.

Herrn Prof. Dr. Günther Heinritz möchte ich danken dafür, dass er mir als „Fachfremden“ meine derzeitige Stelle an der Fakultät für Geographie angeboten und damit überhaupt erst die Möglichkeit zur Promotion eröffnet hat.

Meine Eltern und meine Freundin waren mir immer eine große Stütze. Insbesondere mit meinem Vater konnte ich viele anregende Gespräche über die Arbeit führen, die ganz wesentlich zu ihrem Gelingen beigetragen haben. Ihnen und allen anderen Verwandten und Freunden, die mir geholfen haben, danke ich sehr.

Vorbemerkungen zum Sprachgebrauch

Die Implementierung des entwickelten Optimierungsverfahrens stellt ein *Framework* („Rahmenwerk“) dar. Der Begriff des *Framework* taucht daher des Öfteren in der Arbeit auf, und zwar teilweise (wie in der letzten Zeile) auch im Genitiv. Auf eine „deutsche“ Flexion des englischen Begriffs („*des Frameworks*“) wird dabei durchgängig verzichtet.

Das Problem des Handlungsreisenden (*Traveling Salesman Problem*) wird politisch korrekt neuerdings oft als *Traveling Salesperson Problem* bezeichnet. Da sie bekannter ist, findet sich in der Arbeit immer die ursprüngliche Bezeichnung *Traveling Salesman Problem*. Auch alle übrigen Personen- und Funktionsbezeichnungen sollen für Frauen und Männer in gleicher Weise gelten.

Für meine Eltern.

Inhaltsverzeichnis

Kapitel 1

Einordnung, Ziele und Aufbau der Arbeit

1.1 Einleitung und Zielsetzung der Arbeit

Gegenstand dieser Arbeit sind Kombinatorische Optimierungsprobleme (KOP) aus der Praxis und Ansätze zu ihrer Bearbeitung, die problemspezifisches Anwenderwissen integrieren.

Man unterscheidet in der Optimierungstheorie abstrakte *Optimierungsprobleme* und deren konkrete *Problemexemplare*. Unter dem Begriff „Traveling-Salesman-Problem“ (TSP) fasst man beispielsweise Exemplare zusammen, bei denen kürzeste Rundreisen zu einer gegebenen Menge von Städten gebildet werden sollen. Die einzelnen Exemplare des TSP unterscheiden sich dabei in Anzahl und Lage (bzw. Entfernung) der zu besuchenden Städte.

Mathematisch kann ein Exemplar eines Optimierungsproblems durch eine (endliche) Menge von *Entscheidungsvariablen* beschrieben werden. Gültige Lösungen entstehen durch Belegung der Variablen mit konkreten Werten unter Beachtung eventueller Nebenbedingungen. Die Qualität einer solchen Lösung wird mit Hilfe einer *Zielfunktion* gemessen. Ziel der Optimierung ist es, eine gültige Lösung mit bestmöglicher Qualität zu finden. Ein Optimierungsproblem besteht aus einer Menge solcher Exemplare, deren Zielfunktionen sich zwar beispielsweise in der Anzahl der Entscheidungsvariablen unterscheiden, sich jedoch strukturell ähnlich sind [PS82].

Als „*kombinatorisch*“ bezeichnet man Optimierungsprobleme, deren Entscheidungsvariablen endliche Wertebereiche haben, wie das etwa beim TSP der Fall ist. Solche Probleme treten in der Praxis - beispielsweise in Wirtschaft und Industrie - recht häufig auf und sind ein wichtiges Arbeitsfeld des Wissenschaftszweiges *Operations Research*. Ihr Lösungsraum ist zwar endlich, die Suche nach der besten Lösung durch vollständige Enumeration, d.h. der Evaluierung *aller* gültigen Lösungen, ist aber ab einer gewissen Größenordnung eines konkreten Exemplars nicht mehr mit vertretbarem Zeitaufwand durchführbar. Beim TSP steigt zum Beispiel die Anzahl der gültigen Lösungen mit der Fakultät der Anzahl der Städte an.

Schwerpunkt der Arbeit sollen kombinatorische Optimierungsprobleme mit *Raumbezug* sein, wie sie in der Wirtschafts- und Verkehrsgeographie auftreten. Bei solchen Problemen repräsentieren die Entscheidungsvariablen räumliche Einheiten und/oder Distanzberechnungen über diese Ein-

heiten gehen in die Evaluierung der Zielfunktion ein. Die hier betrachteten Probleme weisen eine gewisse *Dekomponierbarkeit* in Subprobleme über räumliche Teilgebiete auf: Teillösungen für solche Subprobleme lassen sich (relativ) unabhängig voneinander bewerten und zu Gesamtlösungen kombinieren. Dem Anwender ist es daher möglich, die *Teile* einer Lösung aufzuspüren, die noch Verbesserungspotential aufweisen. Meist sind ihm auch problemspezifische Variationsoperatoren bekannt, so dass er kleinere Problemexemplare relativ gut „von Hand“ bearbeiten kann.

Ab einer gewissen Problemgröße ist jedoch eine automatisierte Bearbeitung unumgänglich. Aufgrund der teilweise sehr spezifischen Anforderungen, die Probleme aus der Praxis mit sich bringen, stehen dem Anwender aber selten automatisierte Verfahren zur Verfügung, die auf seine speziellen Bedürfnisse zugeschnitten sind. Außerdem kann der Anwender in der Regel sein Expertenwissen nicht in bestehende Systeme einbringen, so dass das Verfahren in annehmbarer Zeit keine ausreichend gute Lösung findet.

Ziel der Arbeit ist daher die Entwicklung eines *Optimierungsverfahrens* für KOP aus der Praxis und seine prototypische Implementierung in Form eines *Framework*. Das Framework soll Möglichkeiten zur Integration problemspezifischen Anwenderwissens – speziell über die Bewertbarkeit von Teillösungen und über Kriterien zur Auswahl passender Variationsoperatoren – vorsehen, um dieses Wissen effizienzsteigernd ausnutzen. Es soll zunächst problemübergreifend definiert sein und auf einfache Weise für die Bearbeitung praktischer Probleme angepasst werden können. Dies soll anhand von Anwendungsbeispielen demonstriert werden.

Als Anwendungsfälle für das Verfahren wurden die folgenden Kombinatorischen Optimierungsprobleme ausgewählt:

Standortplanung: Ein Standortplanungsproblem aus dem Bereich des *Operations Research* bildet den ersten Anwendungsfall. Das Ziel ist, für Umschlagpunkte einer zweistufigen Distributionslogistik Standorte zu finden, die Transport- und Fixkosten minimieren, wobei eine möglichst vollständige Abdeckung der Kunden im Untersuchungsgebiet anzustreben ist. Die Anzahl zu planender Umschlagpunkte kann dabei ebenfalls Gegenstand der Optimierung sein.

Fahrplanoptimierung: Als zweiter Anwendungsfall wurde die Fahrplanoptimierung – konkret die Optimierung Integraler Taktfahrpläne – gewählt. Ziel ist hier, die Umsteigewartezeiten für Kunden in einem Verkehrsnetz durch günstige Vergabe von so genannten *Rendezvouz*-Zeiten für die Haltepunkte und entsprechende Abfahrtszeiten für die Verkehrsmittel zu minimieren. Dabei sind auch die Betriebskosten für den Fahrzeugeinsatz zu berücksichtigen. Zusätzlich können für bestimmte Strecken Beschleunigungsmaßnahmen vorgeschlagen werden, wobei dann Investitionskosten für die kürzeren Fahrzeiten zu Buche schlagen.

Zuordnungsproblem: Drittes Anwendungsbeispiel ist ein Zuordnungsproblem aus dem universitären Bereich. Das Optimierungsverfahren wird zur Zuteilung von Praktika bei der so genannten *Klinischen Rotation* im Studium der Veterinärmedizin an der LMU München herangezogen. Unter Beachtung bestimmter Nebenbedingungen (maximale Teilnehmerzahlen, erlaubte Kombinationen von Praktika) soll eine hinsichtlich der Wünsche der Studenten möglichst günstige Zuteilung erreicht werden. Dieses Anwendungsbeispiel soll demonstrieren, dass das entwickelte Verfahren nicht nur auf Probleme mit Raumbezug anwendbar ist, sondern auch auf andere KOP, die eine gewisse Dekomponierbarkeit aufweisen.

1.2 Die Bearbeitung von KOP mit Genetischen Algorithmen und Multiagentensystemen

Genetische Algorithmen

Ein bekanntes Verfahren zur Bearbeitung Kombinatorischer Optimierungsprobleme stellen die so genannten Genetischen Algorithmen dar. GA gehören zu den problemunabhängigen, *heuristischen* Lösungsverfahren für Optimierungsprobleme. Unter einer Heuristik versteht man ein Verfahren, das oft gute Lösungen in annehmbarer Zeit findet, den Anspruch auf das Finden (global) optimaler Lösungen jedoch zugunsten dieses Zeitvorteils zurückstellt.

GA sind weitgehend unabhängig vom zu bearbeitenden Problem und nutzen die Zielfunktion nur zur Evaluierung iterativ generierter Lösungen. Solche universellen Heuristiken werden auch als „schwache Verfahren“ [Nis97] bezeichnet. Damit wird zum Ausdruck gebracht, dass im Unterschied zu problemspezifischen „starken Verfahren“ (ebd.) kein Wissen über das Problembeispiel bzw. die Struktur der Zielfunktion in die Suche nach der besten Lösung eingeht.

GA bearbeiten parallel eine Menge (*Population*) von Lösungen (*Individuen*), die auf so genannten *Chromosomen* kodiert werden. In Anlehnung an das Vorbild der natürlichen Evolution werden iterativ neue Lösungen (*Nachkommen*) durch *Rekombination* und *Mutation* von Individuen aus der momentanen Population (*Eltern*) gebildet. Bei der Generierung der Nachkommen werden als Eltern bevorzugt „fittere“ Individuen herangezogen, d.h. solche, die Lösungen höherer Qualität repräsentieren. Mit dem so entstehenden Selektionsdruck versucht man, die Qualität der Individuen in der Population immer weiter zu erhöhen.

Um die Suche nach guten Lösungen zu beschleunigen, können die ursprünglich randomisierten Mutations- und Rekombinationsoperatoren durch problemspezifische Operatoren ersetzt oder ergänzt werden. Dieses Vorgehen wird auch als *Hybridisierung* bezeichnet. Während die Integration spezieller Variationsoperatoren mittlerweile zum Standard bei der Anwendung Genetischer Algorithmen auf praktische Optimierungsprobleme gehört, fehlen bisher Formalismen für die Auswahl der *Stelle*, an der ein Operator angewendet werden soll. Des Weiteren fehlen Methoden zur zielgerichteten Auswahl eines Operators, wenn mehrere zur Verfügung stehen. Ein hybridisierter GA kennt also problemspezifische Operatoren, kann aber nur zufällige Entscheidungen darüber treffen, welchen der Operatoren er in einer bestimmten Situation anwenden soll, und welche Entscheidungsvariablen mit dem Operator modifiziert werden sollen.

Multiagentensysteme

Für Optimierungsprobleme, die die partielle Bewertung von Teillösungen zulassen, bieten sich Techniken der verteilten Lösung an. Eine Möglichkeit besteht darin, die Entscheidungsvariablen des Problems auf die Agenten eines *Multiagentensystems* zu verteilen. Jeder Agent verfügt über gewisses problemspezifisches Wissen und versucht – möglicherweise in Kooperation mit den übrigen Agenten – die Belegungen seiner Variablen optimal einzustellen. Ein solcher Ansatz wird beispielsweise mit *DCHS (Distributed Constrained Heuristic Search)* bei der verteilten Bearbeitung von Constraint-Satisfaction-Problemen (CSP) verfolgt [SRSF91]. Ein anderes Beispiel für die Anwendung eines auf diesem Prinzip beruhenden, hierarchisch organisierten MAS ist der Algorithmus, den Galanda und Weibel [GW03] zur Generalisierung von Polygonmosaiken für

thematische und topographische Karten vorschlagen. Das MAS wird hier eingesetzt, um nach einem möglichst guten Kompromiss bezüglich der Erfüllung von Nebenbedingungen (*Constraints*) zu suchen. Dazu werden auf der unteren Ebene die Kanten von Polygonen und auf höheren Ebenen ganze Polygone und Polygongruppen von Agenten repräsentiert, die entsprechend der Constraints unterschiedliche Ziele hinsichtlich des Aussehens der entsprechenden Kartenelemente haben. Die Agenten versuchen im Laufe des Verfahrens, diese Ziele durch Variation der von ihnen repräsentierten Elemente zu erreichen. In Kooperation generalisieren die Agenten so eine gegebene thematische Karte für die Darstellung in einem bestimmten Maßstab. Das Verfahren ist als gieriges (engl. *Greedy-*) Verfahren einzustufen: Jeder Agent modifiziert die momentane Lösung so, dass lokal (für ihn) die größte Verbesserung des Zielfunktionswertes zu erwarten ist.

Dieses Vorgehen ist im Prinzip auf andere KOP übertragbar, sofern partielle Bewertungsmethoden bekannt sind. Wie auch bei anderen Greedy-Verfahren besteht jedoch die Gefahr, dass zwar schnell ein lokales Optimum gefunden wird, das Verfahren aber dort „hängen bleibt“ und kein globales Optimum findet – das gilt insbesondere dann, wenn die Wechselwirkungen unter Entscheidungsvariablen groß sind. Außerdem arbeitet das Verfahren immer nur an einem Lösungsvorschlag (*Einpunktverfahren*), und sieht zunächst nicht die Möglichkeit der Rekombination von Lösungen vor. Diese Eigenschaft trägt zusätzlich zum Risiko frühzeitiger Konvergenz bei.

1.3 Das Konzept eines integrierten Verfahrens

GA bieten sich zwar prinzipiell für die Lösung kombinatorischer Optimierungsprobleme an, sollten jedoch mit problemspezifischen Heuristiken hybridisiert werden, um ihre Effizienz zu steigern. MAS können dagegen als problemspezifische Optimierungsverfahren eingesetzt werden, indem die Agenten jeweils die Entscheidungsvariablen des Problems verwalten – eine gewisse Dekomponierbarkeit des Problems vorausgesetzt. Durch die zielgerichtete Arbeitsweise der Agenten auf Grundlage des bereitgestellten Problemwissens ist hier eine höhere Konvergenzgeschwindigkeit zu erwarten, es besteht jedoch die Gefahr, dass das Verfahren in einem *lokalen* Optimum konvergiert. Die Vorteile von MAS und GA sind bei der Anwendung auf kombinatorische Optimierungsprobleme also unterschiedlich gelagert und können sich ergänzen.

Der neuartige Ansatz für das zu entwickelnde Verfahren besteht aus einer Verknüpfung eines Genetischen Algorithmus mit einem Multiagentensystem. Das MAS soll als Wissensträger zu verwendet werden, um einen GA durch Möglichkeiten der zielgerichteten Auswahl zu variierender Lösungsteile sowie dazu passender Variationsoperatoren zu beschleunigen. Bei der Integration der beiden Ansätze MAS & GA soll so gleichzeitig (1) die Performanz gegenüber einem GA durch die Nutzung Anwender-spezifischen Wissens erhöht und (2) die Gefahr vorzeitiger Konvergenz in lokalen Optima gegenüber einem MAS-Ansatz, der nur an einer Lösung arbeitet, verringert werden.

Grundkonzept des Verfahrens ist die individuelle Verwaltung der *Entscheidungsvariablen* des Optimierungsproblems durch *Agenten*, die parallel einen gemeinsamen Pool von Lösungsvorschlägen bearbeiten und dabei die wesentlichen Ablaufschritte *Initialisierung*, *Selektion*, *Bewertung* und *Variation* eines GA übernehmen. Die Agenten verfügen über *partielle Bewertungsfunktionen* für

die Belegung der jeweils verwalteten Variablen, auf deren Basis sie das *lokale Verbesserungspotential* des Vorschlags durch Variation dieser Variablen abschätzen können.

Die Agenten bringen im Laufe des Verfahrens immer wieder neue Vorschläge in den Pool ein, die auf bisherigen Vorschlägen basieren. Dazu werden wie beim herkömmlichen GA Vorschläge mutiert und rekombiniert (*Variation*). Die Gesamtgüte eines Lösungsvorschlags ergibt sich durch Aggregation der partiellen Bewertungsfunktionen (*Bewertung*). Insgesamt weniger gut bewertete Lösungsvorschläge werden regelmäßig aus dem Pool entfernt (*Selektion*).

Mit Hilfe der Analyse des lokalen Verbesserungspotentials von Lösungsvorschlägen können die Entscheidungsvariablen, die variiert werden sollen, im Unterschied zum klassischen GA jedoch *gezielt* ausgewählt werden. Außerdem wird auf Grundlage der lokalen Bewertungsfunktionen eine *situationsbezogene* Auswahl des Operators möglich, der zur Variation von Vorschlägen eingesetzt wird.

Die Verwendung der Multiagenten-Technologie erlaubt die Nutzung zusätzlicher Konzepte, wie individuelle „Gedächtnisse“, in denen partiell besonders günstige Vorschläge abgelegt werden können, sowie das „Lernen“, in bestimmten Situationen spezifische Variationsoperatoren einzusetzen. Außerdem begünstigt die individuelle Verwaltung und Bewertung einzelner Entscheidungsvariablen durch die Agenten effiziente Formen der Rekombination von Teillösungen.

Das Verfahren erinnert an eine *Konferenz* mit Vertretern („Agenten“) unterschiedlicher Interessensgruppen, die *Vorschläge* einbringen und die Vorschläge der übrigen Teilnehmer nach eigenen Interessen abwandeln, mit dem Ziel, einen für alle Vertreter zufriedenstellenden Vorschlag zu erarbeiten. Aus diesem Grund – und in Anlehnung an den zugrunde liegenden Ansatz der Genetischen Algorithmen – wird das entwickelte Optimierungsverfahren als *„Genetische Konferenz“* bezeichnet.

1.4 Wichtigste Ergebnisse der Arbeit

Die Genetische Konferenz wurde prototypisch als Framework für die Bearbeitung von KOP implementiert. Das Framework verfügt über eine graphische Benutzeroberfläche und sieht die Möglichkeit der parallelen Bearbeitung mehrerer Probleme vor.

Durch die Ausformulierung abstrakter Methoden kann der Anwender problemspezifisches Wissen einbringen und den Optimierungsalgorithmus im Framework so an das jeweilige Problem anpassen. Dabei kann er Wissen über Möglichkeiten der partiellen Bewertung von Lösungsteilen ebenso integrieren, wie bekannte lokale Suchheuristiken. Mittels einer Wahrscheinlichkeits-Matrix kann der Anwender außerdem vorgeben, welche Operatoren in welchen Situationen angewendet werden sollten. Die Agenten nutzen diese Informationen, um möglichst gute Belegungen ihrer Variablen zu finden.

Eine genauere Kenntnis der Eigenschaften der Zielfunktion ist nicht erforderlich, der Algorithmus kann auch mit wenigen Vorgaben arbeiten. Ebenso ist es - beispielsweise zu Vergleichszwecken - möglich, nur „herkömmliche“, randomisierte Suchoperatoren zu verwenden.

Das Vorgehen bei der Anpassung des Framework auf konkrete Optimierungsprobleme wird anhand des TSP exemplarisch dargestellt. Um die praktische Anwendbarkeit des Verfahrens in unterschiedlichen Problemumgebungen zu demonstrieren, wurde das Framework dann für die

Bearbeitung der drei eingangs beschriebenen Anwendungsfälle problemspezifisch implementiert. Mit der Genetischen Konferenz lassen sich auch bei Verwendung relativ einfacher Bewertungsmethoden und Variationsoperatoren in allen drei Fällen sehr gute Optimierungsergebnisse erzielen.

- Bei der Firma WiGeoGIS wird das für die Standortplanung angepasste System mittlerweile erfolgreich zur Planung von Distributionslogistiken eingesetzt. In einem Projekt für die Bosch-Siemens-Hausgeräte konnten neue Standorte für Umschlagpunkte ermittelt werden, bei denen die prognostizierten Kosten gegenüber dem momentanen Stand deutlich niedriger ausfallen.
- Das auf die Fahrplanoptimierung angepasste System wurde in das bestehende Expertensystem *OptiTakt* eingebettet, das weitere Werkzeuge zur Erstellung, Analyse und Optimierung Integraler Taktfahrpläne bereitstellt. Das Gesamtsystem wurde bereits in mehreren Projekten mit professionellen Planungsunternehmen im Schienen-Personenverkehr erfolgreich eingesetzt.
- Die Verteilung der Praktikumsplätze zur Klinischen Rotation an der Fakultät für Tiermedizin der LMU München findet seit der Planungsperiode für die Praktika im Jahr 2007 ausschließlich mit dem für dieses Zuordnungsproblem implementierten Framework statt. Die Ergebnisse waren so überzeugend, dass mittlerweile auch weitere Optimierungsprobleme an der Fakultät für Tiermedizin, wie die Einteilung von Prüfungsgruppen bei Promotionen mit der Genetischen Konferenz bearbeitet werden.

Für alle drei Anwendungsbeispiele wurden Tests durchgeführt, die die deutliche Überlegenheit des integrierten Verfahrens der Genetischen Konferenz aufzeigen gegenüber Varianten, die die Basisverfahren isoliert simulieren.

Anhand zweier Problembeispiele des TSP werden die Auswirkungen der Verwendung unterschiedlicher Konzepte dann noch einmal im Einzelnen evaluiert. Es zeigen sich hier deutliche Beschleunigungen durch die Möglichkeit der partiellen Bewertung und der situationsbezogenen Auswahl von Variationsoperatoren. Auch wenn nur permutationsbasierte Operatoren eingesetzt werden, können durch die zielgerichtete Auswahl zu variierender Lösungsteile deutlich schneller gute Lösungen gefunden werden, als mit einem klassischen GA, der die Operatoren an zufälligen Stellen einsetzt.

Auch die Verwendung weiterer Konzepte wie beispielsweise die Verwendung individueller Gedächtnisse wirkt sich vorteilhaft auf Performanz des GA aus. Dabei wird die Konvergenzsicherheit eines genetischen Algorithmus durch die Verknüpfung mit dem MAS und die einhergehende Verwendung von Greedy-Techniken bei der Auswahl von Variationsstelle und -operator sogar noch erhöht.

1.5 Aufbau der Arbeit

Kapitel 2 liefert zunächst Definitionen und führt grundlegende Begriffe aus der Optimierungstheorie sowie das Traveling Salesman Problem als Standard-Problem der Kombinatorischen Optimierung ein.

In **Kapitel 3** wird der Anwendungsbereich des entwickelten Optimierungsverfahrens auf partiell evaluierbare Probleme mit eingeschränktem Anwenderwissen festgelegt. Außerdem werden die Anwendungsbeispiele Standortplanung, Fahrplanoptimierung und Praktikumszuordnung (Klinische Rotation) erläutert.

Kapitel 4 beschreibt gängige Optimierungsverfahren sowie Eignungskriterien der Verfahren für bestimmte Probleme. Ansätze der Evolutionären Algorithmen, speziell der Genetischen Algorithmen (GA) werden näher beschrieben und das Schemata-Theorem von Holland wird eingeführt. Aufbauend auf den NFL-Theoremen werden die Vorteile der Integration vorhandenen problemspezifischen Wissens in universelle Verfahren begründet. Im entwickelten Verfahren soll dazu ein Multiagentensystem (MAS) als Wissensträger herangezogen werden. Daher werden anschließend MAS eingeführt und Methoden vorgestellt, mit ihnen kombinatorische Optimierungsprobleme zu bearbeiten. Zuletzt werden Möglichkeiten diskutiert, universelle und problemspezifische Verfahren zu verknüpfen.

In **Kapitel 5** werden zunächst die Vor- und Nachteile von GA und MAS zur Bearbeitung von KOP, die zum Anwendungsgebiet des entwickelten Optimierungsverfahrens zählen, noch einmal zusammengefasst. Anschließend folgt eine formale Definition des aus GA und MAS verknüpften Verfahrens. Die zugrunde liegenden Konzepte werden dann im Detail besprochen.

Kapitel 6 beschreibt die Modellierung und Implementierung der Genetischen Konferenz in einem Programmsystem, das ein problemübergreifend anwendbares Framework für Optimierungsprobleme darstellt. Das Framework ist zwar an sich problemunabhängig, muss jedoch je nach Anwendungsfall auf das spezielle Problem angepasst werden. Dazu müssen im Wesentlichen Methoden, die in abstrakten Klassen zusammengestellt sind, konkret implementiert werden.

Ein Vorgehensmodell für diese Implementierungen wird in **Kapitel 7** anhand des Standard-Problems TSP erläutert. Anpassung und Anwendung des Framework auf die drei Anwendungsfälle werden beschrieben.

In **Kapitel 8** werden die einzelnen Konzepte des entwickelten Verfahrens anhand zweier TSP-Exemplare evaluiert.

Die Ergebnisse der Arbeit werden in **Kapitel 9** zusammengefasst. Die Arbeit schließt mit Fazit und Ausblick.

Im **Anhang** finden sich einige weiterführende Erläuterungen und Beispiele zum theoretischen Kapitel (Teil A). Diese sollen Anwendern des entwickelten Verfahrens, die beispielsweise aus der Wirtschafts- und Verkehrsgeographie stammen und naturgemäß über weniger Vorwissen zur Optimierungstheorie verfügen, einen leichteren Zugang zur Arbeit ermöglichen. Außerdem werden in einem zweiten Teil (Teil B) die Klassen des wichtigsten Programmpakets der prototypischen Implementierung im Detail beschrieben.

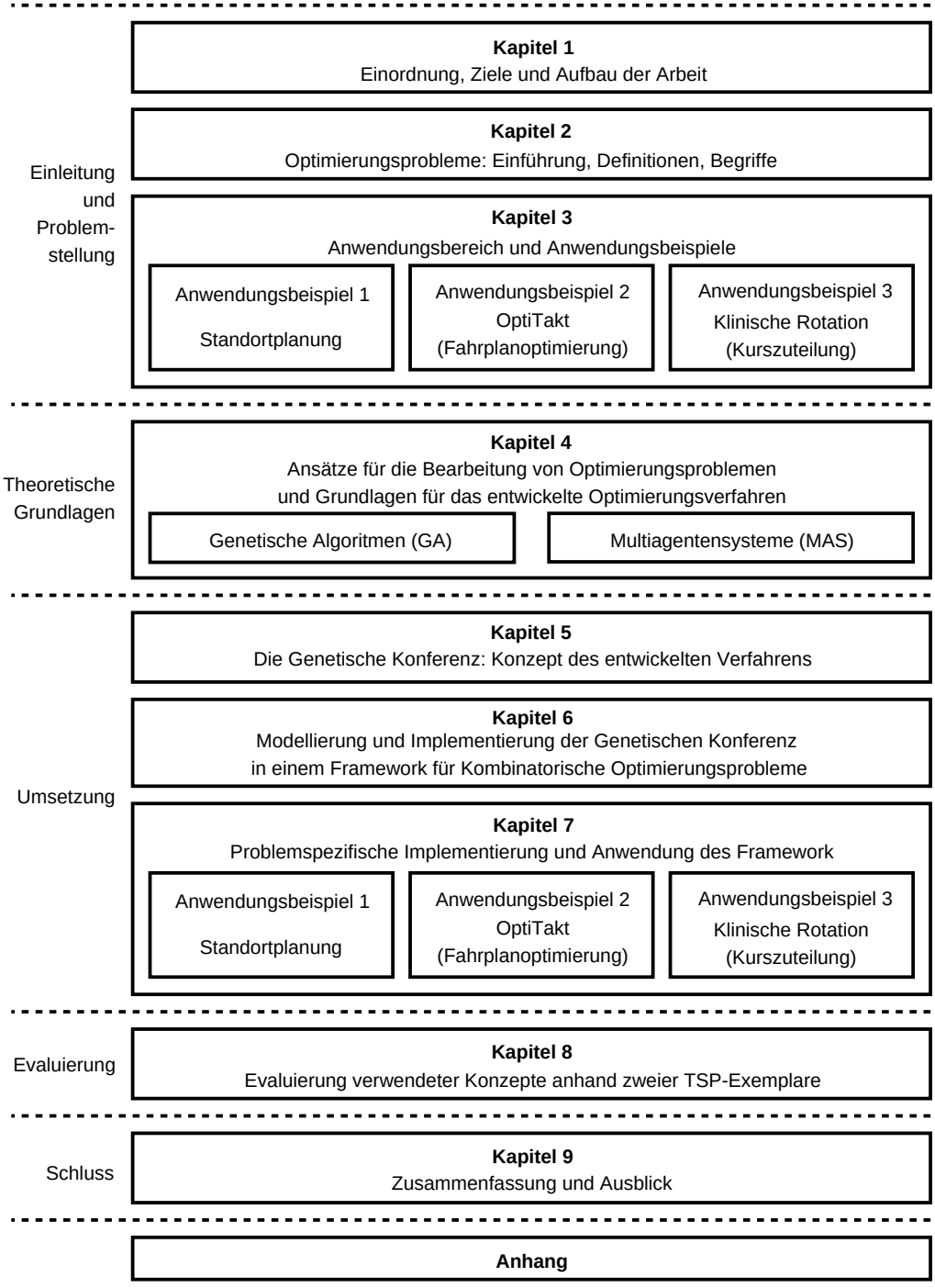


Abbildung 1.1: *Aufbau der Arbeit*

Kapitel 2

Optimierungsprobleme: Einführung, Definitionen, Begriffe

Kapitel 2 führt zunächst das Traveling Salesman Problem (TSP) als Standard-Optimierungsproblem ein und beschreibt die Relevanz von Optimierungsproblemen in der Praxis (Abschnitt 2.1).

Es folgt die Einführung wichtiger Definitionen (2.2) und Begriffe (2.3) aus der Optimierungstheorie sowie ein Abschnitt zur Komplexität von Optimierungsproblemen (2.4)

Das Kapitel schließt mit der Einführung von Kombinatorischen Optimierungsproblemen (KOP), die den Anwendungsbereich des im Zuge dieser Arbeit entwickelten Optimierungsverfahrens darstellen (2.5).

2.1 Einführung von Optimierungsproblemen und das Standard-Problem TSP

Optimierungsprobleme treten in der Praxis in vielen Bereichen auf. Die Aufgabenstellung besteht aus praktischer Sicht darin, in einer Entscheidungssituation aus einer Reihe von Alternativen hinsichtlich Kosten und Nutzen die bestmögliche auszuwählen.

Genau genommen setzt sich ein Optimierungsproblem aus einer *Menge von Problembeispielen (Exemplaren des Problems)* zusammen, die bezüglich der Aufgabenstellung gleich oder sehr ähnlich sind, sich aber in konkreten Parameterbelegungen unterscheiden.

Beispiel Traveling Salesman Problem (TSP):
Beim „Traveling Salesman Problem“ (TSP) besteht die Aufgabenstellung darin, kürzeste Rundreisen für eine gegebene Menge von Städten zu finden, wobei jede Stadt genau einmal besucht werden soll. Verschiedene Problembeispiele des TSP unterscheiden sich in der Anzahl und Lage der Städte bzw. der Entfernung der Städte untereinander sowie eventuell zusätzlichen Nebenbedingungen. Abbildung 2.1 zeigt zwei mögliche Rundreisen für ein TSP-Exemplar. Man unterscheidet verschiedene TSP-Varianten:

- Bei einem *symmetrischen* TSP gilt für die Distanzen d_{ij} und d_{ji} zwischen zwei Städten i und j: $d_{ij} = d_{ji}$.
- Von einem *metrischen* TSP spricht man, wenn zusätzlich die Entfernungen d_{ij}, d_{ik} und d_{jk} dreier Städte i, j und k jeweils die Dreiecksungleichung $d_{ij} \leq d_{ik} + d_{kj}$ erfüllen.
- Bei *euklidschen* TSP lassen sich die Städte zusätzlich so in eine zweidimensionale euklidsche Ebene einbetten, dass die Entfernungen ihrer euklidschen Distanz entsprechen.

Das TSP gilt als besonders anschauliches „Standardproblem“ der Optimierungstheorie, daher wird es im Folgenden des Öfteren als Beispiel zur Verdeutlichung bestimmter Begriffe und Sachverhalte herangezogen werden.[1] Dabei wird jeweils von der metrischen Variante ausgegangen. Das Vorgehensmodell für die Anpassung und Anwendung des im Zuge dieser Arbeit entwickelten Optimierungsverfahrens wird ebenfalls anhand des TSP erläutert.

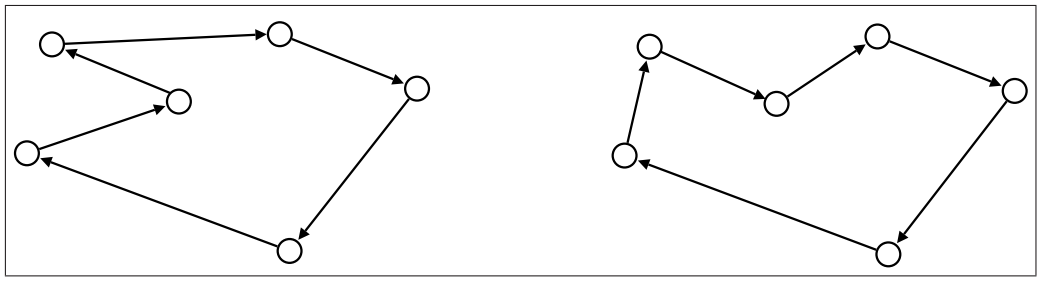


Abbildung 2.1: *Zwei Lösungsvorschläge für ein TSP-Exemplar*

[1] Einen Überblick über Varianten und Lösungsansätze geben z.B. [LLRS85] und [GP02].

Das TSP ist zwar an sich ein „theoretisches“ Problem. Gerade im Bereich der Logistik treten aber häufig ähnliche Probleme auf, darunter beispielsweise das Vehicle Routing Problem (VRP).

Beispiel Vehicle Routing Problem (VRP):
Beim „Vehicle Routing Problem“ (VRP) besteht die Aufgabe darin, möglichst kurze Auslieferungstouren für eine gegebene Anzahl von Fahrzeugen mit beschränkter Ladekapazität von einem Lager zu einer Menge von Zielorten mit bestimmter Warennachfrage zu finden. Unterschiedliche Problembeispiele ergeben sich beispielsweise durch Variation der Anzahl, Lage und Nachfrage der Zielorte sowie der Ladekapazitäten der Fahrzeuge. Außerdem kann das Problem durch die Aufnahme zusätzlicher Nebenbedingungen (beispielsweise bestimmte Zeitfenster, in denen die Kunden beliefert werden müssen) erweitert werden (vgl. [Chr85]). Abbildung 2.2 zeigt mögliche Auslieferungstouren für ein VRP-Exemplar.

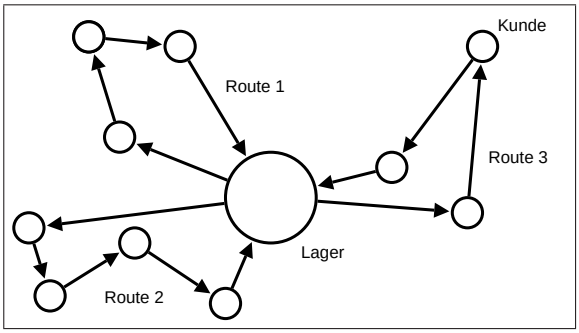


Abbildung 2.2: *Ein Lösungsvorschlag für ein VRP-Exemplar*

Die Aufgabenstellungen vieler Optimierungsprobleme stammen aus Bereichen der Wirtschaft und der Industrie. Mit guten Lösungen sind dort oft erhebliche Zeit und/oder Kosteneinsparungen verbunden. Die Informatik liefert Methoden, mit denen solche Probleme effizient – d.h. in annehmbarer Zeit – gelöst werden können, oder mit denen Lösungen gefunden werden können, die zumindest ausreichend gut hinsichtlich bestimmter Vorgaben sind.

Die Disziplin „Operations Research“

Mit der *Operations Research* (auch *Operational Research*, kurz OR), existiert sogar eine wissenschaftliche Disziplin, die ihr Hauptaugenmerk auf die Entwicklung und Anwendung von Entscheidungs- und Optimierungsmethoden für solche Probleme zur Unterstützung privater und öffentlicher Betriebe richtet. Das OR etablierte sich während des Zweiten Weltkriegs in Grossbritannien und den USA. Arbeitsfelder waren damals vorrangig im Bereich der Planung und Koordination von Militäreinsätzen angesiedelt. Heute überwiegen jedoch ökonomische Anwendungen. Einsatzfelder des OR, bei denen Optimierungsprobleme eine Rolle spielen, liegen beispielsweise in der Raum- und Standortplanung, bei Fragen der Lagerhaltung und dem Maschineneinsatz in der Produktionswirtschaft sowie im logistischen Bereich der Absatzwirtschaft.

2.2 Mathematische Definition eines Optimierungsproblems

Bei der mathematischen Definition von Optimierungsproblemen wird von der *Aufgabenstellung*, also der Spezifikation eines praktischen Optimierungsproblems, abstrahiert. An die Stelle dieser Aufgabenstellung tritt direkt eine zu optimierende *Zielfunktion*. Bei Minimierungsproblemen besteht die Aufgabe darin, *Variablen* der Zielfunktion so zu belegen, dass diese ein *globales Minimum* annimmt – analog soll bei einem *Maximierungsproblem* ein *globales Maximum* gefunden werden. Wiederum muss bei der Definition des Begriffs „Optimierungsproblem“ sorgfältig unterschieden werden zwischen konkreten *Problembeispielen* und dem abstrakten *Problem* selbst:

Definition Problembeispiel (Exemplar, Instanz) eines Optimierungsproblems:
Ein Exemplar (oft auch Instanz genannt) eines Optimierungsproblems ist ein Paar (L, f), wobei L die Grundmenge gültiger Lösungsmöglichkeiten und $f : L \to \mathbb{R}$ eine Abbildung (Zielfunktion, Kostenfunktion) dieser Menge auf die reellen Zahlen ist. Das (Minimierungs-)Problem besteht darin, ein $l \in L$ zu finden, für das gilt:

$$\forall y \in L : f(l) \leq f(y)$$

Ein solches l wird als globales Optimum für ein gegebenes Problemexemplar bezeichnet [PS82].

Definition Optimierungsproblem:
Ein Optimierungsproblem ist eine Menge von Problembeispielen. [PS82]

Sinnvollerweise werden zu einem Optimierungsproblem solche Problembeispiele zusammengefasst, deren Zielfunktionen von der Struktur her ähnlich sind und die sich nur in der Belegung von Konstanten sowie der Anzahl der Variablen unterscheiden.

2.3 Elementare Begriffe aus der Optimierungstheorie

Die Menge F der *Lösungsmöglichkeiten* (kurz auch *Lösungen*[2]) eines Exemplars eines Optimierungsproblems kann als Menge von Vektoren reeller Zahlen aufgefasst werden, die möglicherweise bestimmte Nebenbedingungen – so genannte *Constraints* – erfüllen müssen. Jedes Element eines solchen Vektors wird dann als (belegte) *Entscheidungsvariable* bezeichnet. Im Allgemeinen existieren mehrere Möglichkeiten, ein Optimierungsproblem aus der Praxis und seine Lösungsmöglichkeiten durch Entscheidungsvariablen zu beschreiben.

Eine Lösungsmöglichkeit besteht aus konkreten Belegungen der Entscheidungsvariablen unter Beachtung von Nebenbedingungen. Den Raum, der durch die Entscheidungsvariablen aufgespannt und möglicherweise durch Nebenbedingungen eingegrenzt wird, bezeichnet man als *Lösungsraum*. Normalerweise existiert eine Vielzahl potenzieller Lösungen in diesem Lösungsraum. Mathematisch besteht die Lösung eines Exemplars eines Optimierungsproblems in der Suche nach gültigen Belegungen der Entscheidungsvariablen, deren Kombination hinsichtlich der Zielfunktion ein globales Optimum darstellt.

[2]Der Begriff „Lösung“ ist in diesem Kontext zweideutig. Die Lösung eines Problemexemplars besteht in der erfolgreich abgeschlossenen Suche nach einer *optimalen* unter den *potenziellen* Lösungen des Exemplars.

Beispiel TSP:
Beim TSP für n Städte können die Lösungsmöglichkeiten (mögliche Rundreisen) als n-stellige Vektoren $\langle p_1 ... p_n \rangle$ dargestellt werden. p_1 bis p_n sind die Entscheidungsvariablen des Problems und können mit natürlichen Zahlen von 1 bis n belegt werden. Die Belegung der Variable p_i bezeichnet dabei die Position in der Rundreise, an der die Stadt i besucht wird.[a] Weil jeder Stadt eine verschiedene Positionsnummer zugewiesen werden muss, gilt bei dieser Kodierung als Nebenbedingung, dass die Stellen p_1 bis p_n des Vektors paarweise unterschiedlich belegt sein müssen. Der Lösungsraum L_{TSP} eines TSP lässt sich daher folgendermaßen darstellen:

$$L_{TSP} = \{\langle p_1..p_n \rangle\} \text{ mit } p_i \in \{1, 2, ..., n\} \text{ unter der Nebenbedingung: } p_i \neq p_j \forall i \neq j$$

Beim TSP mit n Städten ist der Lösungsraum also der Raum der Permutationen von n natürlichen Zahlen mit der Mächtigkeit $n!$. Allerdings ist jede mögliche Rundreise in diesem Lösungsraum in beiden Richtungen und mit jeder Stadt als Ausgangspunkt vertreten. Für metrische TSP lässt sich die Anzahl *verschiedener* Rundreisen damit durch $(n-1)!/2$ berechnen [MF00, S.13]. Bereits für ein 20-Städte-TSP ergeben sich so über 6×10^{16} mögliche Rundreisen.

[a] Eine equivalente, häufiger verwendete Art der Kodierung besteht darin, die *Indizes* der Städte in der *Reihenfolge ihres Besuchs* in den Lösungsvektor einzutragen. Hier wurde jedoch bewusst eine Kodierung gewählt, bei der *jede Stadt* immer von *derselben Entscheidungsvariablen* repräsentiert wird. Mit dieser Kodierung eignet sich das TSP besser als Anwendungsbeispiel für das im Zuge dieser Arbeit entwickelte Optimierungsverfahren.

Lösungsmöglichkeiten, die ein Algorithmus im Laufe der Bearbeitung eines Problembeispiels produziert, werden in dieser Arbeit auch als *Lösungsvorschläge* bezeichnet. Die Qualität eines Lösungsvorschlags wird durch die bereits angesprochene Zielfunktion gemessen. Insbesondere bei Evolutionären Algorithmen (4.3) wird die Qualität einer Lösung auch als *„Fitness“* und die Zielfunktion analog als *„Fitness-Funktion“* bezeichnet. Sie weist jeder Lösung einen Wert zu, dabei unterscheidet man Minimierungs- und Maximierungsprobleme.

Beispiel TSP:
Sei eine Rundreise l wie oben durch den Lösungsvektor $l = \langle p_1..p_n \rangle$ kodiert. Sei $d_{i,j}$ die Distanz zwischen Stadt i und Stadt j. $c(l)_{ij}$ messe die Distanzen zwischen Städten i und j, die in der durch den Lösungsvorschlag l definierten Rundreise *direkt hintereinander* besucht werden (und sei 0, wenn Stadt j nicht direkt nach Stadt i besucht wird):

$$c(l)_{ij} = \begin{cases} d_{i,j} & \text{falls } p_i + 1 = p_j \\ d_{i,j} & \text{falls } p_i = n \text{ und } p_j = 0 \\ 0 & \text{sonst.} \end{cases}$$

Die Zielfunktion f(l) errechnet sich dann als Summe dieser Distanzen:

$$f(l) = \sum_{(i,j)} c_{i,j}$$

Die Suche nach einer optimalen Lösung entspricht somit der Suche nach einem Minimum bzw. nach einem Maximum dieser Zielfunktion. Da die Zielfunktion ggf. durch ihr negatives Pendant ersetzt werden kann, kann man auch allgemein von einem Minimierungsproblem ausgehen.

Multikriterielle Probleme

Von *multikriteriellen Problemen* und der *Mehrziel-Optimierung* spricht man, wenn man an Lösungen interessiert ist, die *mehrere* Zielsetzungen gleichzeitig erfüllen. Dabei können sich die einzelnen Ziele auch gegenseitig ausschließen oder zumindest „behindern“, beispielsweise werden beim OR oft Lösungen gesucht, die sowohl hinsichtlich der Kosten für ihre Umsetzung als auch ihres Nutzens möglichst günstig sind.

Eine einfache Möglichkeit der Behandlung multikriterieller Probleme besteht darin, für die Zielsetzungen einzelne Teilziel-Funktionen zu formulieren und diese – möglicherweise mit unterschiedlicher Gewichtung – zu einer gemeinsamen Zielfunktion zu aggregieren. Auf die gemeinsame Zielfunktion („Nutzensumme“) können dann Methoden für monokriterielle Probleme angewendet werden. Eine andere Herangehensweise ist die Betrachtung so genannter *Pareto-optimaler* Lösungen[3]: Eine Lösung f gilt dann als Pareto-optimal, wenn sie von keiner anderen Lösung $\bar{f}$ *dominiert* wird. Eine Lösung $\bar{f}$ dominiert eine Lösung f, wenn $\bar{f}$ mindestens eine Zielsetzung besser erfüllt als f und bezüglich der anderen Zielsetzungen nicht schlechter abschneidet als f. Ergebnis einer Optimierung ist hier eine *Menge* Pareto-optimaler Lösungen (auch *Pareto-Front* genannt), aus der in der Praxis dann anhand zusätzlicher Informationen eine einzelne ausgewählt werden kann (vgl. z.B. [FF93], [FF95]).

Constraint-Satisfaction-Probleme

Eine Unterklasse der Optimierungsprobleme stellen die *Constraint-Satisfaction-Probleme (CSP)* dar. Hier sucht man ebenfalls nach einer *gültigen* Lösung, die Zielfunktion ist dagegen konstant und die Qualität aller gültigen Lösungen daher gleich. Für CSP existieren spezielle Bearbeitungsverfahren. Die Suche nach einer gültigen Lösung kann jedoch auch als Minimierungsproblem angesehen werden, bei dem es gilt, die Anzahl verletzter Nebenbedingungen zu minimieren.

2.4 Komplexität von Optimierungsproblemen

Unterschiedliche Exemplare eines Optimierungsproblems können sich in der Anzahl der Entscheidungsvariablen unterscheiden. Die Anzahl der Entscheidungsvariablen wird auch als *Problemgröße* bezeichnet. Bei der Messung der *Komplexität* eines Optimierungsproblems betrachtet man die Anzahl der zur *exakten* Lösung des Problems notwendigen elementaren Rechenschritte („Rechenaufwand“) und bringt diese ins Verhältnis zur Problemgröße.

Als einfach gelten Probleme, für deren Lösung (deterministische) Algorithmen existieren, deren Rechenaufwand höchstens polynomiell abhängig ist von der Problemgröße *(P-Probleme)*. Lineare Optimierungsprobleme (Abschnitt 2.5) sind solche P-Probleme. Als schwierig gelten dagegen Optimierungsprobleme, zu deren Lösung Rechenaufwand benötigt wird, der nicht mehr in ein

[3]Benannt nach dem italienischen Ingenieur, Ökonom und Soziologen Vilfried Pareto (1848-1923)

polynomielles Verhältnis zur Problemgröße gebracht werden kann. Das ist im Allgemeinen für nichtlineare Optimierungsprobleme der Fall.

Eine spezielle Komplexitätsklasse bilden die nichtdeterministisch-polynomiellen Probleme *(NP-Probleme)*. Hier können mit jeweils polynomiell von der Problemgröße abhängigem Zeitaufwand Lösungen erstellt, auf Gültigkeit überprüft und hinsichtlich ihrer Güte verglichen werden. Es können also nichtdeterministische Algorithmen entwickelt werden, die mit polynomiellem Zeitaufwand eine potenzielle Lösung „raten“ dabei jedoch nicht zuverlässig das tatsächliche Optimum finden.

Der Beweis steht zwar bisher noch aus, man vermutet jedoch, dass die Klasse der NP-Probleme eine echte Obermenge der P-Probleme darstellt. Bei dieser Vermutung spielen die so genannten *NP-vollständigen* Probleme eine wichtige Rolle. Diese Probleme liegen in NP, sind (in polynomieller Zeit) aufeinander reduzierbar[4] und es ist bisher kein Algorithmus bekannt, der ein solches Problem in polynomieller Zeit exakt löst. Man vermutet daher, dass diese Probleme zwar in NP, nicht jedoch in P liegen und damit $P \neq NP$ gilt. Viele kombinatorische Optimierungsprobleme (siehe Abschnitt 2.5, S.16) sind NP-vollständig, darunter beispielsweise das TSP.

Auch wenn bezüglich ihrer Komplexität alle NP-vollständigen Probleme „gleichschwer zu lösen“ sind, können für einige unter ihnen schnellere und bessere *Approximationsalgorithmen* gefunden werden, als für andere: Während beispielsweise für metrische TSP mit dem *Christofides-Algorithmus* ein polynomieller Algorithmus existiert, der die Länge der kürzesten Rundreise auf den Faktor 3/2 approximiert, ist für allgemeine TSP der Größe n und ein beliebiges Polynom p(n) bereits das Problem NP-schwer, eine Tour zu finden, die $2^{p(n)}$ mal so lang ist, wie die optimale Tour [MAK07, S.14].

2.5 Lineare, Nichtlineare und Kombinatorische Optimierungsprobleme

Optimierungsprobleme können nicht nur hinsichtlich ihrer Komplexität, sondern auch anhand der Beschaffenheit der Zielfunktion f und der Nebenbedingungen klassifiziert werden.

Lineare Probleme

Sind alle Nebenbedingungen sowie die Zielfunktion eines Optimierungsproblems linear, d.h. Zielfunktion und Nebenbedingungen können als Summe von Termen geschrieben werden, in denen kein Produkt von Entscheidungsvariablen vorkommt, so spricht man von einem *linearen Optimierungsproblem* oder in Anlehnung an die entsprechende Lösungsmethode von *Linearer Programmierung (LP)*. Lineare Probleme mit reellwertigen Variablen sind P-Probleme, ihre Exemplare können also mit polynomiell von der Problemgröße anhängigem Zeitaufwand gelöst werden. Exemplare bei denen nur ganzzahlige Belegungen zugelassen werden *(Integer Programmierung, IP)*, liegen dagegen in NP.

[4] Sind zwei Probleme P und Q aufeinander Polynomialzeit-reduzierbar, so kann man Algorithmen finden, die Lösungen des Problems P in polynomieller Zeit in Lösungen des Problems Q transformieren und umgekehrt.

Nichtlineare Probleme

Sobald die Zielfunktion nicht ohne Produkte von Entscheidungsvariablen geschrieben werden kann, spricht man von *Nichtlinearer Programmierung (NLP)*. Exemplare nichtlinearer Probleme liegen im Allgemeinen in NP. Die Stärke der Nichtlinearitäten unter den Entscheidungsvariablen wird als *Epistasie* bezeichnet (vgl. 4.4.2).

Kombinatorische Probleme

Die Aufgabenstellung so genannter *kombinatorischer Optimierungsprobleme (KOP)* besteht in der Suche nach der bestmöglichen Anordnung, Zuordnung, Gruppierung oder Auswahl einer (üblicherweise endlichen) Menge diskreter Objekte [Law01, S.1]. Solche Probleme treten häufig in Planungsbereichen, z.B. in der Wirtschaft auf und sind ein wichtiges Arbeitsfeld der Operations Research (Abschnitt 2.1). Beispiele für KOP im Bereich des OR sind Lagerhaltungsprobleme, Standort-Einzugsbereich-Probleme (Kapitel 3.2.2), und Reihenfolgeprobleme wie das TSP oder das VRP.

Mathematisch versteht man unter einem KOP eine Menge von Problemexemplaren mit jeweils endlichen Wertemengen für alle Entscheidungsvariablen. An die Zielfunktion und an die Nebenbedingungen werden keine zusätzlichen Anforderungen gestellt. KOP können also lineare und nichtlineare Probleme umfassen. Aufgrund der Endlichkeit der Menge der Entscheidungsvariablen ist bei KOP auch die Grundmenge potenzieller Lösungen endlich.[5] Theoretisch ist es also möglich, die bestmögliche Lösung durch Kombination *aller möglichen Werte* für *alle Entscheidungsvariablen* zu ermitteln. Aus Zeitgründen ist dies jedoch im Allgemeinen nicht durchführbar, deshalb wird meist auf Heuristiken zurückgegriffen, die in annehmbarer Zeit möglichst gute Lösungen produzieren (sollen). Die Effizienz solcher Heuristiken kann gesteigert werden, wenn bestimmte Eigenschaften des Problem(exemplar)s bekannt sind und ausgenutzt werden.

Auch das im Zuge dieser Arbeit entwickelte Optimierungsverfahren für KOP soll bestimmtes Anwenderwissen über die zu bearbeitenden Probleme ausnutzen – konkret das Wissen über die Bewertbarkeit von Teillösungen und die Situations-spezifische Einsetzbarkeit von Verbesserungsoperatoren. Daher umfasst der Anwendungsbereich des Verfahrens speziell die Probleme, bei denen solches Wissen vorhanden ist.

[5] Teilweise werden auch Optimierungsprobleme mit *abzählbar unendlich* vielen Lösungsmöglichkeiten als KOP bezeichnet, so z.B. in [MAK07].

Kapitel 3

Anwendungsbereich und Anwendungsbeispiele

Ziel der Arbeit ist die Entwicklung eines Optimierungsverfahrens, das universelle und problemspezifische Ansätze zur Lösung von Optimierungsproblemen verknüpft und die Integration von Anwenderwissen erlaubt. Bevor die dem Verfahren zugrunde liegenden Ansätze diskutiert werden, soll in diesem Kapitel zunächst der Anwendungsbereich festgelegt (Abschnitt 3.1) sowie die Anwendungsbeispiele eingeführt werden (Abschnitte 3.2 bis 3.4).

Neben dem TSP, anhand dessen das Vorgehensmodell für die Anpassung des Framework auf Optimierungsprobleme demonstriert wird, werden drei Optimierungsprobleme aus der Praxis als Anwendungsfälle ausgewählt, die aus laufenden Projekten des Autors stammen. Es handelt sich dabei zunächst um zwei Anwendungsbeispiele aus den Geowissenschaften: Ein Standortplanungsproblem aus der Wirtschaftsgeographie (3.2) und ein Fahrplan-Optimierungsproblem aus der Verkehrsgeographie (3.3). Ein weiteres Anwendungsbeispiel, die Zuteilung zur Klinischen Rotation (3.4), stammt aus dem universitären Bereich.

Das Kapitel schließt mit einer Zusammenfassung der Ausgangslage und der Anforderungen (3.5) für das zu entwickelnde Optimierungsverfahren.

3.1 Anwendungsbereich des entwickelten Optimierungsverfahrens

Der Anwendungsbereich des entwickelten Verfahrens erstreckt sich über kombinatorische Optimierungsprobleme aus der Praxis, bei denen der Anwender über *eingeschränkte Kenntnisse* verfügt. Ziel ist es, dieses Wissen in das Verfahren zu integrieren und zur Steigerung der Performanz auszunutzen.

Abbildung 3.1 verdeutlicht den Einsatzbereich des Verfahrens: Er wird einerseits begrenzt durch Fälle, in denen mangels integrierbarem Anwenderwissen keine Verbesserung gegenüber universellen Verfahren zu erwarten sind und auf der anderen Seite durch Fälle, in denen sich die problemspezifische Anpassung eines Framework aufgrund der Kenntnis ausreichend guter problemspezifischer Heuristiken erübrigt.

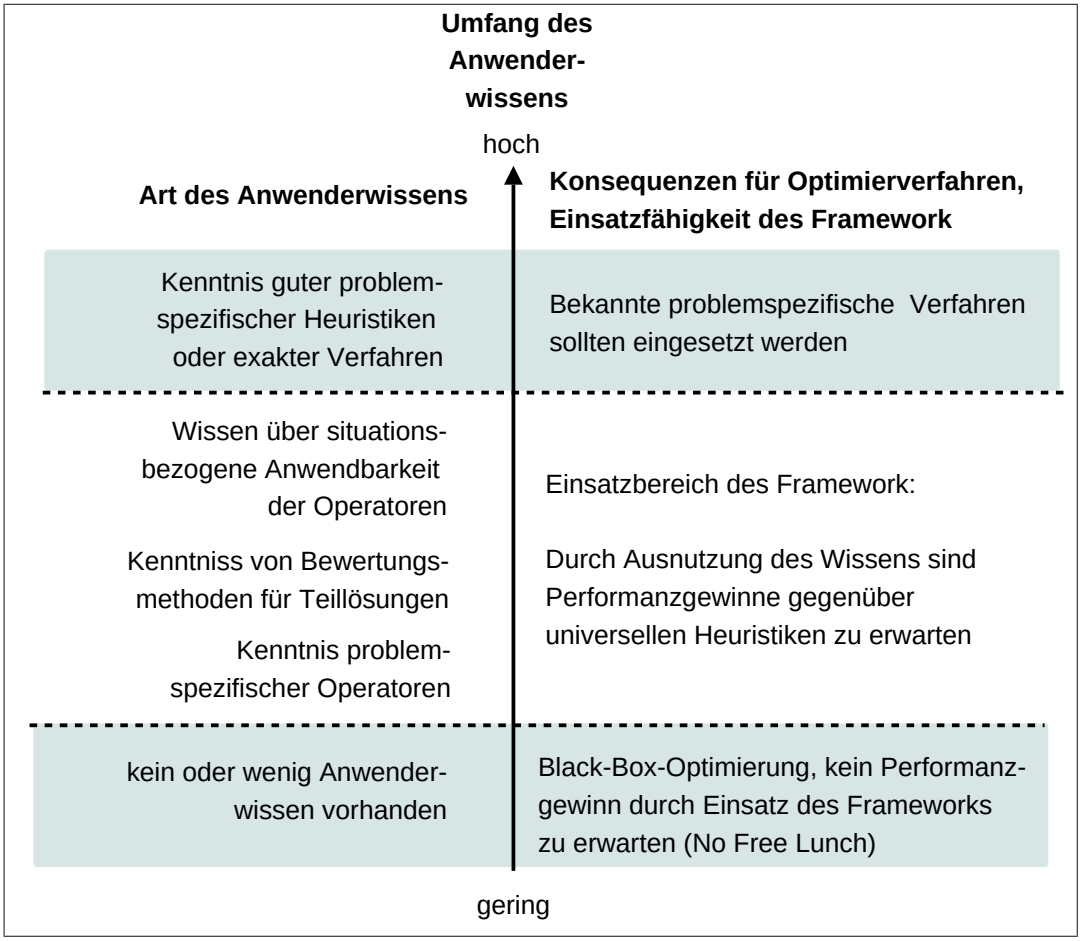


Abbildung 3.1: *Umfang des Anwenderwissens und Einsatzbereich des Framework*

Steht kein Anwenderwissen zur Verfügung (so genannte „*Black-Box-Optimierung*“), so muss mit einer universellen Heuristik (oder vollständiger Enumeration) optimiert werden. Mittelt man über alle theoretisch möglichen Problemexemplare, so sind bei der Bearbeitung mit einem „neuen“

Verfahren keine Effizienzsteigerungen zu erwarten, da kein Wissen ausgenutzt werden kann (vgl. *No-Free-Lunch-Theoreme*, Abschnitt 4.5). Dieser Fall ist daher für die Arbeit nicht relevant.

Ebenso nicht zum Einsatzbereich des Verfahrens zählen Probleme, bei denen dem Anwender gute problemspezifische Heuristiken oder sogar exakte Verfahren bekannt sind, die in angemessener Zeit annehmbare Ergebnisse produzieren, sowie Fälle, in denen der Anwender über die notwendigen Kenntnisse und Ressourcen verfügt, ein speziell zugeschnittenes Verfahren zu entwickeln oder entwickeln zu lassen. In diesen Fällen erübrigt sich die Anpassung eines Framework.

Im Fokus der Arbeit stehen dagegen KOP aus der Praxis, bei denen der Anwender über Problemwissen *in eingeschränktem Umfang* verfügt. Dazu gehört zum Beispiel die Kenntnis bestimmter *Bewertungsmethoden für Teillösungen* sowie *problemspezifischer Operatoren zur Variation von Lösungsvorschlägen*. Mit solchem Wissen ist der Anwender in der Lage, Exemplare kleinerer Größe „von Hand“ zu optimieren oder zumindest relativ gute Lösungen zu erstellen. Diese Situation ergab sich auch bei den Projekten, aus denen die Anwendungsfälle für das entwickelte Optimierungsverfahren stammen.

Die Bearbeitung der Probleme von Hand ist jedoch zeitintensiv, so dass sie sich – jedenfalls ab Problemen gewisser Größe – nicht anbietet, sondern eine automatisierte Bearbeitung mit einem an das konkrete Problem(exemplar) angepassten Verfahren wünschenswert ist. Um Erfahrung und Wissen der Anwender auf einfache und transparente Weise integrieren und ausnutzen zu können, orientiert sich das entwickelte Optimierungsverfahren bei der automatisierten Erstellung von Lösungsvorschlägen an Methoden, die auch bei der Optimierung von Hand angewendet werden.

3.1.1 Problembearbeitung „von Hand“

Eine Möglichkeit zur Bearbeitung von Problemexemplaren von Hand besteht darin, einen gegebenen oder auf beliebige Weise konstruierten Lösungsvorschlag iterativ durch kleinere Variationen der Belegungen zu verbessern (vgl. dazu auch *Lokale Suche*, Abschnitt 4.2.2). Der Anwender kann dabei etwa wie folgt vorgehen:

1. Der Anwender sucht im Lösungsvorschlag einen *Lösungsteil*, d.h. eine Teilmenge der Belegungen, die noch *Verbesserungspotential* aufzuweisen scheint. Dabei macht er sich sein Wissen über die Bewertbarkeit von Teillösungen zunutze.
2. Er sucht für die zu verbessernden Belegungen einen *passenden (problemspezifischen) Variationsoperator* aus.
3. Er wendet den Operator auf die Belegungen an, mit dem Ziel, einen verbesserten Lösungsvorschlag zu finden.

Dieses Vorgehen soll am Beispiel der Bearbeitung von metrischen TSP[1] von Hand verdeutlicht werden.

[1] Für das TSP existieren zwar viele effiziente Algorithmen auch für sehr große Exemplare – vor allem wenn spezielle Eigenschaften der Exemplare wie die Erfüllung der Dreiecksungleichung vorausgesetzt werden (vgl. 2.1). Das TSP gehört daher eigentlich weder zum Anwendungsbereich des Framework, noch werden in der Praxis Problemexemplare von Hand bearbeitet. Der Anschaulichkeit halber wird hier dennoch das beschriebene Vorgehen anhand der Bearbeitung eines solchen metrischen TSP demonstriert.

Beispiel: Schrittweise Verbesserung eines Lösungsvorschlags bei metrischen TSP

1. Der Anwender sucht eine Stadt, die in der durch den Lösungsvorschlag beschriebenen Rundreise über verhältnismäßig lange Strecken angebunden ist oder deren Anbindung sich mit anderen Strecken überschneidet. Dabei reicht es aus, jeweils *Teile des Lösungsvorschlags* zu betrachten. Der Anwender geht davon aus, dass sich durch Verbesserung der Belegungen eines solchen Lösungsteils mit hoher Wahrscheinlichkeit auch insgesamt ein besserer Vorschlag ergibt.
2. Als problemspezifische Operatoren kommen beispielsweise der Tausch zweier Städte in der Rundreise oder – bei Überschneidungen von Kanten – die Auflösung einer Schleife in Frage. Der Anwender wählt je nach Situation einen solchen Operator aus.
3. Nach Anwendung des Operators ergibt sich – wenigstens zu Beginn des Verfahrens – mit hoher Wahrscheinlichkeit ein besserer Lösungsvorschlag.

Eine weitere Möglichkeit besteht darin, mehrere Lösungsvorschläge zu einem neuen, besseren Lösungsvorschlag zu kombinieren (vgl. *Rekombination*, Abschnitt 4.2.3):

1. Der Anwender extrahiert aus bestehenden Lösungsvorschlägen gute Teillösungen (Teilmengen von Belegungen). Wiederum nutzt er dabei sein Wissen über die Bewertbarkeit von Teillösungen.

2. Er versucht, einen neuen Lösungsvorschlag durch Kombination der extrahierten Teillösungen zu bilden. Lösungsteile sind jedoch im Allgemeinen nicht problemlos miteinander kombinierbar (ohne dass z.B. Nebenbedingungen verletzt werden), außerdem führt die Kombination guter Teillösung natürlich nicht zwangsläufig zu einer guten Gesamtlösung (sonst wäre das Problem sehr einfach). Möglicherweise kann der Anwender aber problemspezifisches Wissen über erlaubte bzw. erfolgversprechende Rekombinationen von Teillösungen nutzen.

Dieses Vorgehen sei wiederum am Beispiel der Bearbeitung eines TSP von Hand verdeutlicht:

Beispiel TSP: Rekombination von Teillösungen metrischer TSP-Exemplare.

1. Zunächst extrahiert der Anwender gute „Teilreisen“ aus bestehenden Lösungen. Eine gute Teilreise könnte aus einer überkreuzungsfreien Verbindung mehrerer paarweise relativ nahe beieinander liegender Städte bestehen.
2. Im Anschluss versucht er, die Teilreisen zu einer neuen Rundreise zusammen zu fügen. Dabei können natürlich Probleme auftreten, wenn Städte in den Teilreisen gar nicht oder mehrfach besucht werden. Außerdem können Teilreisen nicht beliebig kombiniert werden, wenn eine gute Gesamtlösung resultieren soll. Der Anwender könnte jedoch das Wissen ausnutzen, dass die Teilreisen ohne „Qualitätsverlust“ auch in umgekehrter Reihenfolge verwendet werden können.

3.1.2 Voraussetzbares Anwenderwissen

Der Anwendungsbereich des entwickelten Verfahrens wird auf KOP eingegrenzt, für deren Bearbeitung folgendes (zusätzliches) Anwenderwissen vorhanden ist und ausgenutzt werden kann:

Bewertungsmethoden für Teillösungen: In der Praxis kann der Anwender nicht nur die Güte eines Lösungsvorschlags insgesamt anhand der Zielfunktion messen, er kennt zusätzlich *Bewertungsmethoden für Teillösungen*. Er kann also nicht nur komplette Lösungsvorschläge evaluieren, sondern auch *partielle Bewertungen* durchführen, d.h. die Güte einer *Teillösung* auf Grundlage einer *Teilmenge* der Belegungen evaluieren. So ist er in der Praxis meist dazu in der Lage, lokales Verbesserungspotential eines Lösungsvorschlags abzuschätzen, also Variablen zu identifizieren, die (noch) ungünstig belegt sind. Voraussetzung dafür ist, dass das zu bearbeitende Problem(exemplar) eine partielle Evaluation überhaupt zulässt. Dabei ist die Epistasie, d.h. der wechselseitige Einfluss von Entscheidungsvariablen auf die Zielfunktion von entscheidender Bedeutung (vgl. Abschnitt 4.4.2): Sinnvolle partielle Bewertungen können nur abgegeben werden, wenn die Epistasie nicht zu groß ist.

Problemspezifische Operatoren: Meist kennt der Anwender einfache aber erfolgversprechende *problemspezifische Operatoren* zur Variation und Kombination von Vorschlägen. Es bietet sich dann auch bei der automatisierten Bearbeitung an, solche problemspezifischen Operatoren anstelle von (oder zusätzlich zu) randomisierten, universellen Operatoren zu verwenden.

Situationsspezifische Operatorauswahl: Kennt der Anwender *mehrere* problemspezifische Operatoren, so kann er meist auch spezifizieren, in welchen Situationen welche Operatoren zur Anwendung kommen sollten. Dieses *Wissen über die Situations-spezifische Operatorauswahl* sollte ebenfalls ausgenuzt werden können.

3.1.3 KOP mit Raumbezug und die partielle Evaluierbarkeit

Gegenstand dieser Arbeit sollen vor allem *reale kombinatorische Optimierungsprobleme mit Raumbezug* sein. Darunter sollen KOP fallen, bei denen die Objekte, die angeordnet, zugeordnet, gruppiert oder ausgewählt werden sollen, im Raum verortet sind, und bei denen diese Verortung der Objekte die Güte von Lösungen entscheidend beeinflusst oder sogar selbst Gegenstand der Optimierung ist. Grund für die Konzentration auf solche Probleme ist, dass sie einerseits – bei intuitiver Kodierung – in der Regel die genannte Eigenschaft der *partiellen Bewertbarkeit* aufweisen, die mit dem Verfahren effizienzsteigernd ausgenutzt werden soll. Unter einer *intuitiven Kodierung* eines KOP soll dabei eine Kodierung mit beliebigen Datentypen verstanden werden, bei der jede Entscheidungsvariable *genau einem* der Objekte zugeordnet werden kann.[2] Andererseits sind Probleme mit Raumbezug meist sehr anschaulich und bieten sich daher als Anwendungsfälle für das Verfahren in besonderer Weise an. Daher wurden mit der Standortplanung (Abschnitt 3.2) und der Fahrplanoptimierung (3.3) zwei KOP mit Raumbezug als Anwendungsfälle gewählt. In der Praxis werden solche Probleme beispielsweise von Geowissenschaftlern bearbeitet.

[2]Die Kodierung eines praktischen Optimierungsproblems kann auf unterschiedliche Weise erfolgen. Bei einer *intuitiven Kodierung* übernimmt jede Entscheidungsvariable die Konfiguration genau eines Objekts (oder trägt zur Konfiguration genau eines Objekts bei). Es findet also keine Transformation der Variablen statt mit dem Ziel, deren Abhängigkeiten im Bezug auf die Zielfunktion zu verringern (vgl. Abschnitt A.6).

Sofern es sich nicht um Standardprobleme handelt, existieren meist keine problemspezifischen Verfahren für solche praktischen Probleme. Der Anwender ist daher auf universelle Verfahren, die Anpassung eines entsprechenden Framework oder die Optimierung von Hand angewiesen. Für viele Exemplare von KOP mit Raumbezug sind von Hand in relativ kurzer Zeit recht gute (jedoch selten optimale) Lösungen erstellbar, obwohl der Lösungsraum bereits bei kleinen Problemexemplaren sehr groß ist. Die Grundlage dazu bildet sicherlich die *Visualisierbarkeit* unterschiedlicher Lösungsvorschläge – wenigstens für Exemplare kleinerer Größe. Die wichtigste Eigenschaft für die aussichtsreiche Bearbeitung von KOP mit Raumbezug „von Hand“ liegt jedoch im Zusammenhang zwischen der *räumlichen Nähe der Objekte* und der *Stärke des wechselseitigen Einflusses ihnen zugeordneter Entscheidungsvariablen* auf die Zielfunktion und der daraus resultierenden partiellen Bewertbarkeit von Lösungsvorschlägen: Im Bezug auf die Zielfunktion ist vorrangig die Kombination der Belegungen von Variablen *benachbarter* Objekte entscheidend. Oft ist eine partielle Evaluation von *Teilgebieten*, d.h. auf Basis der Variablen benachbarter Objekte möglich, die auch die Abschätzung von Verbesserungspotential durch Neukonfiguration von Objekten in bestimmten Teilgebieten zulässt.[3] Dieses a-priori-Wissen kann auch bei der automatisierten Bearbeitung von Problemexemplaren herangezogen werden, um die Performanz des Verfahrens gegenüber universellen Heuristiken (die kein Anwenderwissen nutzen) zu steigern.

Für die erfolgversprechende Bearbeitung mit dem Framework ist der Raumbezug der Optimierungsprobleme zwar *vorteilhaft*, im Unterschied zur partiellen Bewertbarkeit jedoch nicht unbedingt *notwendig*. Neben den beiden KOP mit Raumbezug wird daher noch ein partiell bewertbares KOP ohne Raumbezug als letztes Anwendungsbeispiel herangezogen.

Um die Eigenschaft der partiellen Bewertbarkeit zu verdeutlichen, sei das *einfache Rucksackproblem (engl. Simple Knapsack Problem, SKP)* als Beipiel für ein KOP genannt, das diese Eigenschaft *nicht* aufweist:

Das Simple Knapsack Problem (SKP) (Einfaches Rucksackproblem)
Aus einer Menge von n Objekten mit den Gewichten w_1 bis w_n soll eine Teilmenge mit maximalem Gesamtgewicht zur Mitnahme in einem Rucksack ausgewählt werden, wobei eine vorgegebene Kapazität b des Rucksacks nicht überschritten werden darf (vgl. [Hro03, S.109]).

Das einfache Rucksackproblem SKP kann mit binären Entscheidungsvariablen $e_1..e_n$ kodiert werden, die jeweils über die Mitnahme eines einzelnen Objekts i entscheiden. Anders als etwa beim TSP können hier kaum sinnvolle Prognosen über die Güte einer Teillösung, d.h. die Belegung einzelner Variablen oder kleiner Teilmengen gemacht werden.[4]

SKP ist NP-schwer und ein Spezialfall des *allgemeinen Rucksackproblems* (*General Knapsack Problem KP*). Neben den Gewichten sind den Objekten beim KP Nutzenwerte $n_1..n_n$ zugeordnet, die den Gewichten nicht entsprechen müssen.[5] Ziel ist es, für den Rucksack eine Teilmenge von Objekten mit maximalem *Gesamtnutzen* auszuwählen, wobei deren Gesamtgewicht die Kapazität des Rucksack b weiterhin nicht überschreiten darf.

[3] Anstatt der reinen „Luftlinien-Distanz“ kann auch eine andere auf der Verortung der Objekte beruhende Relation wie z.B. Fahr- oder Sendezeiten zwischen den Objekten verwendet werden.

[4] Die Zielfunktion ist zwar linear (Summe aller Gewichte), Wechselwirkungen zwischen den Entscheidungsvariablen ergeben sich jedoch durch die Nebenbedingung der Maximalkapazität des Rucksacks.

[5] Das SKP entsteht aus dem KP, in dem die Nutzenwerte jeweils den Gewichten gleichgesetzt werden.

3.2 Anwendungsbeispiel I: Standortplanung bei WIGeoGIS

Der erste Anwendungsfall für das entwickelte Optimierungsverfahren ergab sich im Rahmen eines Beratungsprojekts der Firma WIGeoGIS für die Bosch und Siemens Hausgeräte GmbH (BSH), einem großen deutschen Hersteller von Haushaltsgeräten. WIGeoGIS ist ein europaweit führendes Unternehmen im Bereich Geomarketing und Internet / Mobile GIS. Kunden der Firma sind unter anderem Banken, Versicherungen, Telekommunikationsfirmen, Callcenter und Energieversorger. Die Firma berät Unternehmen unter anderem bei der Optimierung ihrer Standorte, Filialnetze und Distribution.

Ziel des Beratungsprojekts war in erster Linie die Optimierung der Distributionslogistik der BSH in Deutschland. Teile des Projekts wurden als Diplomarbeit an einen Studierenden der Geographie an der LMU München vergeben [Buc05]. Dieser sollte die bestehenden Strukturen anhand von Lieferdaten analysieren und mögliche Algorithmen für eine Kosten sparende Standortoptimierung und Reduzierung von Warenumschlagspunkten in Deutschland und Österreich untersuchen. Bei der Mitbetreuung dieser Diplomarbeit durch den Autor entstand die Idee, das im Zuge dieser Arbeit entwickelte Verfahren auch für die Optimierung der Distributionslogistik anzupassen und einzusetzen. Dabei soll die Anpassung zwar zunächst auf die speziellen Anforderungen der BSH zugeschnitten sein, in späteren Varianten dann aber auch bei anderen, ähnlichen Planungsproblemen eingesetzt werden können.

3.2.1 Standortplanung als Optimierungsproblem

Ziel der Standortplanung allgemein ist es, aus einer (nicht notwendigerweise endlichen) Menge potenzieller Standorte einen oder mehrere auszuwählen, die bestimmte Anforderungen am besten erfüllen. Für viele Unternehmen hängt die Konkurrenzfähigkeit und somit der wirtschaftliche Erfolg entscheidend von der Wahl günstiger Standorte ab. Domschke und Drexl [DD96, S.3] betonen die „*Notwendigkeit einer in die Zukunft gerichteten Standortplanung und damit einer Einbeziehung von Standortüberlegungen in die strategische Unternehmensplanung*“.

Domschke und Drexl [DD96] unterscheiden u.a. zwischen *innerbetrieblicher* und *betrieblicher* Standortplanung. Während bei der innerbetrieblichen Standortplanung die räumliche Anordnung von *Betriebsmitteln* (sogenannte *Layoutplanung*) im Vordergrund steht, beschäftigt sich die betriebliche Standortplanung mit Fragen der *Standortwahl* für einzelne Betriebe – in der Logistik zählt dazu beispielsweise die Wahl der Standorte für Zentral-, Beschaffungs- und Auslieferungslager. Dabei treten Optimierungsprobleme auf, deren Lösung in einer möglichst kostengünstigen Wahl der Anzahl und Lage logistischer Knoten („Standorte“) sowie der Zuordnung gegebener „Kunden“ zu diesen Knoten besteht. Sowohl bei der Auswahl der Standorte als auch bei der Zuordnung der Kunden müssen dabei eventuell bestimmte Nebenbedingungen beachtet werden.

3.2.2 Klassifikationskriterien betrieblicher Standortplanungsprobleme

Man unterscheidet verschiedene Formen von Standortplanungsproblemen, und zwar unter anderem anhand folgender Kriterien (vgl. z.B. [DD96], [Dre95]):

Anzahl zu planender Standorte
Ein entscheidendes Kriterium ist die Anzahl der zu planenden Standorte und insbesondere, ob

diese Anzahl im Vorfeld festgelegt ist oder variabel, d.h. selbst Gegenstand der Optimierung.

Lokations- und Allokationsprobleme
Zu den *Lokationsproblemen* gehören *Medianprobleme* und *Zentrumsprobleme*. Bei *Medianproblemen* besteht die Aufgabe darin, Standorte so zu positionieren, dass eine gewichtete Summe von Entfernungen[6] dieser Standorte zu den Kunden minimiert wird. Zur Gewichtung der Kunden wird beispielsweise der Warenbedarf verwendet. Bei *Zentrumsproblemen* soll die maximal auftretende Distanz eines Kunden zu einem der Standorte minimiert werden. *Allokationsprobleme* behandeln dagegen eine möglichst kostengünstige Zuordnung der Kunden zu bereits lokalisierten Standorten, meist unter Berücksichtigung individueller Kapazitäten der Standorte. Von *Lokation-Allokationsproblemen* oder *Standort-Einzugsbereich-Problemen* spricht man, wenn sowohl die optimale Anzahl und Lage der Standorte als auch die optimale Zuordnung der Kunden Gegenstand der Optimierung ist. Da sich beide Teilprobleme gegenseitig beeinflussen, ist ihre Lösung besonders schwierig.

Standortplanung für Distributionsnetze und Filialen
Standortplanungsprobleme spielen einerseits bei der Konzeption und Optimierung von Distributionsnetzen eine wichtige Rolle. Hier werden Waren von Werken – möglicherweise über Umschlagpunkte – zu Kunden geliefert. Aus diesem Anwendungsbereich stammt auch das Anwendungsbeispiel der Standortplanung für die BSH.

Ein anderer Anwendungsbereich ergibt sich bei der Planung von Standorten untereinander konkurrierender Filialen. Hier werden die Waren nicht an die Kunden geliefert, stattdessen gilt es, bestimmte Kundenpotentiale durch günstige Positionierung von Filialen abzuschöpfen. Bei der Optimierung können beispielsweise Gravitationsmodelle zur Anwendung kommen. Wiederum spielen dabei Entfernungen zwischen den Standorten und den Kunden eine wichtige Rolle, da solche Modelle auf der Annahme basieren, dass die Wahrscheinlichkeit für einen Kundenbesuch mit zunehmender Distanz exponentiell abnimmt (vgl. z.B. [Dre95b]). Für eine Zusammenstellung verschiedener Formen der Filialstandortplanung und entsprechende Lösungsansätze sei auf den Teil III von [Dre95] verwiesen.

Diskrete und kontinuierliche Probleme
Sind die in Frage kommenden Standorte (beispielsweise durch Spezifikation einer Liste von Gewerbegebieten) im Vorfeld festgelegt, so handelt es sich um ein *diskretes* Standortplanungsproblem. Solche Probleme können durch Graphen modelliert werden. Für die Kostenberechnungen können im Vorfeld erstellte Entfernungstabellen zum Einsatz kommen. Bei *kontinuierlichen* Problemen ist dagegen jeder Punkt in der Ebene potenzieller Standort. In Distributionsnetzen auftretende Transportkosten können dann nicht im Voraus berechnet werden, stattdessen müssen für die Evaluation eines Lösungsvorschlags zunächst die auftretenden Distanzen zwischen Standorten und Kunden ermittelt werden.

Stufigkeit von Distributionsnetzen
Bei der Planung von Distributionsnetzen unterscheidet man zwischen Netzen mit *direkter* und *indirekter Warenverteilung*. Bei Netzen mit direkter Warenverteilung werden die Waren direkt von den Produktionsstandorten („Werken“) an die Kunden geliefert. Die Lage der Kunden kann nicht beeinflusst werden, daher bezieht sich die Optimierung von Standorten hier auf die Werke. Zwischenlager („*Umschlagpunkte*“) treten dagegen nur bei indirekter Warenverteilung auf. Hier

[6] Statt der Distanzen können auch Transportkosten in die Zielfunktion eingehen, die beispielsweise auf Grundlage der Distanzen berechnet werden.

unterscheidet man *zwei-* und *mehrstufige Netze*. Als zusätzliche *Stufe* bezeichnet man dabei jedes Zwischenlager, das auf dem Weg von der Produktionsstätte (erste Stufe) zum Kunden angelaufen wird. Bei zweistufigen Distributionsnetzen (vgl. Abbildung 3.2) durchläuft also jede Ware genau ein Zwischenlager. Hier können Standortentscheidungen auf Ebene der Werke als auch der Lager getroffen werden.

Die Stufigkeit des Netzes beeinflusst natürlich auch die Komplexität der Berechnung der erwarteten Transportkosten. Bei direkter Warenverteilung müssen die Kunden nur den jeweils „nächstliegenden“ (d.h. den mit geringstmöglichen Kosten erreichbaren) Werken zugeordnet werden, bevor die Berechnung der Kosten durch Aggregation der Kosten für die einzelnen Kunden stattfinden kann. Entsprechen die den Kosten zugrunde gelegten Entfernungen in etwa den Luftlinienentfernungen, ergeben sich *Voronoi-Diagramme* für die „Vertriebsgebiete“ der Werke. Bei indirekter (zweistufiger) Warenverteilung setzen sich die Transportkosten aus primären Kosten (Transport von den Werken zu den Umschlagpunkten) und den sekundären Kosten (Transport von den Umschlagpunkten zu den Kunden) zusammen. Dabei liegt der Transportkostensatz (pro Tonne und Kilometer) für den Sekundärverkehr meist deutlich über dem des Primärverkehrs. Die Bereiche mit den einem Auslieferungslager zugeordneten Kunden sind dann „keulenförmig“: Ein Kunde kann nicht mehr einfach dem „nächstliegenden“ Auslieferungslager zugeordnet werden, da sonst vom Werk über dieses Lager zum Kunden gegebenenfalls unnötige Umwege gefahren werden müssten.

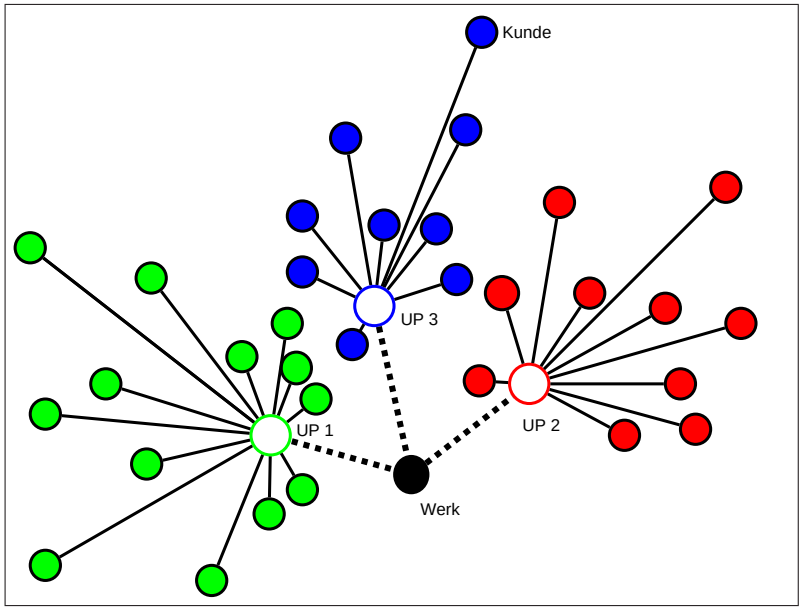


Abbildung 3.2: *Zweistufige Distributionslogistik (UP = Umschlagpunkt)*

Kapazitierung von Umschlagpunkten für Distributionsnetze
Des Weiteren können die Umschlagpunkte *Kapazitätsbeschränkungen* unterliegen. Man spricht dann von *kapazitierten* Standortplanungsproblemen und entsprechend von *unkapazitierten* Problemen, wenn keine Beschränkungen vorliegen oder die Beschränkungen vernachlässigbar sind. Während bei ersteren die Zuordnung eines Kunden zum Auslieferungslager nur in Abhängigkeit

der Entfernungen des Kunden zu den Lagern und unabhängig von den Zuordnungen der übrigen Kunden stattfinden kann, müssen die übrigen Zuordnungen im kapazitierten Fall berücksichtigt werden – hier ergibt sich gewissermaßen als untergeordnetes Optimierungsproblem das bereits erwähnte *Allokationsproblem*. Neben der Kapazitierung von Umschlagpunkten kann auch eine maximale Abdeckungsdistanz gegeben sein, ab der ein Kunde nicht mehr einem Umschlagpunkt zugeordnet werden darf.

Transportmodus in Distributionsnetzen
Bei der Berechnung der Transportkosten für Distributionsnetze spielt außerdem der *Modus der Belieferung* eine entscheidende Rolle. Bei *Direkt-Belieferungen* wird immer genau ein Kunde vom Werk bzw. vom Umschlagpunkt aus angefahren, bei *Touren-Belieferungen* werden Auslieferungen zu Touren gebündelt. Die Ermittlung von Touren mit minimalen Kosten für ein gegebenes Distributionsnetz ist dann wiederum ein untergeordnetes Optimierungsproblem – ein so genanntes *Vehicle Routing Problem (VRP)* (vgl. Abschnitt 2.1).

Einbeziehung von Fixkosten für die Standorte
Möglicherweise sollen neben den Transportkosten *Fixkosten* für die Erstellung und den Betrieb der Standorte an bestimmten Orten berücksichtigt werden. Die Berücksichtigung solcher Fixkosten ist dann besonders wichtig, wenn diese Fixkosten für unterschiedliche potenzielle Standorte verschieden sind oder wenn die Anzahl der zu planenden Standorte nicht im Vorfeld festgelegt ist und dann durch das „Auflassen“ von Standorten Fixkosten gespart werden können.

3.2.3 Planungsszenario bei WiGeoGIS

Beim konkreten Anwendungsfall der Standortplanung für die BSH müssen die Umschlagpunkte einer zweistufigen Distributionslogistik geplant werden.

Waren werden in Werken mit festen Standorten produziert und müssen von dort über Umschlagpunkte an Kunden ausgeliefert werden. Die Kunden sind in diesem Fall Einzelhandelsbetriebe, die die Waren weitervertreiben. Sie sind meist relativ gleichmäßig über das Untersuchungsgebiet verteilt, ihre Positionierung ist *nicht* Gegenstand der Optimierung. Jedes Werk erzeugt nur eine festgelegte Kategorie von Produkten, so dass von mehreren Werken aus über eventuell verschiedene Umschlagpunkte zum Kunden geliefert werden muss.

Gegenstand der Optimierung ist die Anzahl und die Lage der Umschlagpunkte: Einerseits sollen die Kosten für den Transport der Waren von den Werken über die Umschlagpunkte zum Kunden minimiert werden. Herangezogen werden soll dazu ein *Transportkostensatz*, der für Primär- und Sekundärverkehr verschieden ist. Andererseits sollen möglichst geringe *Fixkosten* für die Umschlagpunkte und *Strafkosten* für Nichtabdeckung von Kunden und Überschreitung von Kapazitätsbeschränkungen anfallen. Es handelt sich also gewissermaßen um ein multikriterielles Problem (vgl 2.3). Für die zu planenden Umschlagpunkte ist eine Maximalzahl vorgegeben.

Es soll keine simultane Optimierung der Standorte *und* der Lieferwege stattfinden. Stattdessen werden der Einfachheit halber *Pendeltouren* als Transportmodus gewählt, d.h. modellhaft erhält jeder Kunde pro Werk eine *direkte* Lieferung vom Umschlagspunkt. Außerdem erfolgt die Zuordnung der Kunden zu den Umschlagpunkten nur in Abhängigkeit der jeweiligen Entfernungen, Kapazitätsbeschränkungen werden nicht durch alternative Zuordnungen sondern durch *Strafkosten* modelliert. Die Warennnachfrage eines Kunden von einem Werk wird also jeweils über den Umschlagpunkt abgewickelt, für die die Transportkosten (Primär- und Sekundärverkehr) mi-

nimal werden, auch wenn an diesem Umschlagpunkt dadurch Kapazitätsgrenzen überschritten werden. Eine Ausnahme stellt die Überschreitung der maximalen Abdeckungsdistanz dar: Ist die Entfernung eines Kunden zum nächsten Umschlagpunkt zu groß, so wird der Kunde nicht zugeordnet. Stattdessen werden Strafkosten verteilt.

Die Planung der Umschlagpunkte soll sowohl mit vorgegebenen als auch mit beliebigen potenziellen Standorten erfolgen können. Im ersten Fall erfolgt die Berechnung der Distanzen aufgrund des Straßennetzes. Aus Performanzgründen sollten Distanzmatrizen für Primär- und Sekundärverkehr dann bereits vorab erzeugt werden können. Für die potenziellen Standorte können unterschiedliche Fixkosten angegeben werden. Im zweiten Fall wird die Distanz auf Grundlage der (euklidschen) Luftlinien-Distanz approximiert, die mit einem *inflation factor* multipliziert wird, um die Abweichung der Straßen- von den Luftlinienentfernungen zu modellieren. Andere, möglicherweise genauere Schätzmethoden wären denkbar[7], allerdings ergeben sich aufgrund der Modellierung des Sekundärverkehrs als Pendeltouren ohnehin gewisse Ungenauigkeiten, so dass mit diesem einfachen Modell gearbeitet werden kann.

In der Praxis wird eine Distributionslogistik selten von Grund auf neu geplant. In den überwiegenden Fällen geht es darum, in bestehenden Strukturen lokale Optimierungen von Umschlagpunkten z.B. eine Zusammenlegung zweier Standorte unter Beibehaltung umliegender Standorte durchzuführen. Es muss daher möglich sein, bestimmte Umschlagpunkte zu „fixieren“.

3.2.4 Eigenschaften des Problems und auszunutzendes Anwenderwissen

Ein gegebener Lösungsvorschlag für die Planung von Umschlagpunkten ist nicht nur global durch Evaluation der Zielfunktion bewertbar. Bezüglich ihrer Güte können auch *Teile* eines Lösungsvorschlags relativ unabhängig voneinander untersucht werden, da sich insbesondere Lösungsteile für entfernte Teilräume kaum gegenseitig beeinflussen: Die Güte einer Standortentscheidung ist hauptsächlich von der Lage umliegender Standorte, jedoch kaum von Standortentscheidungen in anderen, entfernten Teilräumen abhängig.

Diese Eigenschaft des Problems kann man sich bei der Optimierung von Hand zunutze machen, da – wenigstens bis zu einem gewissen Maße – ungünstige Lokalisationen von Umschlagpunkten schnell gefunden werden können. Je nach konkretem Problemexemplar besteht in folgenden Situationen beispielsweise meist lokales Verbesserungspotential durch Variation der entsprechenden Belegungen:

- Einem Standort sind sehr wenige Kunden zugeordnet bzw. die über den Standort abgewickelte Warennachfrage ist sehr gering. Liegt die Anzahl der zugeordneten Kunden bzw. die Höhe der abgewickelten Nachfrage weit unter dem Durchschnitt, so ist davon auszugehen, dass der Standort entweder in einem „ungünstigen“ Gebiet (mit geringem Kundenpotential) geplant wurde oder die Warennachfrage potenzieller Kunden über einen benachbarten Umschlagpunkt abgewickelt wird, da die beiden Umschlagpunkte sehr nahe beieinander liegen. Dabei geht man implizit von einer einigermaßen gleichmäßigen Verteilung der Kunden im Untersuchungsgebiet aus (vgl. 3.2.3).

[7] Ein Überblick zu alternativen Möglichkeiten zur Schätzung der (Straßen-)Distanz zweier Orte findet sich beispielsweise in [BL95].

- Einem Standort sind sehr viele Kunden zugeordnet bzw. die über den Umschlagpunkt auszuliefernde Warenmenge ist sehr hoch, bei kapazitierten Problemen entstehen dadurch eventuell sogar Strafkosten. Grund dafür könnte sein, dass sich der Umschlagpunkt in einem Gebiet mit hohem Kundenpotential befindet und die Nachfrage auch relativ weit entfernter Kunden abwickeln muss, da die nächsten anderen Umschlagpunkte weit entfernt sind.
- Bei relativ gleichmäßiger Verteilung der Kunden(potentiale) über das Gesamtgebiet (vgl. 3.2.3) ist die Distanz zwischen zwei Umschlagpunkten sehr gering, beispielsweise deutlich unter der durchschnittlichen Distanz. Die Nachfrage der Kunden, die den beiden Umschlagpunkten zugeordnet sind, ließe sich eventuell auch von einem einzelnen Umschlagpunkt abdecken und der andere Umschlagpunkt könnte in einem anderen Gebiet geplant werden.
- Ein Umschlagpunkt verursacht sehr hohe Fixkosten, die durch Einsparung oder Planung an einem alternativen Standort eingespart oder gemildert werden könnten.
- Es existieren große, zusammenhängende Bereiche nicht abgedeckter Kunden. Strafkosten durch Nichtabdeckung ließen sich verringern, wenn ein Umschlagpunkt in diesem Gebiet geplant würde – indem er beispielsweise aus einem anderen, weniger lukrativen Gebiet verschoben wird.

Für die Verbesserung von Lösungsvorschlägen können folgende einfache Variations- und Rekombinationsoperationen eingesetzt werden:

- Ein Umschlagpunkt wird geringfügig verschoben, so dass zum Beispiel zusätzliche Kunden abgedeckt werden können oder der Umschlagpunkt eine günstigere Lage bezüglich der zugeordneten Kunden erhält.
- Ausgehend von einem Umschlagpunkt werden die Distanzen zu umliegenden Umschlagpunkte durch Verschieben ihrer Standorte vergrößert oder verringert, um die Auslastungen der Standorte anzugleichen und eventuell zusätzliche Kunden beliefern zu können.
- Ein Umschlagpunkt wird aus einem weniger lukrativen Gebiet in ein anderes Gebiet verschoben, in dem sich viele nicht (oder nur über große Entfernungen) abgedeckte Kunden befinden.
- Ein Standort wird komplett aus der Planung genommen, da beispielsweise seine Fixkosten gegenüber der Menge abgedeckter Kunden unverhältnismäßig hoch ist. Analog kann ein Umschlagpunkt hinzugefügt werden, und zwar möglichst an einen kostengünstigen potenziellen Standort in einem Gebiet mit vielen bisher nicht abgedeckten Kunden.
- Zwei (oder mehr) Lösungsvorschläge werden kombiniert, indem je Vorschlag die Standorte der Umschlagpunkte aus einem bestimmten Teilbereich in einen neuen Vorschlag übernommen werden.

Bei der Wahl bzw. Entwicklung eines Optimierungsverfahrens muss beachtet werden, dass die Evaluation von Lösungsvorschlägen relativ zeitaufwändig ist. Nach der Lokalisation der Umschlagpunkte müssen zunächst die Kunden den Umschlagpunkten zugeordnet werden (Allokation), bevor die entstehenden Transport- und Strafkosten berechnet werden können.

3.3 Anwendungsbeispiel II: Die Optimierung Integraler Taktfahrpläne

Als zweites Anwendungsbeispiel wurde die Optimierung Integraler Taktfahrpläne (ITF) gewählt. Neben der Anpassung des entwickelten Optimierungsverfahrens zur Bearbeitung solcher Problemexemplare wird die Integration des resultierenden Moduls in das Expertensystem *OptiTakt* beschrieben. *OptiTakt* ist ein System zur computergestützten Planung und Optimierung symmetrischer Integraler Taktfahrpläne und wurde von Prof. Michael Guckert entwickelt [Guc97]. Das Programm wird seit 2002 vom Autor in Zusammenarbeit mit Herrn Guckert weiterentwickelt und wurde bereits in diversen Projekten zur Verkehrsplanung – insbesondere zur Planung von Fahrplänen im Schienen-Personen-Verkehr (SPV) – eingesetzt, zum Beispiel in Nordbayern und Sachsen.[8] Im Jahr 2006 wurde das im Zuge dieser Arbeit entwickelte Optimierungsverfahren als Modul in *OptiTakt* integriert. So konnten die Methoden zur ITF-Optimierung *von Hand*, die *OptiTakt* bis dahin vorsah, um Funktionen zur *automatisierten* Optimierung ergänzt werden.

3.3.1 Integrale Taktfahrpläne

Ein Integraler Taktfahrplan zielt darauf ab, durch hohe räumliche und zeitliche Verfügbarkeit der Verkehrsmittel im Öffentlichen Personenverkehr (ÖPV) eine dem Individualverkehr vergleichbare Mobilität anzubieten. ITF kommen häufig im SPV, aber auch in anderen regionalen Verkehrsnetzen (z.B. Busverkehren) zum Einsatz.

Definition Integraler Taktfahrplan (ITF)
Ein Integraler Taktfahrplan ist ein *symmetrischer* Fahrplan mit *getaktet verkehrenden* Linien, der *ITF-Knoten (Rendezvouz-Plätze)* aufweist (vgl. [Lie05]).

Die zeitliche Verfügbarkeit wird beim ITF durch *Taktung der Linien* erreicht. Verbindungen werden (wenigstens in der Kernzeit) in *regelmäßigen* Zeitabständen einer Taktzeit t bedient. Für den SPV wird als Taktzeit beispielsweise oft t = 60 Minuten gewählt, Stadtbahnen verkehren oft im 15- oder 20-Minuten-Takt. Beim ITF sind die Taktzeiten der Linien zusätzlich aufeinander abgestimmt, so dass nicht nur die Linien, sondern auch die Anschlüsse „vertaktet“ sind. Im Gegensatz zu Bedarfsverkehren bieten Taktfahrpläne für den Kunden den Vorteil *häufiger* und vor allem *verlässlicher* Abfahrtszeiten, die – bei zweckmäßiger Wahl der Taktung (z.B. $t = 60$ Minuten) - außerdem *leicht merkbar* sind.

Räumliche Verfügbarkeit wird beim ITF durch die *Integration* von Verkehrsträgern auch unterschiedlicher Ebenen (Fern-, Regional-, Stadtverkehr) und die *Verknüpfung* entsprechender Linien in bestimmten *Knotenpunkten*, d.h. Treffpunkten mehrerer Linien gewährleistet. Bei Reisewegen ohne Direktverbindung sollen an den Knotenpunkten möglichst *kurze Umsteigewartezeiten* auftreten. Dazu wird auf *symmetrische* Taktfahrpläne mit speziellen *ITF-Knoten* zurückgegriffen.

[8] Eine Aufstellung einiger Projekte findet sich in [HGH05].

Symmetrie

Umsteigen zwischen verknüpften Linien mit möglichst geringem Zeitverlust soll an den Knotenpunkten jeweils *in beiden Richtungen* möglich sein: (Potenzielle) Fahrgäste des SPV beispielsweise möchten oft Hin- *und* Rückreise absolvieren, hier kann bereits eine lange Umsteigewartezeit in *eine* Richtung abschreckend wirken, so dass die Reise gar nicht gebucht wird. Gleiche Umsteigezeiten für beide Richtungen setzen eine einheitliche *Fahrplansymmetrie* für alle Linien voraus, die die Abfahrtszeit einer Linie an einem Bahnhof in Abhängigkeit der Ankunftszeit der jeweiligen Gegenlinie festlegt: Die Abfahrtszeit einer Linie und die Ankunftszeit der Gegenlinie an einem Bahnhof summieren sich (modulo t) immer zu einem festgelegten Wert. Dieser Wert stellt die *Symmetrieachse* dar, die theoretisch beliebig zwischen 0 und (t-1) gewählt werden kann. In der Praxis wird meist die Minute 0 als Achse gewählt. In einem symmetrischen Taktfahrplan mit Symmetrieache Null und einer Taktzeit t von 60 Minuten ergänzen sich alle Abfahrtszeiten einer Linie mit der jeweiligen Ankunftszeit der Gegenlinie zur Minute Null (modulo 60).

Abbildung 3.3 verdeutlicht diesen Zusammenhang. Dargestellt sind *Netzgrafiken* für Taktfahrpläne mit Taktzeit t = 60. Abfahrtszeiten und Ankunftszeiten von Linien werden in einer Netzgrafik an den Haltepunkten angegeben, die Ankunftszeit steht dabei jeweils näher am Haltepunkt. Links ein Fahrplan mit Symmetrieachse Null, rechts ein asymmetrischer Fahrplan. Während am jeweils rot eingefärbten Haltepunkt links ein Umsteigen in beiden Richtungen mit dem gleichen Zeitverlust (hier 8 Minuten) möglich ist, differieren die Zeitverluste beim Fahrplan rechts (8 bzw. 53 Minuten).

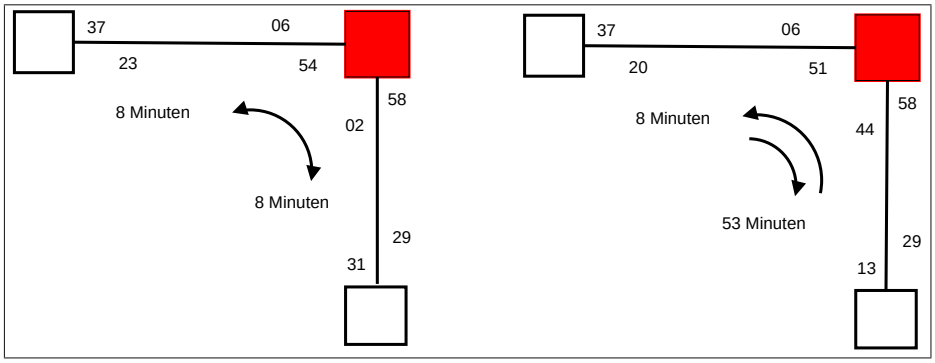


Abbildung 3.3: *Symmetrische und unsymmetrische Taktfahrpläne*

Je nach der Topologie des Verkehrsnetzes und der Fahrzeiten zwischen den Haltepunkten sind symmetrische Fahrpläne bezüglich der Gesamt-Wartezeiten zwar nicht immer optimal (vgl. [Lie04]), dennoch haben sie – neben den genannten Richtungs-neutralen Umsteige-Wartezeiten – technische Vorteile und sind leichter zu handhaben. In der Praxis werden – zumindest in der Hauptverkehrszeit – daher meist symmetrische Fahrpläne verwendet, so zum Beispiel in vielen SPV-Netzen in Deutschland oder nahezu flächendeckend in der Schweiz. Dabei ergeben sich teilweise geringfügige Abweichungen von der Symmetrie aufgrund leicht abweichender Fahrzeiten auf bestimmten Strecken in den unterschiedlichen Richtungen, die beispielsweise Steigungen oder Ausweichstellen auf eingleisigen Abschnitten geschuldet sind.

Rendezvous-Plätze (ITF-Knoten): Nullknoten, Halbknoten und Richtungsknoten

Das wesentliche zusätzliche Charakteristikum eines *Integralen* Taktfahrplans gegenüber einem symmetrischen Taktfahrplan ist die Abstimmung der Taktzeiten aller beteiligten Linien und die dadurch ermöglichte Festlegung von so genannten „*Rendezvous-Plätzen*“ (auch *ITF-Knoten* genannt), bei denen besonders günstige Umsteigebeziehungen vorherrschen. Dabei wird die Tatsache ausgenutzt, dass sich bei einer Taktzeit von t Minuten Linie und Gegenlinie alle $t/2$ Minuten begegnen. Wird als Symmetrieachse die Minute Null und die Taktzeit t = 60 Minuten verwendet, findet also für alle Linien eine Begegnung mit der entsprechenden Gegenlinie zur vollen und zur halben Stunde statt. Besonders günstig ist es dann, wenn sich Haltepunkte genau an diesen Begegnungsstellen befinden, da die Fahrgäste nun aus *beiden* Richtungen (nämlich aus Verkehrsmitteln der Linie *und* der Gegenlinie) etwa gleichzeitig ankommen und *gemeinsam* in Verkehrsmittel weiterer Linien umsteigen können. Einen optimalen ITF-Knoten definiert man daher so, das sich die Linien aus *allen* Richtungen im Knoten treffen, so dass *aus* allen Richtungen *in* alle Richtungen ohne Zeitverlust umgestiegen werden kann.

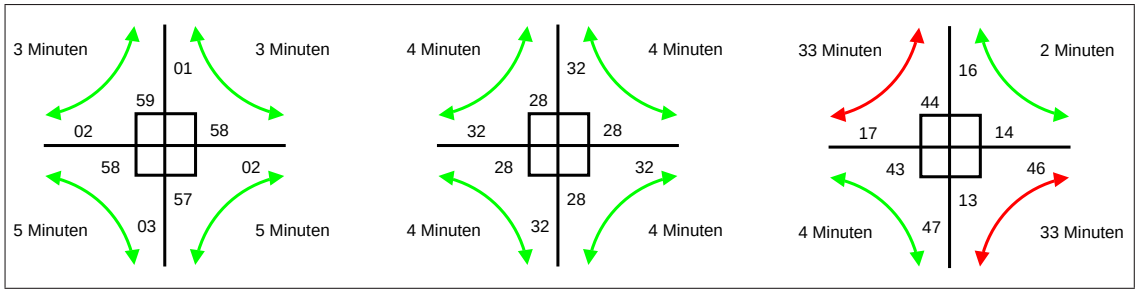


Abbildung 3.4: *ITF-Knoten und Umsteige-Wartezeiten*

Bei *Nullknoten* (engl. *zero hub*) liegt die Symmetrieachse des Taktfahrplans immer zwischen Ankunft und Abfahrt der den Knoten bedienenden Linien. Bei einer Symmetrieachse von Null und Taktzeit 60 fahren am Nullknoten also alle Verkehrsmittel kurz nach der vollen Stunde ab. Im *Halbknoten* (engl. *half hub*) begegnen sich die Linien entsprechend eine halbe Taktzeit nach der Symmetrieachse, bei Symmetrieachse 0 und Taktzeit 60 also jeweils zur halben Stunde [Lie05]. Findet in einem Haltepunkt keine Begegnung der Linien statt, so kann dort nicht in alle Richtungen ohne Zeitverlust umgestiegen werden. Abbildung 3.4 zeigt einen Nullknoten (links) und einen Halbknoten (Mitte) mit günstigen Umsteige-Wartezeiten sowie einen Haltepunkt, in dem sich zwei Linien zwar treffen (rechts), Umsteigen mit geringem Zeitverlust jedoch nur „von Norden nach Osten“ (und umgekehrt) sowie „von Westen nach Süden“ (und umgekehrt) möglich ist. Ein solcher Haltepunkt soll im Folgenden als „Richtungsknoten“ bezeichnet werden, da er wenigstens in bestimmte Richtungen zeitnahes Umsteigen zulässt. Anstelle einer eindeutigen Rendezvouz-Zeit bietet der Richtungsknoten zwei Rendevouz-Zeiten (die sich modulo t zur Symmetrieachse ergänzen).

Die Begegnungszeiten der Linien in einem Knoten sollen im Folgenden als „Knotenzeiten“ bezeichnet werden. Der Nullknoten in Abbildung 3.4 hat also die Knotenzeit „0“, der Halbknoten die Knotenzeit „30“. Für Richtungsknoten werden beide Begegnungszeiten angegeben, der Richtungsknoten in der Abbildung rechts hat die Knotenzeiten „15/45“.

Vor- und Nachteile von ITF

Als Vorteile von ITF sind einerseits natürlich die geringen Umsteigewartezeiten in den ITF-Knoten zu nennen. Für den Kunden bieten sie neben günstigen Umsteigebeziehungen den Vorteil der einfachen Merkbarkeit von Abfahrtszeiten, die im Null- und Halbknoten sogar in alle Richtungen gelten. Außerdem können sich Abfahrtszeiten von Linien anderer Verkehrssysteme ebenfalls an den Knotenzeiten orientieren, so dass ein integraler Taktfahrplan für Verkehrsnetze unterschiedlicher Ebenen entsteht. Als symmetrische Fahrpläne sind sie einfacher zu planen und bieten insbesondere für Fahrgäste, die eine Reise in beide Richtungen zurücklegen, den Vorteil identischer Umsteigewartezeiten für Hin- und Rückreise.

Sinnvolle ITF-Knoten können jedoch nur dann konstruiert werden, wenn die Infrastruktur dies auch zulässt. Dafür ist es zunächst einmal günstig, wenn die Fahrzeit zwischen zwei Haltepunkten überall knapp unter dem Vielfachen einer halben Taktzeit liegt. Allerdings können auch dann Situationen auftreten, in denen eine eindeutige Identifikation von Null- und Halbknoten nicht überall möglich ist. Solche Situationen ergeben sich, wenn das Netz Kreise enthält, bei denen die Summe der jeweils auf die nächste halbe Taktzeit aufgerundeten Fahrzeiten beteiligter Strecken kein Vielfaches der Taktzeit ist (vgl. dazu auch [Lie05]). Abbildung 3.5 zeigt einen sehr einfachen Fall, bei dem ein Haltepunkt weder als Null- noch als Halbknoten sinnvoll definierbar ist. In dieser vereinfachten Netzgrafik sind die Fahrzeiten auf den Strecken an den entsprechenden Kanten angegeben, in den Knoten steht jeweils die Rendezvous-Zeit (Knotenzeit).

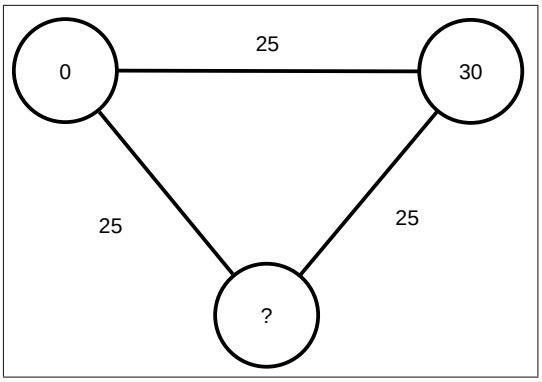


Abbildung 3.5: *Verkehrsnetz, bei dem für einen der Haltepunkte keine sinnvolle Zuweisung einer Knotenzeit möglich ist*

In realen Verkehrsnetzen kommen natürlich auch Fahrzeiten von beispielsweise knapp über dem Vielfachen einer halben Taktzeit vor. Um das Verkehrsnetz dennoch mit einem ITF bedienen zu können, müssen Richtungsknoten und/oder von der Knotenzeit abweichende Ankunfts- und Abfahrtszeiten für bestimmte Linien zugelassen werden. Alternativ kann versucht werden, durch Streckenausbau und/oder die Anschaffung schnellerer Fahrzeuge die Fahrzeit auf bestimmten Linienabschnitten zu verkürzen, um dann Null- oder Halbknoten rechtzeitig erreichen zu können.[9] Dem Nutzen geringer Umsteigezeiten stehen dann Kosten für Infrastrukturmaßnahmen

[9] In diesem Fall richtet sich also mit Fahrzeugen und Strecken gewissermaßen die „Hardware“ nach der „Software“ (dem Fahrplan).

gegenüber. Liebchen [Lie05] führt in diesem Zusammenhang den zusätzlichen Nachteil an, dass dadurch Investitionen sehr langfristige Auswirkungen haben und so unter Umständen Schrittinovationen verhindert werden.

Als weiteren Nachteil von ITF nennt Liebchen die mangelnde Robustheit aufgrund der Anfälligkeit für Folgeverspätungen, die sich durch besonders kurze Umsteigewartezeiten ergeben. Außerdem werden Strecken und Haltepunkte über die Taktzeit hinweg ungleichmäßig ausgelastet. Im Übrigen steht er symmetrischen Fahrplänen insgesamt kritisch gegenüber. Dabei dürfte die Tatsache eine Rolle spielen, dass sich diese nur unzureichend als Periodic Event Scheduling Problem (PESP) beschreiben lassen (vgl. [Lie04]), ein Ansatz zur Modellierung von Fahrplan-Optimierungsproblemen, den er verfolgt.

Bei aller – teilweise sicherlich auch berechtigter – Kritik gelten ITF als besonders kundenfreundlich und werden daher in der Praxis auch zunehmend eingesetzt. Beispiele sind in Deutschland der Allgäu-Schwaben-Takt, der Bayern-Takt und der Rheinland-Pfalz-Takt. In der Schweiz, die in dieser Beziehung als besonders fortschrittlich gilt, wurde der ITF bereits 1982 eingeführt. Das Konzept „Bahn 2000“ der Schweizerischen Bundesbahnen (SBB), das seit 1997 schrittweise umgesetzt wurde und seit 2000 flächendeckend implementiert ist, sieht im Fernverkehr sogar einen landesweiten ITF mit Halbstundentakt vor.[10]

Das Expertensystem *OptiTakt* setzt ebenfalls auf ITF und die Anpassung der Genetischen Konferenz beschränkt sich auf die Optimierung symmetrischer Taktfahrpläne mit ITF-Knoten.

3.3.2 Das Expertensystem *OptiTakt*

OptiTakt ist ein Expertensystem zur Erstellung, Analyse und Optimierung symmetrischer Taktfahrpläne insbesondere für den SPV. Die erste Version (*OptiTakt 1*) bietet zunächst ein CAD-Werkzeug, mit dem die Infrastruktur eines Verkehrsnetzes, d.h. die Haltepunkte und deren Verbindungen festgelegt werden können. Abbildung 3.6 zeigt das CAD-Werkzeug.

Anschließend können Taktfahrpläne definiert werden, indem zunächst auf den Verbindungen verkehrende Linien festgelegt und deren Abfahrtszeiten (modulo der Taktzeit) an den Haltepunkten in spezielle Textfelder eingegeben werden. Dabei verfügt *OptiTakt* über einen automatischen Weitergabemechanismus für die Ankunfts- und Abfahrtszeiten von Linien: Werden einzelne Zeiten geändert, passt *OptiTakt* die übrigen Zeiten der Linie und ihrer Gegenlinie automatisch an.

Für die Fahrpläne stellt *OptiTakt* unterschiedliche Sichten zur Verfügung. Neben einer Netzgrafik-Ansicht können Bildfahrpläne und Kurstabellen erstellt werden. Mit der Version *OptiTakt 2* kamen zwei weitere Sichten hinzu: Eine spezielle Geo-Ansicht zeigt die Haltepunkte gemäß ihrer tatsächlichen Koordinaten vor einer beliebigen, vom Anwender spezifizierbaren georeferenzierten Hintergrundkarte. Optional kann *OptiTakt* aus der Karte über die Farbwerte kodierte Informationen bezüglich der Gewichtung der Haltepunkte extrahieren. Abbildung 3.7 zeigt einige der beschriebenen Ansichten.

Bei der Analyse der Güte erstellter Fahrpläne hilft *OptiTakt* einerseits mittels einer erweiterten Netzgrafik, in der entstehende Umsteige-Verluste berechnet und angezeigt werden. Andererseits können spezielle Kennzahlen für die Qualität des Fahrplans berechnet werden.[11]

[10] Infomationen zur Geschichte des Konzepts „Bahn 2000“ sind in [KS04] zusammengestellt.
[11] Auf diese Kennzahlen wird in Abschnitt 7.3.3 näher eingegangen.

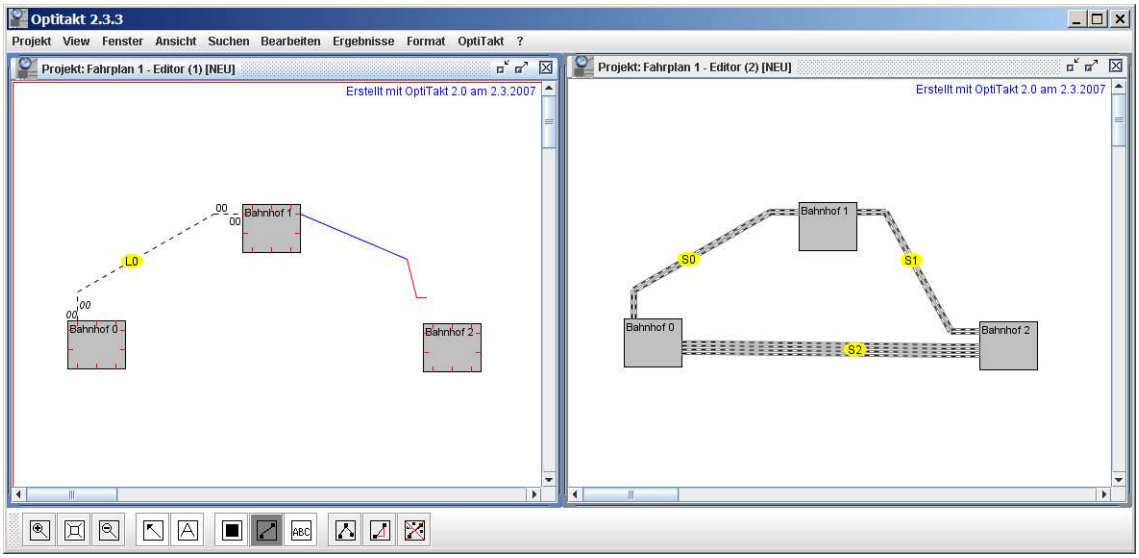


Abbildung 3.6: *CAD-Werkzeug zur Planung der Netzinfrastruktur.*

Bisher fehlten jedoch Möglichkeiten zur *automatisierten* Optimierung von Taktfahrplänen. Dieser Mangel wurde mit der Version 3 behoben, mit der *OptiTakt* um ein entsprechendes Modul erweitert wurde. Dieses Modul basiert auf einer problemspezifischen Implementierung des im Zuge dieser Arbeit entwickelten Optimierungsverfahrens. Das neue Modul hilft dabei insbesondere bei der Grobplanung des Fahrplans, d.h. bei der Zuweisung passender Knotenzeiten zu den Haltepunkten.

3.3.3 Grobplanung von ITF mit dem Shuttle-Modell

Die Optimierung eines Taktfahrplans besteht in der Zuweisung von Abfahrtszeiten zu gegebenen Linien(segmenten), so dass bei gegebenen Fahrgastzahlen insgesamt minimale Reisezeiten resultieren. Dabei können (und sollten) zusätzlich Kosten für die Betriebsmittel berücksichtigt werden. Außerdem müssen ggf. auch die Fahrzeiten auf den Linien reduziert werden, wobei dann weitere Kosten für entsprechende Beschleunigungsmaßnahmen anfallen. Es handelt sich also wiederum um ein multikriterielles Problem (vgl. 2.3).

Es existieren unterschiedliche Ansätze zur Optimierung von Taktfahrplänen, auf einige wird unter 7.3.6 näher eingegangen. Während bei den meisten Ansätzen jedoch von bereits bekannten Linien-Durchbindungen ausgegangen wird, basiert der Ansatz, der hier verfolgt wird, auf einem *Shuttle-Modell*.

Die Idee dabei ist, einen Integralen Taktfahrplan zunächst nicht auf Grundlage von gegebenen *Linien*, sondern nur auf Basis gegebener *Strecken* zu planen. Im Modell verkehrt auf jeder Strecke zwischen zwei Halteorten ein *Shuttle*, eine entsprechende (derzeitige) Mindestfahrzeit sowie die Mindest-Umsteige-Zeit in den Halteorten ist gegeben. Außerdem gegeben ist die Anzahl von Fahrgästen für jedes Paar von Haltepunkten. Ziel ist die *Grobplanung* eines ITF, d.h. die Festlegung von Null- und Halbknoten und der Abfahrtszeiten sowie der Taktfrequenz auf den

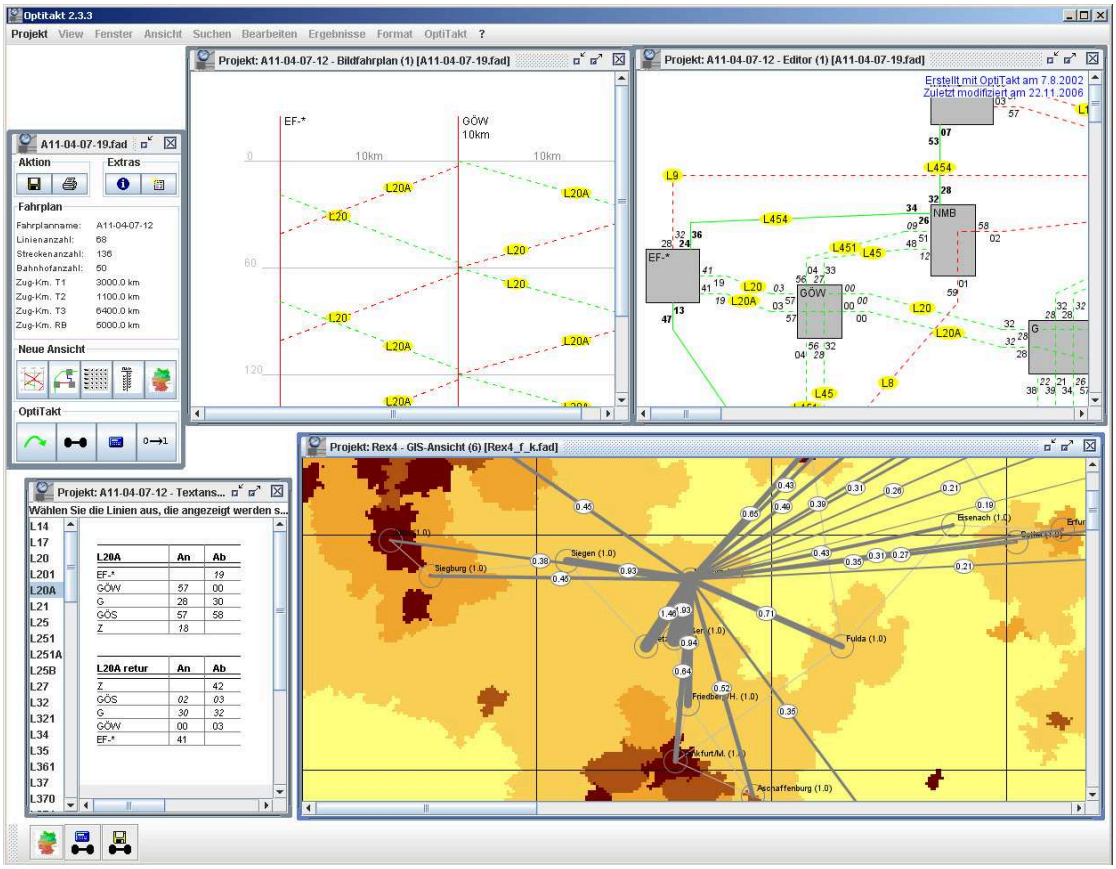


Abbildung 3.7: *Graphische Benutzeroberfläche von OptiTakt mit unterschiedlichen Ansichten eines ITF. Oben links ein Übersichtsfenster, dann im Uhrzeigersinn ein Bildfahrplan, eine Netzgrafik, eine Geo-Ansicht und eine Kurstabelle.*

Strecken, so dass im Gesamtnetz minimale Warteverluste entstehen sowie damit einhergehend die Ermittlung „lohnender“ Ausbauinvestitionen.[12]

Wie die Linien des Netzes später konkret „durchgebunden“ werden, wird dagegen nicht berücksichtigt. Dieser Ansatz hat den Vorteil hoher Flexibilität: Die Identifikation potenzieller ITF-Knoten und das Auffinden sinnvoller Investitionsmaßnahmen wird auch ohne Festlegung auf bestimmte Linien-Durchbindungen ermöglicht.

3.3.4 Eigenschaften des Problems und auszunutzendes Anwenderwissen

Sowohl die Grob- als auch die Feinplanung von Fahrplänen ist prinzipiell auch von Hand durchführbar – wird jedoch insbesondere für größere Netze sehr zeitintensiv.

Bei der Konstruktion von Fahrplänen von Hand nutzt der Planer die partielle Bewertbarkeit

[12] Verluste ergeben sich aus der Differenz von tatsächlicher Zeit zwischen *Ankunft* eines Kunden am Startbahnhof und *Ankunft* am Zielbahnhof und *geringstmöglicher Fahrzeit*, wenn die Reise sofort angetreten werden kann und an den Haltepunkten nur eine Mindesthalte- und Umsteigezeit auftritt.

von Lösungsvorschlägen. Relativ einfach können für gegebene Fahrpläne die Haltepunkte gefunden werden, an denen größere Umsteigewartezeiten entstehen. Das Programm *OptiTakt* bietet beispielsweise entsprechende Funktionen, die den Anwender bei der semi-automatischen Optimierung von ITF unterstützen. Abbildung 3.8 zeigt die automatische Erkennung von Wartezeiten: Für Fahrgäste des Shuttle, der den Haltepunkt X nach „Osten“ verbindet, fallen – zusätzlich zu einer hier festgesetzten Mindest-Transferzeit von 2 Minuten – in beiden Richtungen mindestens 43 Minuten Wartezeit am Haltepunkt X an, wenn Fahrgäste hier von anderen Shuttle aus bzw. in andere Shuttle umsteigen.

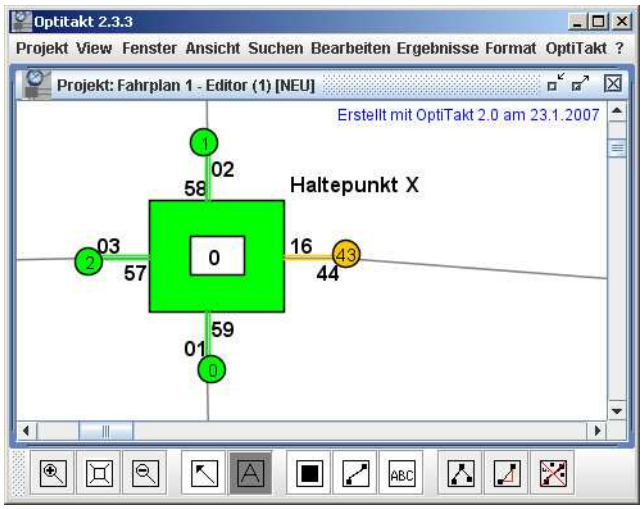


Abbildung 3.8: *Automatische Erkennung von Wartezeiten mit OptiTakt*

Bei einer Optimierung basierend auf dem Shuttle-Modell kann ein Planer zur Verbesserung von Vorschlägen unter anderem folgende lokale Variationsoperatoren verwenden:

- Er passt eine oder mehrere Abfahrtszeiten eines Shuttle an die Knotenzeit eines bedienten Haltepunktes oder an die Ankunftszeit eines jeweils anderen Shuttle an, das den Haltepunkt bedient.
- Für einen Shuttle oder einen Linienabschnitt sieht er eine Beschleunigung vor, damit beispielsweise anliegende Knotenzeiten eingehalten werden können.
- Zur Kombination zweier Vorschläge werden die Netzkonfigurationen aus Teilgebieten bisheriger Vorschläge übernommen.

Der Anwender kann zwar relativ leicht Regionen in einem Fahrplan ausmachen, an denen noch Verbesserungspotential vorhanden ist. Ebenso wie bei der Standortplanung ist jedoch die (exakte) Bewertung von Lösungen sehr rechenintensiv. Für eine gegebene Konfiguration des Fahrplans müssen zunächst die Verkehrsströme berechnet werden, bevor die entstehenden Verlustzeiten ausgewertet werden können, da sich je nach Fahrplan unterschiedliche kürzeste Wege im Netz ergeben können. Ein automatisiertes Optimierungsverfahren sollte also die Anzahl auszuwertender Lösungsvorschläge möglichst gering halten.

3.4 Anwendungsbeispiel III: Die Zuteilung von Praktikumsplätzen der Klinischen Rotation

Beim letzten Anwendungsbeispiel, der Optimierung der Zuteilung von Studierenden zu den Praktika der Klinischen Rotation, handelt es sich um ein Kombinatorisches Optimierungsproblem – genauer ein Zuordnungsproblem – das keinen Raumbezug aufweist. Dennoch können Lösungsvorschläge partiell bewertet werden und problemspezifische Variationsoperatoren sind bekannt, so dass das Problem zum Anwendungsbereich des entwickelten Verfahrens gehört.

3.4.1 Die Klinische Rotation

Die so genannte „Klinische Rotation“ wurde im Sommersemester 2005 in der Tiermedizinischen Fakultät der LMU München eingeführt. Mit der Klinischen Rotation sollte u.a. eine praxisnähere Ausbildung durch stärkere Integration der Studierenden in Kliniken und mehr klinische Unterrichtsstunden realisiert werden.

Das Angebot der Klinischen Rotation besteht aus unterschiedlichen Blockpraktika („Blöcken“) die über ein Jahr hinweg mehrmals regelmäßig stattfinden. Die Praktika unterscheiden sich einerseits inhaltlich, andererseits dauern sie unterschiedlich lang (eine bis sieben Wochen) und haben verschiedene Kapazitäten. Eine Übersicht der angebotenen Praktika findet sich in Tabelle 3.1.

Bezeichnung	**Art**	**Dauer**	**Kapazität**	**Anzahl der Starttermine**
Innere Kleintier	Pflicht	sieben Wochen	40 Plätze	7 Termine (alle sieben Wochen)
Nutztiere	Pflicht	sieben Wochen	40 Plätze	7 Termine (alle sieben Wochen)
Patho, Lebensmittel	Pflicht	sieben Wochen	70 Plätze	4x im Jahr (unregelmäßig)
Chirurgie Kleintier	Pflicht	drei Wochen	18 Plätze	17 Termine (alle drei Wochen)
Pferd	Pflicht	eine Woche	6 Plätze	48x im Jahr (fast jede Woche)
Geflügel	Wahl	drei Wochen	8 Plätze	17 Termine (alle drei Wochen)
Schwein	Wahl	drei Wochen	8 Plätze	17 Termine (alle drei Wochen)
Fische und Reptilien	Wahl	drei Wochen	8 Plätze	17 Termine (alle drei Wochen)

Tabelle 3.1: *Praktikumsangebot der klinischen Rotation*

Jeder Studierende muss im Laufe seines 8. und 9. Fachsemesters genau sechs dieser acht Praktika besuchen. Mit Ausnahme der Wahlpraktika „Geflügel“„Schwein“ und „Fische und Reptilien“ müssen alle Praktika genau einmal belegt werden (Pflichtpraktika). Von den Wahlpraktika muss jeder Studierende dagegen nur genau *eines* besuchen. Insgesamt besuchen die Studierenden damit in 28 Wochen das Jahres Praktika, in den übrigen 24 Wochen haben sie „frei“.

Einige der Studierenden müssen jedoch im Zeitraum der klinischen Rotation zusätzlich andere Praktika (z.B. die begleitend angebotenen Schlachthof- und Hygienepraktika) belegen oder zu bestimmten Prüfungen antreten, für die sie sich im Vorfeld vorbereiten müssen. Dadurch wird der Zeitraum weiter eingeschränkt, in denen sie Rotationsblöcke besuchen können.

Neben der Möglichkeit, unter den drei genannten Wahlpraktika auswählen zu können, sollen außerdem Präferenzen der Studierenden hinsichtlich der zeitlichen Verteilung ihrer Praktika über das Jahr hinweg beachtet werden.

3.4.2 Die Zuteilung der Praktikumsplätze: Ein Zuordnungsproblem

Die Problemstellung bei der Zuteilung der Studenten zu den Rotationsblöcken liegt in der regelkonformen Verteilung der ca. 280 Studierenden auf die Blöcke unter Berücksichtigung der Platzkapazitäten sowie der (zeitlichen und fachlichen) Präferenzen der Studierenden. Unabhängig von der unbedingt notwendigen Einhaltung der Platzkapazitäten sollte die Belegung einzelner Praktika über das Jahr hinweg möglichst gleichmäßig sein. Das Problem kann daher als multikriterielles kombinatorisches Optimierungsproblem mit Nebenbedingungen angesehen werden: Die regelkonforme, überschneidungsfreie Zuteilung unter Berücksichtigung der Kapazitäten ist (unbedingt zu erfüllende) Nebenbedingung (*hard constraint*), eine möglichst weitreichende Berücksichtigung der Präferenzen der Studierenden sowie eine gleichmäßige Auslastung der Praktikumskurse über das Jahr hinweg ist wünschenswert und Gegenstand der Optimierung (*soft constraint*).

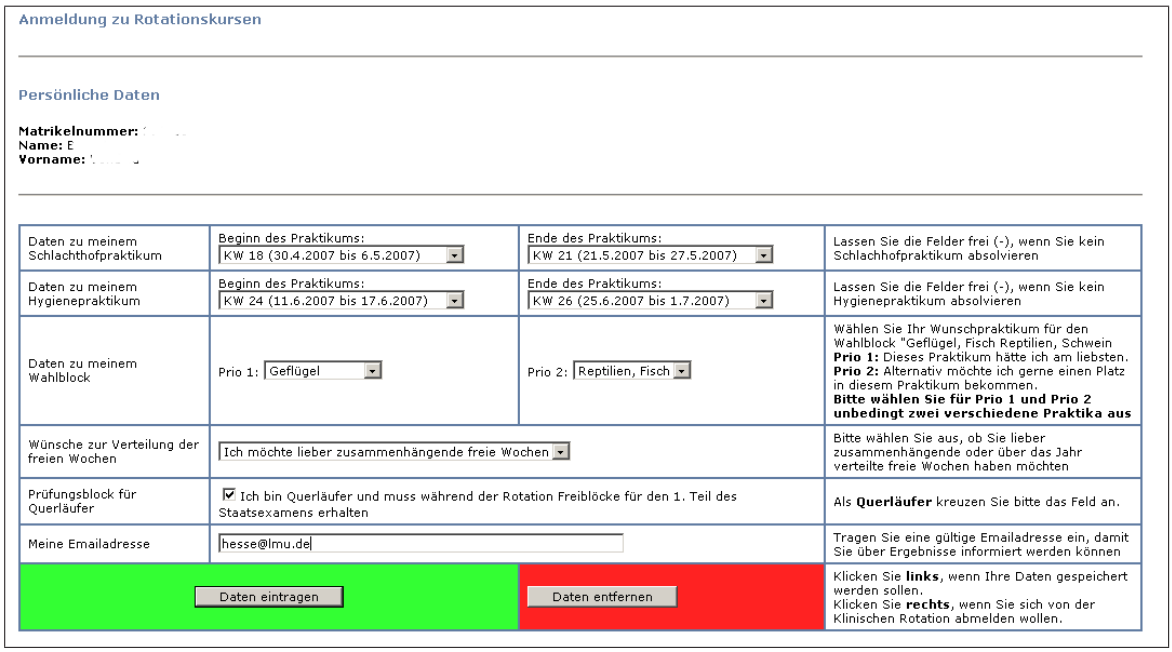
Eine sehr bekannte Gruppe der KOP bilden die so genannten *Stundenplanprobleme*, die beispielsweise bei der Planung der Unterrichtstermine in Bildungseinrichtungen auftreten. Das Problem der Zuteilung der Studierenden zu den Rotationsblöcken ist jedoch kein Stundenplanproblem, wie man aufgrund der Bezeichnung vielleicht assoziieren könnte. Bei letzteren ist die Zuordnung der „Teilnehmer“ zu „Kursen“ gegeben, während eine optimale Zuordnung der Kurse zu Zeitfenstern Gegenstand der Optimierung ist. Im vorliegenden Fall sind dagegen die Termine der Kurse (hier: Praktika) gegeben, gesucht ist eine überschneidungsfreie und Regel-konforme *Zuordnung* der Studierenden zu den Praktika, die deren Präferenzen möglichst beachtet. Es handelt sich daher um ein so genanntes *Zuordnungsproblem*.

3.4.3 Die Zuteilung von Hand

In den letzten Semestern wurde den Studierenden bereits ein halbes Jahr vor dem Start der Rotationsblöcke ein Fragebogen ausgehändigt, auf dem sie angeben konnten, welches der Wahlpraktika sie bevorzugen und ob die freie Zeit eher zusammenhängen oder mit den zu besuchenden Praktika alternieren soll. Außerdem war der Zeitraum der eventuell vorher vom Studierenden individuell vereinbarten Schlacht- und Hygienepraktika anzugeben, um hier Überschneidungen mit Rotationsblöcken auszuschließen. Abbildung 3.9 zeigt das online-Anmeldeformular, das dann bei der Anmeldung zur Rotation 2007/2008 anstatt des Papierfragebogens zum Einsatz kam.

Die Zuteilung der Kurse für die Rotation 2006/2007 wurde im Jahr 2005 noch von Hand auf einer großen Papierrolle vorgenommen, auf der eine Tabelle eingezeichnet war. Jede Zeile entsprach einem Studierenden, jede Spalte einer Kalenderwoche. Kalenderwochen von Studierenden, die bereits anderweitig belegt waren, wurden direkt auf der Rolle eingetragen. Durch Aufbringen von Kärtchen verschiedener Größe wurden dann die Zuteilungen von Studierenden zu Kursplätzen markiert.

Zunächst wurde durch iteratives Hinzufügen von Kärtchen ein hinsichtlich Überschneidungsfreiheit gültiger Ausgangsvorschlag erzeugt, bei dem aufgrund von Kapazitätsbeschränkungen und Überschneidungsproblemen jedoch noch nicht allen Studierenden in allen Bereichen Plätze zugeteilt werden konnten. Auf diesen Ausgangsvorschlag wurde dann im zweiten Schritt folgende „Verbesserungsheuristik“ angewendet: Der Reihe nach wurde für jeden Studierenden überprüft, ob die momentane Belegung bereits einer gültigen Zuteilung entspricht, d.h. ob in allen Bereichen Kurse zugeteilt waren. War dies nicht der Fall, so wurde versucht, eine solche gültige Zuteilung durch Verschieben, Tauschen und Hinzufügen von Kursen zu erreichen (vgl. Abschnitt 3.4.4).



Anmeldung zu Rotationskursen

Persönliche Daten

Matrikelnummer:
Name: E
Vorname:

Daten zu meinem Schlachthofpraktikum	Beginn des Praktikums: KW 18 (30.4.2007 bis 6.5.2007)	Ende des Praktikums: KW 21 (21.5.2007 bis 27.5.2007)	Lassen Sie die Felder frei (-), wenn Sie kein Schlachhofpraktikum absolvieren
Daten zu meinem Hygienepraktikum	Beginn des Praktikums: KW 24 (11.6.2007 bis 17.6.2007)	Ende des Praktikums: KW 26 (25.6.2007 bis 1.7.2007)	Lassen Sie die Felder frei (-), wenn Sie kein Hygienepraktikum absolvieren
Daten zu meinem Wahlblock	Prio 1: Geflügel	Prio 2: Reptilien, Fisch	Wählen Sie Ihr Wunschpraktikum für den Wahlblock "Geflügel, Fisch Reptilien, Schwein **Prio 1:** Dieses Praktikum hätte ich am liebsten. **Prio 2:** Alternativ möchte ich gerne einen Platz in diesem Praktikum bekommen. **Bitte wählen Sie für Prio 1 und Prio 2 unbedingt zwei verschiedene Praktika aus**
Wünsche zur Verteilung der freien Wochen	Ich möchte lieber zusammenhängende freie Wochen		Bitte wählen Sie aus, ob Sie lieber zusammenhängende oder über das Jahr verteilte freie Wochen haben möchten
Prüfungsblock für Querläufer	☑ Ich bin Querläufer und muss während der Rotation Freiblöcke für den 1. Teil des Staatsexamens erhalten		Als **Querläufer** kreuzen Sie bitte das Feld an.
Meine Emailadresse	hesse@lmu.de		Tragen Sie eine gültige Emailadresse ein, damit Sie über Ergebnisse informiert werden können
Daten eintragen		Daten entfernen	Klicken Sie **links**, wenn Ihre Daten gespeichert werden sollen. Klicken Sie **rechts**, wenn Sie sich von der Klinischen Rotation abmelden wollen.

Abbildung 3.9: *Online-Anmeldebogen für die Klinische Rotation 2007/2008. Eingetragen werden können u.a. individuelle Termine für Schlachthof- und Hygienepraktikum, Präferenzen zum Wahlpraktikum sowie Wünsche zur Verteilung der Pausen.*

War die Zuteilung eines Studierenden bereits gültig, so hätte versucht werden können, auch die Wünsche des Studierenden möglichst optimal zu berücksichtigen – dies konnte aus zeitlichen Gründen jedoch nur noch in Ausnahmefällen gewährleistet werden.

Die Zuteilung „von Hand“ wurde mit Hilfe einer Gruppe von Studierenden durchgeführt und kostete über 80 Stunden Arbeit. Für die Folgejahre war (und ist) mit einer einmal jährlich anstehenden Verteilung der Praktika zu rechnen, wobei die Zahl der Studierenden und die Anzahl, die Termine, die Länge und die Kapazität von Praktika in einem gewissen Rahmen variieren kann. Auch sind gewisse Anpassungen der Rahmenbedingungen möglich. Wünschenswert war daher ein leicht anpassbares Softwaresystem, das eine schnelle automatische Zuteilung leistet, die die *Nebenbedingungen erfüllt* und dabei noch *besser* die Wünsche der Studierenden berücksichtigt, als dies bei der händischen Zuteilung der Fall war.

3.4.4 Auszunutzendes Anwenderwissen

Das Problem der klinischen Rotation kann zwar nicht als KOP mit Raumbezug eingeordnet werden, trotzdem ist es möglich, Löungsvorschläge partiell zu bewerten, um Stellen mit lokalem Verbesserungspotential zu finden – auf genau dieser Eigenschaft des Problems basierte auch die Zuteilung von Hand, wie sie in den letzten Jahren durchgeführt wurde.

Lokales Verbesserungspotential durch Variation der Konfiguration entsprechender Objekte besteht beispielsweise immer dann, wenn ein Studierender in einem Bereich kein Praktikum – oder im Wahlpraktikums-Bereich nicht das mit höchster Priorität gewählte Praktikum – zugeteilt

bekommen hat, obwohl das gewünschte Praktikum nicht an allen Terminen „ausgebucht“ ist. Verbesserungspotential besteht auch, wenn die Verteilung der Pausen nicht dem Wunsch des Studierenden entspricht. Oft ist es dann möglich, durch Verschieben und Tauschen von Praktika einen besseren Vorschlag zu generieren.

Außerdem sind für die Zuteilung zu den Rotationskursen lokale Variationsoperatoren bekannt, die – auf die entsprechende Variationsstelle angewendet – Verbesserungen des Lösungsvorschlags erwarten lassen. Bei der Verbesserung der Zuteilung von Hand wurden vor allem einfache „elementare“ Operationen wie zum Beispiel die folgenden verwendet:

- Die *Hinzunahme eines weiteren, noch nicht besuchten Praktikums* – oder die Hinzunahme eines Wahlpraktikums, sofern noch keines der Wahlpraktika besucht wird. Sofern eine solche Hinzunahme möglich ist, kann die Anzahl der verletzten Nebenbedingungen so verringert werden.
- Das *Verschieben eines Praktikumsplatzes* eines Studierenden in einen anderen Zeitabschnitt. Dadurch kann eine gleichmäßigere Auslastung der Praktika erreicht werden oder ein freier Zeitraum für weitere Praktika geschaffen werden. Außerdem kann so versucht werden, den Präferenzen des Studierenden bezüglich der Lage seiner freien Wochen nachzukommen.
- Der *Tausch eines vergebenen Praktikumsplatzes* mit einem noch freien Platz in einem anderen Praktikum, das zur gleichen Zeit stattfindet. Damit kann beispielsweise eine bessere Berücksichtigung der Prioritäten des Studierenden bezüglich des Wahlpraktikums erreicht werden.
- *Das Tauschen zweier parallel stattfindender Praktikumsplätze*, die an unterschiedliche Studierende vergeben wurden. Damit kann ebenso die die Berücksichtigung der Wünsche der Studierenden erhöht werden.

Bei der angestrebten automatisierten Bearbeitung des Problems bietet sich außerdem an, Rekombinationen zweier oder mehrerer Vorschläge zu einem neuen Vorschlag zuzulassen. Dabei werden die Belegungen einiger Studierender von einem Vorschlag übernommen und mit den Belegungen der übrigen Studierenden aus anderen Vorschlägen kombiniert. Bei der Bearbeitung von Hand wurde eine solche Operation aus Komplexitätsgründen jedoch nicht angewendet.

3.5 Zusammenfassung der Ausgangslage und der Anforderungen

Ziel der Arbeit ist die Entwicklung eines Optimierungsverfahrens für Kombinatorische Optimierungsprobleme in der Praxis, das unter anderem zur Lösung von Exemplaren der beschriebenen Anwendungsfälle herangezogen werden kann. Die Ausgangslage lässt sich dabei wie folgt zusammenfassen:

(1) Für die zu bearbeitenden Probleme existiert kein ausreichend gutes problemspezifisches Verfahren.

(2) Der Anwender verfügt über eingeschränktes Wissen für die Bearbeitung konkreter Problemexemplare und ist daher in der Lage, kleinere Exemplare auch von Hand zu bearbeiten.

  (2a) Lösungsvorschläge sind nicht nur global sondern auch partiell bewertbar und dem Anwender sind entsprechende *partielle Bewertungsmethoden* bekannt, aufgrund derer er das lokale Verbesserungspotential von Lösungsvorschlägen einschätzen kann. Die partielle Bewertbarkeit der Lösungsvorschläge setzt implizit auch eine gewisse Dekomponierbarkeit des Problems voraus.

  (2b) Der Anwender kennt *problemspezifische Variationsoperatoren* und kann einschätzen, welcher Operator in einer bestimmten Situation am besten herangezogen werden sollte.

An das Verfahren bestehen folgende Anforderungen:

(1) Entwickelt werden soll ein zunächst *Problem-übergreifend* angelegtes Optimierungsverfahren, das auf einfache Weise für die Bearbeitung von Exemplaren praktischer Probleme angepasst werden kann.

(2) Von einer *exakten* Lösung von Problemexemplaren darf zugunsten der Bearbeitungzeit abgesehen werden. In der Realität kann als erfolgreiche Bearbeitung eines Optimierungsproblems gelten, wenn das Verfahren nach annehmbarer Zeit eine *sehr gute* Lösung zurück liefert (die beispielsweise besser ist, als eine real existierende Konfiguration).

(3) Die *Evaluation* eines Lösungsvorschlags, d.h. die Auswertung der Zielfunktion kann sehr *aufwändig* sein. Das Verfahren sollte daher möglichst wenig Lösungsvorschläge auswerten müssen.

(4) Das beschriebene Anwenderwissen sollte möglichst umfassend ausgeschöpft werden, um das Optimierungsverfahren zu beschleunigen.

  (4a) Dem Anwender bekannte problemspezifische Variationsoperatoren sollten integrierbar sein.

  (4b) Partielle Bewertungsmethoden sollten angewendet werden, um zu variierende Entscheidungsvariablen und passende Variationsoperatoren auszuwählen.

Kapitel 4

Ansätze für die Bearbeitung von Optimierungsproblemen

Kapitel 4 beschreibt die theoretischen Grundlagen für das entwickelte Verfahren. Zunächst werden Klassifikationskriterien für die Lösungsansätze eingeführt (Abschnitt 4.1). Abschnitt 4.2 behandelt Prinzipien universell einsetzbarer Verbesserungsheuristiken (sog. *schwache Verfahren*) und führt die bekanntesten Verfahren ein. Genetische Algorithmen, die eine Basis des entwickelten Verfahrens darstellen, werden in Abschnitt 4.3 gesondert betrachtet. Abschnitt 4.4 behandelt einige Eignungskriterien für die Bearbeitung gegebener Problemexemplare mit universellen Heuristiken. Im Anschluss werden die No-Free-Lunch-Theoreme zu Optimierungsalgorithmen von Wolpert und McReady besprochen (Abschnitt 4.5). In Abschnitt 4.6 werden Optimierungsverfahren diskutiert, die problemspezifisches Wissen ausnutzen (sog. *starke Verfahren*). Multiagentensysteme als werden dabei wiederum gesondert in Abschnitt 4.7 betrachtet. Zuletzt werden Möglichkeiten der Verknüpfung starker und schwacher Verfahren besprochen (Abschnitt 4.8).

4.1 Klassifikationskriterien unterschiedlicher Ansätze

Tabelle 4.1 zeigt eine Klassifikation wichtiger Optimierungsverfahren anhand der im Folgenden eingeführten Kriterien. Es finden sich jeweils Verweise auf die Abschnitte, in denen die Verfahren näher erläutert werden.

	Universelle Methoden (schwache Verfahren)	**Problemspezifische Methoden (starke Verfahren)**
Exakte Verfahren	explizite Enumeration (4.1.2)	implizite Enumeration, z.B. Branch-and-Bound (4.6.1)
Konstruktions-heuristiken	-	spezielle Konstruktionsheu-ristiken (4.6.2)
Einpunkt-Verbesserungs-heuristiken	Lokale Suche (Hill-Climbing) in problemübergreifend definierten Nachbarschaften, darauf auf-bauende Metaheuristiken (4.2.2)	Lokale Suche in problem-spezifisch definierten Nachbarschaften, z.B. 2-change (4.2.1), Greedy-MAS (4.7.3)
Mehrpunkt-Verbesserungs-heuristiken	Evolutionäre Algorithmen in ursprünglicher Form wie der kanonische GA (4.3.3), Hybridisierungen mit anderen schwachen Verfahren (4.8.1)	Ameisenalgorithmen (4.7.2), Mit speziellen Heuristiken oder Operatoren hybridisierte GA (4.8.1)

Tabelle 4.1: *Klassifikation wichtiger Optimierungsverfahren*

4.1.1 Starke und schwache Verfahren

Ein für diese Arbeit entscheidendes Klassifikationskriterium für Optimierungsverfahren ist, ob das Verfahren ohne zusätzliche Kenntnisse über die Zielfunktion des Optimierungsproblems (bzw. des konkreten Problemexemplars) universell einsetzbar ist, oder ob spezifisches Wissen über das Problem(exemplar) ausgenutzt wird. Man unterscheidet in diesem Zusammenhang so genannte *starke* und *schwache* Methoden (vgl. [Nis97]).

Universell einsetzbare Verfahren nutzen die Zielfunktion zwar zur Evaluation von Lösungen, verfügen jedoch über kein zusätzliches Wissen über das Problem bzw. die Zielfunktion. Sie werden daher als schwache Methoden oder *Black-Box-Verfahren* bezeichnet.[1] Bei der Generation neuer Lösungen kann randomisiert vorgegangen werden oder nach bestimmten Vorschriften, die beispielsweise mittels einer so genannten Metaheuristik (vgl. 4.1.2) vorgegeben werden. Zentrale Aussage der so genannten *No-Free-Lunch* Theoreme (die in Abschnitt 4.5 behandelt werden) ist, dass alle Verfahren, die kein Problemwissen ausnutzen – über alle Zielfunktionen gemittelt

[1] Letztere Bezeichnung bezieht sich auf den Zusammenhang zwischen dem Verfahren und der Zielfunktion: Diese fungiert gewissermaßen als *Black-Box*, da sie nur zur Evaluation *vollständiger* Lösungsvorschläge zur Verfügung steht: Das Verfahren nutzt keinerlei Informationen über die Beschaffenheit der Funktion wie zur Lage und Verteilung von Optima, zur Dekomponierbarkeit oder zur Höhe der räumlichen Autokorrelation (vgl. A.5.1).

– nicht besser abschneiden können als „Random Search“, d.h. bloße Zufallssuche. Starke Methoden dagegen sind auf spezielle Probleme zugeschnitten und oft um Größenordnungen schneller - dafür können sie natürlich jeweils nur auf eine kleine Teilmenge aller Optimierungsprobleme bzw. Problembeispiele angewendet werden. Wie in Abbildung 4.1 angedeutet, besteht zwischen dem *Umfang des Anwendbarkeitsbereichs* und der *Leistungsfähigkeit* eines Optimierungsverfahrens also ein inverser Zusammenhang [Nis97, S.19].

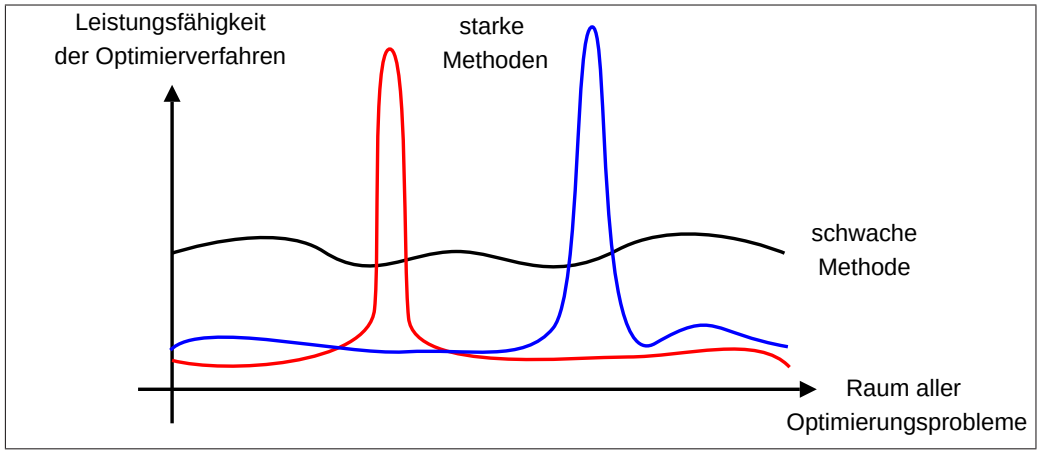


Abbildung 4.1: *Schwache und starke Methoden (vgl. [Nis97, S.19])*

Beispiel TSP:
Betrachten wir beispielsweise neben dem TSP das so genannte *Maximum-TSP* (informell auch als *taxicap ripoff problem* bezeichnet [BGS02]), bei der statt der kürzesten die *längste* Rundreise für gegebene Orte gefunden werden soll.[a] Offenbar wird ein speziell auf das (herkömmliche) TSP zugeschnittenes Spezialverfahren Exemplare von Maximum-TSP-Problemen besonders schlecht lösen können und umgekehrt. Ein universelles Verfahren wird dagegen etwa gleiche Performanz auf Exemplaren beider Problemtypen zeigen.

[a] Beim allgemeinen Maximum-TSP handelt es sich ebenfalls um ein NP-vollständiges Problem. Untersuchungen zur Komplexität verschiedener Varianten finden sich beispielsweise in [BGS02].

4.1.2 Exakte Verfahren und Heuristiken

Exakte Verfahren
Exakte Verfahren liefern ein globales Optimum, sind aber im Allgemeinen sehr rechen- und damit zeitintensiv. Um ein allgemeines Optimierungsproblem ohne zusätzliches Wissen über die Zielfunktion *exakt* zu lösen, muss der Lösungsraum komplett durchsucht werden. Bei kombinatorischen Optimierungsproblemen ist eine solche vollständige *explizite Enumeration* aller Lösungen aufgrund der Endlichkeit des Suchraums zwar theoretisch möglich, aber in der Praxis sehr ineffizient und sie sprengt daher ab einer gewissen Problemgröße technische Grenzen. Verfahren der *impliziten Enumeration*, bei denen bestimmte Bereiche für die Belegungen von Entscheidungsvariablen ausgeklammert werden, können nur angewendet werden, wenn zusätzliches Wissen über

die Zielfunktion besteht. Sie sind daher als starke Verfahren zu bezeichnen. Branch-and-Bound-Verfahren (vgl. Abschnitt 4.6.1), die zur Abschätzung der Güte von Restlösungen Approximationsalgorithmen verwenden, gehören zu Verfahren der impliziten Enumeration.[2]

Heuristiken
Im Gegensatz zu exakten Verfahren durchsuchen heuristische Verfahren weder explizit noch implizit den gesamten Lösungsraum, sondern beschränken sich auf besonders „vielversprechende" Bereiche. Nissen [Nis97] bezeichnet mit *Heuristik* (altgriech.: $\epsilon\upsilon\rho\iota\sigma\kappa\omega$ = ich finde) eine *„nichtwillkürliche, häufig iterative Methode [...], die darauf abzielt, für eine gegebene Problemstellung in begrenzter Zeit eine oder mehrere gute Lösungen zu finden, ohne dass garantiert werden kann, eine global optimale Lösung zu finden"* (S.18). Eine Optimalitäts-Garantie für gefundene Lösungen können Heuristiken zwar nicht liefern. In der Praxis genügt es jedoch oft, eine gute Lösung in annehmbarer Zeit zu finden (vgl. 3.5). Man unterscheidet dabei *Konstruktionsheuristiken* und *Verbesserungsheuristiken*.

Bei *Konstruktionsheuristiken* wird eine Lösung schrittweise aufgebaut, mit dem Ziel, eine möglichst gute Lösung zu erzeugen, d.h. die Entscheidungsvariablen werden der Reihe nach belegt. Ein solches Vorgehen bringt allerdings ohne zusätzliche Kenntnisse über das Problem(exemplar) – insbesondere der partiellen Bewertbarkeit und der damit einhergehenden Abschätzbarkeit der Güte derzeitiger Teillösungen – keine Vorteile. Konstruktionsheuristiken werden oft auch verwendet, um Startlösungen für Verbesserungsverfahren zu produzieren.

Verbesserungsheuristiken gehen von einer Lösung *(Einpunkt-Verfahren)* oder mehreren Lösungen *(Mehrpunkt- oder populationsbasierte Verfahren)* aus und generieren iterativ in deterministischer oder nichtdeterministischer Weise eine bzw. mehrere darauf aufbauende neue Lösung(en). Einige Verfahren akzeptieren dabei auch (zwischenzeitliche) Übergänge zu schlechteren Lösungen. Dieser Vorgang wird abgebrochen, wenn keine bessere Lösung mehr gefunden wird, eine Zeitgrenze überschritten wird, oder ein anderes Abbruchkriterium zutrifft. Eine Verbesserungsheuristik findet im Allgemeinen weder die optimale Lösung, noch kann sie entscheiden, ob die bisher beste gefundene Lösung eine global optimale Lösung ist. Nur wenn zusätzliche Informationen über die Zielfunktion verfügbar sind, kann die Güte einer gefundenen Lösung im Bezug zum globalen Optimum abgeschätzt werden.

Metaheuristiken
Problemübergreifend definierte Vorschriften für die Bearbeitung von Optimierungsproblemen werden auch als *Metaheuristiken* bezeichnet. Metaheuristiken können beispielsweise – vergleichbar mit „Daumenregeln" – den Einsatz eines oder mehrerer untergeordneter (meist lokaler) Optimierungsverfahren steuern und damit den Suchprozess in bestimmte Bereiche lenken, in denen aufgrund der bisherigen Evaluationsergebnisse gute Lösungen vermutet werden. Beispiele für Metaheuristiken sind Simulated Annealing, Tabu-Suche, Sintflut-Algorithmus, (A.1) Ameisenalgorithmen und Evolutionäre Algorithmen. Aber auch die Brute-Force-Suche, d.h. die vollständige Enumeration aller Lösungsvorschläge, kann im Prinzip als Metaheuristik bezeichnet werden.

Eine Metaheuristik nutzt selbst nur das Wissen über die Güte bisher evaluierter Lösungen. Der Begriff Metaheuristik wird daher auch als Gegensatz zu problemspezifischen Heuristiken verwendet, die über zusätzliches Wissen über das Problem(exemplar) verfügen.

[2] Wird bei der Anwendung eines Branch and Bound-Algorithmus jedoch ein *heuristisches* Verfahren zur Abschätzung verbleibender Restlösungen verwendet (vgl. [Hro03, S. 176]), das die Kosten von Restlösungen möglicherweise überschätzt, kann nicht mehr von einem exakten Verfahren gesprochen werden.

4.2 Universelle Heuristiken

Einpunktverfahren gehen von einem Lösungsvorschlag aus, der schrittweise variiert wird. Dabei wird meist in der *Nachbarschaft* der bisherigen Lösung nach einer weiteren – möglicherweise besseren – Lösung gesucht („Nachbarschaftssuche“, *local search heuristic*), mit der impliziten Annahme bzw. Hoffnung, dass zwischen der „Nähe“ und der Güte von Lösungen ein Zusammenhang besteht. Man spricht deshalb auch von *lokalen Verbesserungsverfahren*.[3] Bevor einige solcher Verfahren vorgestellt werden, sollen noch die Begriffe „Nachbarschaft“ und „lokales Optimum“ eingeführt werden.

4.2.1 Nachbarschaften und lokale Optima

Generell sollen Lösungen als benachbart gelten, die sich nur geringfügig unterscheiden. Zur Definition der Nachbarschaft kann eine so genannte *Nachbarschaftsfunktion* N herangezogen werden. Die Funktion spezifiziert für jede Lösung x eine Menge $N(x)$ weiterer Lösungen, die als benachbart eingestuft werden. Sie kann einerseits unabhängig vom konkreten Problem(exemplar) definiert werden und auf der Distanz der Lösungen im Lösungsraum beruhen, der durch die Entscheidungsvariablen aufgespannt wird. Bei m Variablen kann die Distanz zwischen zwei Lösungen $x = \langle x_1..x_m \rangle$ und $y = \langle y_1..y_m \rangle$ durch eine der Minowski-Metriken definiert werden:

$$d_{xy}^r = \sqrt[r]{\sum_{k=1}^{m} |x_k - y_k|^r} \text{ mit } r \in \mathbb{N}$$

Die bekanntesten Minowski-Metriken sind die Manhatten-Distanz (r=1) und die euklidsche Distanz (r=2). Mit der Menge $N_d(x)$ von Lösungen innerhalb einer bestimmten Distanz d ist eine natürliche Definition der Nachbarschaft zu einer Lösung x gegeben [PS82, S.7]:

$$N_d(x) = \{y | d_{xy} \leq d\}$$

Bei einer solchen Definition auf Grundlage *kodierter* Lösungsvorschläge variiert die Nachbarschaft eines Vorschlags bei unterschiedlichen Kodierungen. Alternativ kommen problemspezifische Definitionen der Nachbarschaft in Frage, die auf der Ähnlichkeit der *dekodierten* Lösungsvorschläge beruhen – sofern entsprechendes Wissen über das Problem vorhanden ist. Nähere Erläuterungen sowie Möglichkeiten zur problemspezifischen Definition von Nachbarschaften für bekannte KOP finden sich beispielsweise in [MAK07].

Beispiel TSP
Bei TSP können Nachbarschaften über die Anzahl von Kanten definiert werden, die aus einer bestehenden gültigen Rundreise entfernt und durch neue Kanten ersetzt werden, um wieder eine gültige Rundreise zu erhalten. Die *„k-change“* - Nachbarschaft einer Lösung l umfasst alle (gültigen) Rundreisen, die – ausgehend von der Lösung l – durch den Tausch von höchstens k Kanten entstehen. Die Kardinalität einer solchen Nachbarschaft ist $\Omega(n^k)$ (vgl. [Hro03]). Bei metrischen TSP stellen überkreuzungsfreie Rundreisen lokale Optima bezüglich der *2-change*-Nachbarschaft dar.

[3] Optimierungsalgorithmen werden oft auch in lokale und globale Suchverfahren eingeteilt, je nachdem, ob der Algorithmus dazu in der Lage ist, ein lokales Optimum wieder zu verlassen. Sowohl lokale als auch globale Suchverfahren können auf lokale Verbesserungsoperationen zurückgreifen.

Auf Grundlage einer Nachbarschaftsfunktion N können *lokale Optima* der Fitness-Funktion f definiert werden. Eine Lösung x wird als *lokales Optimum* bezüglich N bezeichnet, wenn (für ein Maximierungsproblem[4]) gilt:

$$\forall y \in N_x : f(x) \geq f(y)$$

Ein globales Optimum (vgl. Abschnitt 2.2) ist somit immer auch gleichzeitig ein lokales Optimum.

4.2.2 Lokale Suche in diskreten Lösungsräumen

Der bekannteste lokale Suchalgorithmus für diskrete Lösungsräume ist das *Steepest Ascent Hill Climbing* (bzw. *Steepest Descent Hill Descending*)[5]-Verfahren (vgl. [RN03], S.112). Ausgehend von einem momentanen Lösungsvorschlag wird zunächst der beste benachbarte Vorschlag als neuer Lösungskandidat ausgewählt. Wenn er einen höheren Zielfunktions-Wert erreicht, als der momentane Lösungsvorschlag, wird zu diesem gewechselt und in der (neuen) Nachbarschaft nach einer noch besseren Lösung gesucht. Ansonsten endet das Verfahren und liefert den letzten Lösungsvorschlag als Ergebnis zurück (bei dem es sich dann per Definition immer um ein lokales Optimum bezüglich der Nachbarschaft handelt).

Natürlich hängt der Erfolg sowie die Performanz einer solchen lokalen Suche von der Definition der Nachbarschaft und der daraus resultierenden Anzahl und Lage lokaler Optima ab: Kleinere Nachbarschaften können schneller evaluiert werden, allerdings wächst hier die Gefahr vieler (möglicherweise gegenüber dem globalen Optimum auch deutlich schlechterer) lokaler Optima. Mit Nachbarschaften höherer Kardinalität besteht geringere Gefahr, in schlechten lokalen Optima gefangen zu bleiben. Dafür erhöht sich natürlich der Aufwand zur Evaluation aller Nachbarn (vgl. z.B. [Hro03]) – ein Nachteil insbesondere bei der Bearbeitung von Problemen, bei denen die Auswertung der Zielfunktion zeitinzensiv ist, wie bei den Anwendungsbeispielen dieser Arbeit.

Beispiel TSP
Bei der Bearbeitung von TSP durch lokale Suche in *k-change*-Nachbarschaften (4.2.1) wird für k meist ein Wert ≤ 3 gewählt, da der Aufwand zur Evaluation der $\Omega(n^k)$ Lösungen in der Nachbarschaft sonst zu hoch wird. Dadurch kann jedoch die zu erwartende Lösungsgüte stark eingeschränkt werden, wie Papadimitriou und Steiglitz [PS78] an einigen Beispielen zeigen: Sie konstruieren Problemexemplare der Größe n, die für eine *k-change*-Nachbarschaft mit $k < \frac{3n}{8}$ bezüglich n exponentiell viele, beliebig schlechte lokale Optima aufweisen.

Andere *Hill-Climbing*-Varianten und darauf aufbauende Metaheuristiken wir *Simulated Annealing* [Cer83, KGV83] oder die *Tabu-Suche* [Glo86] versuchen, zwei Nachteile des *Steepest Ascent Hill-Climbing* zu mildern: Sie sorgen dafür, dass lokale Optima zugunsten übergangsweise schlechterer Lösungen wieder verlassen werden können, um eventuell dadurch später bessere Lösungen erreichen zu können und/oder sie erlauben auch Übergänge zu neuen (besseren) Lösungen, ohne dass alle Lösungen der entsprechenden Nachbarschaft evaluiert werden müssen. Beispiele dazu finden sich im Anhang (A.1).

[4] Für Minimierungsprobleme ist $f(x) \geq f(y)$ durch $f(x) \leq f(y)$ zu ersetzen.

[5] Wird im Folgenden von *Hill-Climbing*-Verfahren gesprochen, so sollen darunter auch die entsprechenden *Hill-Decending*-Verfahren für Minimierungsprobleme fallen.

4.2.3 Rekombination von Lösungsvorschlägen

Mehrpunktverfahren behandeln mehrere Lösungen parallel und kombinieren in der Regel auch „Teile“ bisheriger Lösungen zu neuen Lösungen. Zur Konstruktion einer neuen Lösung werden dazu Belegungen der Entscheidungsvariablen von zwei oder mehr bisherigen Lösungen *rekombiniert*. Die Rekombination von Teillösungen bietet sich besonders dann an, wenn die Abhängigkeiten der Entscheidungsvariablen untereinander im Bezug auf die Zielfunktion nicht zu stark sind, so dass die Kombination von Belegungen guter (Teil-)Lösungen mit hoher Wahrscheinlichkeit wieder zu guten Lösungen führt. Mit der Epistasie-Varianz (Kapitel 4.4.2) wird versucht, die Stärke der Nichtlinearitäten unter den Variablen zu messen.

Verschiedene Formen der Rekombination können unterschieden werden. Bei der *diskreten Rekombination* wird jede Entscheidungsvariable mit ihrem Wert in einer der bisherigen Lösungen belegt.[6] Die Auswahl der Lösung, deren Belegung jeweils herangezogen wird, kann beispielsweise stochastisch erfolgen. Von *intermediärer Rekombination* spricht man, wenn die Entscheidungsvariablen in einem Bereich, der von den Belegungen bisheriger Lösungen abhängig ist, zufällig belegt werden. Für Kombinatorische Optimierungsprobleme bietet sich aufgrund der diskreten Wertebereiche der Entscheidungsvariablen im Normalfall die intermediäre Rekombination nicht an. Für Genetische Algorithmen schlägt Grefenstette [Gre87] *problemspezifische* Rekombinationsoperatoren vor, die vorhandenes Anwenderwissen bei der Kombination von Lösungen ausnutzen.

4.2.4 Populationsbasierte Methoden

Bei populationsbasierten Ansätzen wird nicht mehr nur von *einem* Lösungsvorschlag ausgegangen, stattdessen wird immer eine *Menge* (Population) von Vorschlägen parallel betrachtet.

Mit *Evolutionären Algorithmen* wird eine ganze Klasse von Optimierungsverfahren bezeichnet, die sich am Vorbild der natürlichen Evolution (Mutation, Rekombination und Selektion von Individuen) orientieren. Aus der Population einer Generation werden schrittweise Lösungsvorschläge ausgewählt (selektiert), die lokal variiert (mutiert) und rekombiniert werden. Die so entstehenden Lösungsvorschläge bilden die Population der nächsten Generation. Da die Evolutionären Algorithmen, speziell die Genetischen Algorithmen (GA) eine wichtige Grundlage für diese Arbeit darstellen, werden sie im nächsten Abschnitt getrennt und ausführlich behandelt.

Die zu den *Schwarmalgorithmen* zählenden *Ameisenalgorithmen* wurden 1991 von Dorigo [DMC91, Dor92] eingeführt. Virtuelle Ameisen versuchen, kürzeste Wege in Graphen zu finden, wobei sie Wissen über die Entfernungen der Knoten nutzen („Sichtbarkeit“) und untereinander Informationen über die Güte bereits abgeschrittener Wege austauschen („Pheromone“). Ameisenalgorithmen können für die Lösung von Optimierungsproblemen eingesetzt werden, die man als Wegsuche in Graphen modellieren kann. Die Zielfunktion ist hier keine *Black-Box*, denn neben der Güte von Gesamtlösungen ist auch die Entfernung der Knoten untereinander bekannt und fließt als problemspezifisches Wissen in das Verhalten der virtuellen Ameisen ein. Ameisenalgorithmen sind insofern als starke Verfahren zu bezeichnen. Sie können als Multiagentensysteme eingeordnet werden [DC99], die eine weitere wichtige Grundlage für das entwickelte Optimierungsverfahren darstellen. Eine genauere Beschreibung findet sich daher im Abschnitt 4.7.

[6] Bei Genetischen Algorithmen mit binärer Kodierung der Entscheidungsvariablen werden stattdessen Teile der Bitketten (*Chromosome*) miteinander kombiniert.

4.3 Genetische Algorithmen

4.3.1 Die natürliche Evolution als Vorbild für Optimierungsalgorithmen

Der Evolutionsprozess passt im Laufe der Zeit verschiedenartigste Lebewesen an ihre wechselnden Umwelt- und Lebensbedingungen an und kann insofern als permanenter Optimierungsvorgang unterschiedlicher Arten in einer dynamischen Umgebung angesehen werden: „*To be shure, selection is an optimization process ...*“ [May88, S.105].

Der Prozess beruht in vereinfachender Sicht auf zwei entscheidenden, aufeinander aufbauenden Prinzipien, und zwar der *Variation* einer Population durch Kombination und Mutation von genetischem Material sowie der *Selektion* bestimmter Individuen [May88, S.97f]. Besser an ihre Umwelt angepasste Individuen können das genetische Material, auf dem ihre Eigenschaften kodiert sind, öfter an Nachkommen weitergeben, da sie aufgrund höherer Lebenserwartung und Durchsetzungsfähigkeit gegenüber Artgenossen häufiger am Reproduktionsprozess beteiligt sind (Selektion). Bei der Weitergabe des Materials wird das elterliche Erbmaterial durchmischt, außerdem zu zufälligen Veränderungen in der DNS kommen.

Kombinationen und Änderungen, die sich beim Phänotyp – d.h. beim in ein Lebewesen dekodierten Erbmaterial bewähren – haben eine höhere Chance, auch in späteren Generationen aufzutreten. Die Evolution selbst verfügt dabei über keinerlei Gedächtnis, sie „arbeitet“ ausschließlich auf dem gegenwärtig vorhandenen Genmaterial. Einmal „ausgestorbene“ Eigenschaften können also nur zufällig wieder zurückgewonnen werden.

Die Erkenntnis, dass für die Evolution primär die zufällige Durchmischung und Variation des elterlichen Genmaterials bei der Reproduktion und der Selektionsprozess unter den Individuen verantwortlich ist, ist noch nicht besonders alt: Lamarck (1744 bis 1829) vermutete bei Lebewesen noch die Fähigkeit, sich selbstständig weiterzuentwickeln und erworbene Eigenschaften an ihre Nachkommen weiterzugeben.[7] Heutiger Stand der Wissenschaft ist jedoch, dass das Individuum während seiner Lebenszeit keine Eigenschaften erwirbt, die es an Nachkommen vererben kann, sondern als Phänotyp der auf den Chromosomen kodierten Eigenschaften lediglich zur „Erprobung“ zufälliger Variationen des elterlichen Erbmaterials dient. Charles Darwin (1809 bis 1882) führte 1859 mit seinem Werk „*On the Origin of Species by Means of Natural Selection*“ dieses heute weitestgehend[8] anerkannte Prinzip der natürlichen Selektion ein.

4.3.2 Evolutionäre Algorithmen

Anfang der sechziger Jahre begannen unterschiedliche Forschergruppen Optimierungsalgorithmen zu entwickeln, die auf Modellen der natürlichen Evolution beruhen (vgl. [FOW66], [Rec73], [Schw77], [Hol75], [Koz92]). Diese Algorithmen haben seit Mitte der 1980er Jahre enorm an Bedeutung gewonnen. Dazu trägt sowohl ihre universelle Einsetzbarkeit als auch die technische Entwicklung (Hardware, Parallelrechner) bei. Aufgrund der Verwendung von Rekombinationsoperatoren erstreckt sich ihre Einsetzbarkeit teilweise auch über (Optimierungs-)Probleme, die sich mit Einpunkt-Verfahren (Kapitel 4.1.2) nur unbefriedigend lösen lassen.

[7] Die so genannten Memetischen Algorithmen (Abschnitt 4.8.2), eine Verknüpfung Genetischer Algorithmen mit lokaler Suche, werden oft mit dieser „Lamarckschen“ Form der Evolution in Verbindung gebracht.

[8] Kritiker dieser Theorie der natürlichen Selektion bezweifeln oft, dass die extrem unterschiedlichen Eigenschaften verschiedener Arten tatsächlich durch solche schrittweisen, kleinen Veränderungen entstehen können.

Die genannten Algorithmen werden als *„Evolutionäre Algorithmen (EA)“* bezeichnet.[9] Sie bearbeiten eine „Population“ von *Lösungsvorschlägen* für ein Problembeispiel und imitieren den Evolutionsprozess, indem sie zumeist folgendes „Evolutionsmodell“ verwenden (vgl. [Fog00], S.159):

$$p[t+1] = s(v(p[t])).$$

Die Population p zum Zeitpunkt $t+1$, bezeichnet mit $p[t+1]$, entsteht aus $p[t]$, der Population zum Zeitpunkt t, die zufällig variiert (v) und dann der Selektion (s) unterworfen wird.

Mit den *Evolutionsstrategien (ES)* [Rec73, Schw77], der *Evolutionären Programmierung (EP)* [FOW66], den *genetischen Algorithmen (GA)* [Hol92] und der *Genetischen Programmierung* [Koz92] entstanden vier wesentliche Hauptströmungen auf dem Gebiet der Evolutionären Algorithmen. Diese Ansätze unterscheiden sich – insbesondere in ihrer ursprünglichen Form – hauptsächlich in der Art der Kodierung der Entscheidungsvariablen und der verwendeten Variationsoperatoren: Für GA in ihrer ursprünglichen, so genannten *kanonischen* Form werden ausschließlich binäre Kodierungen der Entscheidungsvariablen verwendet. Der Ansatz setzt hauptsächlich auf den genetischen Operator „Crossover“. Der Mutationsoperator, der bei GA eine untergeordnete Rolle spielt[10], steht dagegen im Fokus der Anhänger von ES und EP. Sie kodieren die Entscheidungsvariablen in reellen Zahlen. Dabei kodieren Evolutionsstrategen (wie Anhänger der GA) einzelne Individuen, während bei EP eine Spezies insgesamt kodiert wird (daher findet auch keine Rekombination unter den Individuen statt [Fog00]). Bei GP repräsentieren die Individuen *Programme*. Ziel ist es, automatisch den Kode für ein Programm zu generieren, das bestimmte Funktionalitäten erfüllt [Koz92]. GP können als „moderne Unterform“ von GA angesehen werden (vgl. [Nis97, S.12]).

Auch wenn die Unterschiede der verschiedenen EA-Ansätze in den letzten Jahren zunehmend verschwimmen, kann die Variante, die für das im Zuge dieser Arbeit entwickelte Optimierungsverfahren eingesetzt wird, am ehesten den Genetischen Algorithmen zugerechnet werden.

4.3.3 Der kanonische GA

Vorreiter für die Entwicklung der Genetischen Algorithmen war John Holland. Bei seinem Hauptwerk „*Adaption in Natural and Artificial Systems*“ [Hol75] (neuere Ausgabe: [Hol92]) steht die Frage im Mittelpunkt, wie sich in der Natur Arten durch die beschriebenen Operatoren an ihre Umgebung adaptieren können (z.B. durch die Entwicklung von Intelligenz, Selbstorganisation usw.). Die von ihm ursprünglich „*reproductive plans*“ genannten GA entwickelte er in diesem Sinne ursprünglich auch nicht zur Lösung bzw. Bearbeitung von Optimierungsproblemen (der Suche nach der *bestmöglichen* Lösung). Ziel war vielmehr, eine hohe *durchschnittliche* Fitness der Individuen in der aktuellen Population zu erreichen. Es existieren unterschiedliche Ausprägungen genetischer Algorithmen und keine einheitliche formale Notation. Der Ablauf der Anwendung eines *kanonischer GA*, d.h. der GA-Variante, die Holland ursprünglich vorschlug, kann in Pseudokode wie folgt angegeben werden [SHF94, S.187]:

[9] Viele Begriffe aus der Terminologie Evolutionärer Algorithmen wurden von der Biologie bzw. speziell für Genetische Algorithmen auch aus der der Genetik übernommen, eine Aufstellung findet sich im Anhang (A.2).

[10] Holland [Hol92] bezeichnet die Mutation als *background operator*.

Genetischer Algorithmus (kanonische Form):

1. Wähle eine geeignete, binäre *Kodierung* der Entscheidungsvariablen.
2. *Initialisiere* zufällig eine Population von Individuen und kodiere diese.
3. *Variiere* die Population iterativ. Wiederhole dazu bis eine Abbruchbedingung zutrifft:
   (a) *Dekodiere* alle Individuen der aktuellen Generation in Lösungen des Problembeispiels und *bewerte* sie gemäß der Fitness-Funktion.
   (b) *Selektiere* Paare (oder größere Subpopulationen) gemäß Heiratsschema und erzeuge mittels *Rekombination* Nachkommen.
   (c) *Mutiere* Nachkommen.
   (d) *Ersetze* Elemente der aktuellen Generation durch die Nachkommen mittels *Ersetzungsschema*.

Auf die unterschiedlichen Varianten für die genannten Schritte beim Ablauf des GA wird im Anhang näher eingegangen (A.3).

4.3.4 Integration von Nebenbedingungen

Für die Integration von Nebenbedingungen in GA (bzw. in EA insgesamt) gibt es keine allgemeingültigen Methoden oder Richtlinien. Die einfachste Möglichkeit besteht darin, unzulässige Lösungen einfach zu verwerfen. Schränken die Nebenbedingungen den Suchraum stark ein, kann es dadurch jedoch zu unerwünschten Performanzeinbußen des Algorithmus kommen, da sehr viele ungültige Lösungen produziert werden.

Eine weitere, nahe liegende Möglichkeit ist, die Verletzung einer oder mehrerer Nebenbedingungen mittels einer *Straffunktion* zu bewerten, die später mit der Zielfunktion zusammen die Fitness eines Individuums bestimmt. Für die Straffunktion kommen unterschiedliche Varianten in Betracht: Von *statischer* Bestrafung spricht man, wenn die Höhe der Strafkosten nur von der Anzahl der verletzten Nebenbedingungen abhängig ist. Bei der *dynamischen* Bestrafung geht zusätzlich der Grad der Verletzung einer Nebenbedingung in die Berechnung des Strafmaßes mit ein, um den Algorithmus schneller „in die richtige Richtung“ zu lenken. Nachteil von Bestrafungsmethoden ist, dass keine allgemeingültigen Aussagen zu einer angemessenen Höhe der Strafkosten gemacht werden können. Wählt man die Strafkosten zu niedrig, so können unzulässige Lösungen besser eingestuft werden als gültige, wählt man sie zu hoch, so dominieren oft zulässige Lösungen niedriger Qualität die Population (vgl. [Bru96]). Außerdem wird wiederum Rechenzeit für die Erzeugung unzulässiger Lösungen verschwendet.

Weitere Alternativen bestehen darin, bereits die *Produktion* ungültiger Lösungen an unterschiedlichen Stellen zu verhindern, indem Wissen über die Beschaffenheit gültiger bzw. ungültiger Lösungen ausgenutzt wird. Das kann beispielsweise durch günstige Wahl von Kodierung und Dekodierung sowie adäquaten Variationsoperatoren erfolgen. Ein Beispiel sind permutationsbasierte Operatoren für Rundreiseprobleme, die das mehrmalige Besuchen derselben Stadt verhindern. Kann die Produktion gültiger Lösungen allein durch geschickte Wahl von Kodierung und Operatoren nicht gewährleistet werden, so können zusätzlich Reparaturalgorithmen zur Anwendung

kommen, die ungültige Lösungen in gültige überführen. Die Möglichkeit, einen Genetischen Algorithmus mit speziellen Lösungstechniken für *Constraint Satisfaction*-Probleme zu hybridisieren, stellt der Bruns in [Bru96] vor.

4.3.5 Varianten für multikriterielle Probleme

Sollen multikriterielle Probleme mit einem Genetischen Algorithmus bearbeitet werden, so kommt zunächst eine Formulierung der Zielfunktion als gewichtete Summe (bzw. anderweitige Aggregation) der Teilziel-Funktionen und eine damit einhergehende Umwandlung in ein monokriterielles Problem in Frage (vgl. Abschnitt 2.3). Fonseca und Fleming [FF95] nennen mit Syswerda und Plamucci, Jakob und Jones einige Autoren, die diesen Aggregationsansatz verfolgen und ordnen auch die Integration von Nebenbedingungen durch eine Strafkosten-Funktion (vgl. 4.3.4) diesem Ansatz zu. Für populationsbasierte Optimierungsverfahren bieten sich neben dem Aggregationsansatz jedoch auch andere Herangehensweisen an. Fonseca und Fleming [FF95] unterscheiden hier die folgenden Techniken:

Nicht Pareto-basierte Techniken: Fonseca und Fleming nennen Schaffer [Scha85] als ersten Autor, der die Möglichkeit der gleichzeitigen Suche nach mehreren, nicht-dominierten Lösungen erkannt hat. Sein Ansatz – genannt *Vector Evaluated Genetic Algorithm (VEGA)* – besteht darin, aus einer Generation für jedes der Teilziele (basierend auf der entsprechenden Teilziel-Funktion) eine eigene Sub-Population zu selektieren. Die Individuen werden dann zu einer Gesamtpopulation zusammengefasst, auf die wie üblich Rekombinations- und Mutationsoperatoren angewendet werden. Aufgrund der Vorgehensweise tendiert die Population zur „Artbildung“ (engl. *speciation*), d.h. es werden verschiedene Arten ausgebildet, deren Individuen jeweils hinsichtlich *einer* Teilziel-Funktionen besonders gut evaluiert werden. Dies widerspricht dem übergeordneten Ziel des Auffindens einer guten *Kompromiss*lösung.

Des Weiteren erwähnen Fonseca und Fleming unter anderem die Ansätze von Fourman [Fou85]. Er vergleicht Lösungen paarweise anhand jeweils eines Teilziels. In einem ersten Ansatz lässt er die Teilziele vom Anwender in eine Reihenfolge bringen, vergleicht anhand des am höchsten priorisierten Ziels und fährt nur bei gleicher Fitness mit dem nächsten Ziel fort. Ein zweiter Ansatz besteht darin, die zum Vergleich heran zu ziehende Teilziel-Funktion jeweils zufällig unter den Verfügbaren auszuwählen.

Pareto-basierte Techniken: Auch wenn die genannten Ansätze *implizit* die Produktion (Pareto-)dominanter Lösungen unterstützen, machen sie keinen *expliziten* Gebrauch der Definition Pareto-optimaler Lösungen [FF95]. Goldberg [Gol89] schlägt als erster vor, die Fitness der Individuen durch Zuweisung von *Rängen* zu bestimmen, die auf der Definition der Pareto-Optimalität beruhen: Alle nicht-dominierten Lösungen im Pool erhalten Rang eins und werden bei der Zuordnung der übrigen Ränge nicht mehr betrachtet. Lösungen, die unter den verbleibenden nicht dominiert werden, erhalten Rang zwei. Analog werden die übrigen Ränge verteilt. Fonseca und Fleming [FF93] verfolgen einen ähnlichen Ansatz: Sie weisen einem Individuum einen Rang korrespondierend zur Anzahl der Individuen zu, von denen es dominiert wird. Basierend auf den vergebenen Rängen wird die Fitness der Individuen durch lineare Interpolation zwischen bestem und schlechtestem Individuum der Population bestimmt.

Nischentechniken: Nischentechniken versuchen dem Phänomen der so genannten „Gendrift“ zu begegnen. Damit wird die Eigenschaft sowohl natürlicher als auch virtueller (endlicher) Populationen bezeichnet, beim Auftreten mehrerer (gleichwertiger) globaler Optima jeweils nur auf eines zu konvergieren. Auch bei Pareto-basierten Selektionsmechanismen tritt dieses Phänomen auf. Um die Gendrift zu verhindern, kann das so genannte „sharing“ zur Anwendung kommen. Dadurch soll erreicht werden, dass die Population die Front Pareto-optimaler Vorschläge möglichst breit vertritt. Die Idee des sharing besteht darin, den Fitness-Wert von Individuen zu verringern („aufzuteilen“), wenn sich in der Nähe im Lösungsraum bereits weitere Individuen befinden. Eine Alternative besteht darin, so genannte „mating restrictions“ [Gol89] zu formulieren, um die „Kreuzung“ ähnlicher Lösungsvorschläge zu verhindern und so die Vielfältigkeit der Vorschläge im Lösungspool zu erhöhen.

Die genannten Techniken können auch kombiniert werden. Engelhard-Funke und Kolonko [EK04] beschreiben einen Genetischen Algorithmus zur Fahrplanoptimierung mit adaptivem Selektionsmechanismus: Je nachdem, wie sich die aktuelle Population im Lösungsraum verteilt, wird die Selektion entweder anhand des Pareto-Rankings der Individuen (vgl. [FF93]) oder entsprechend dem VEGA-Ansatz (vgl. [Scha85]) durchgeführt. Voget [Vog95] bearbeitet ähnliche Problemstellungen ebenfalls mit einem GA mit einer adaptiven Steuerung der Selektionsfunktion. Er verwendet dazu einen Fuzzy-Regler.

Auch mit den Genetischen Algorithmen verwandten Optimierungstechniken können multikriterielle Probleme bearbeitet werden. Meyer [Mey04] beschreibt das Programmsystem MOMBES (Multi Objective Modell Based Evolution Strategy), das Evolutionsstrategien und Neuronale Netze verwendet. Im Focus steht die multikriterielle Optimierung *kontinuierlicher* Zielfunktionen. Neben einer qualitativ hochwertigen Approximation der Pareto-Front liegt der Schwerpunkt auf der Entwicklung eines Decision-Making-Moduls, das *„das interaktive Extrahieren von Wissen über diejenigen Zusammenhänge zwischen den Prozessgrößen gestattet, die zur Herausbildung pareto-optimaler Lösungen führen“* (S. 163).

4.3.6 Das Schemata-Theorem und die Building-Block-Hypothese

Das Schema-Theorem von Holland ist der wohl bekannteste theoretische Ansatz zu den Genetischen Algorithmen und bildet die Grundlage für die Building-Block-Hypothese von Goldberg. Holland führt den Begriff *Schema* für eine Menge von Individuen ein, die gewisse Eigenschaften gemein haben. Für die Zuordnung der Individuen zu Schemata verwendet er die Detektoren δ_1 bis δ_l. Jedes Individuum A kann repräsentiert werden als Tupel $(\delta_1(A)..\delta_l(A))$ von l Werten der Detektoren. Der Einfachheit halber wird der Suchraum dann als Menge möglicher solcher Tupel (anstatt Menge möglicher Individuen) aufgefasst [Hol92, S.67]. Ein Schema ist eine Untermenge dieses Suchraums, die entsteht, indem gewisse Detektorwerte vorgegeben werden, andere jedoch frei wählbar bleiben. Für die frei wählbare Detektorwerte soll im Folgenden der Stern * als „don´t care“- Symbol verwendet werden. Mit $\langle v_{11}, v_{21}, * \rangle$ wird also die Menge von Individuen bezeichnet, bei denen die Detektorvariablen δ_1 und δ_2 die Werte v_{11} und v_{21} liefern, der Wert der Variablen v_3 jedoch frei wählbar ist.

Überträgt man diese Überlegungen auf kanonische GA, so können die Genorte als Detektorvariablen $\delta_1..\delta_l$ interpretiert werden, die jeweils die Werte 0 oder 1 annehmen können. Ein Schema kann dann als Zeichenkette über dem Alphabet {0,1,*} notiert werden, ein Chromosom *passt* zu

einem Schema, wenn es in allen definierten Bits mit dem Schema übereinstimmt (beispielsweise *passen* die Chromosomen $\langle 1100 \rangle$ und $\langle 1110 \rangle$ zum Schema $\langle 11 * 0 \rangle$). Bei einer Chomosomenlänge von L besteht der Suchraum aus 2^L Chromosomen und es können 3^L Schemata gebildet werden. Jedes Chromosom passt zu 2^L Schemata.

Als *Ordnung o(h)* eines Schemas h bezeichnet Holland die Anzahl definierter Gene, also die Anzahl der Positionen in der Bitkette, an denen *nicht* der Stern * steht (beispielsweise gilt: $o(\langle 1, *, 1, * \rangle = 2$ und $o(\langle *, *, * \rangle) = 0)$. Unter der *definierenden Länge d(h)* eines Schemas versteht Holland den Abstand zwischen erstem und letztem definierten Bit eines Schemas (beispielsweise gilt: $d(\langle < 0, 1, *, 1, * > \rangle = 3$ und $d(\langle < 1, 0, * > \rangle) = 1)$.

Holland versucht nun, die Auftrittswahrscheinlichkeiten bestimmter Schemata in der Population in Abhängigkeit zu ihrer Ordnung und definierenden Länge zu bringen. Er betrachtet zunächst einen kanonischen GA (Fitness-proportionale Selektion und Ein-Punkt-Crossover) ohne Mutation und berechnet den erwarteten Anteil $P(\xi, t+1)$ von Individuen der Population zum Zeitpunkt t+1, die zum Schema ξ passen in Abhängigkeit vom Anteil $P(\xi, t)$ zum Zeitpunkt t, der Ordnung $o(\xi)$ und der definierenden Länge $l(\xi)$ des Schemas sowie der durchschnittlichen Fitness $\mu_\xi(t)$ der zum Schema passenden Individuen zum Zeitpunkt t und der durchschnittlichen Fitness $\mu(t)$ der Individuen zum Zeitpunkt t insgesamt. Es ergibt sich [Hol92, S.102]:

$$P(\xi, t+1) \geq \left[1 - P_c \bullet \frac{l(\xi)}{l-1}\left(1 - P(\xi, t)\right)\right] \frac{\mu_\xi(t)}{\mu(t)} P(\xi, t)$$

Dabei bezeichnet P_c den Anteil der Individuen, bei denen in der aktuellen Generation der Crossover-Operator angewendet wird und l die Länge der Chromosome. Liegt die durchschnittliche Fitness der Individuen, die auf das Schema passen, über der durchschnittlichen Fitness insgesamt, so steigt der Anteil der auf das Schema passenden Individuen in der Population exponentiell an, weil diese eine entsprechend höhere Reproduktionswahrscheinlichkeit von $\frac{\mu_\xi(t)}{\mu(t)}$ haben. Dieses Wachstum kann jedoch durch „Aufspalten“ des Schemas bei der Anwendung des Crossover-Operators gebremst werden. Werden zwei Exemplare des Schemas gekreuzt, so entstehen zwar bei der Anwendung des Einpunkt-Crossover wieder Nachkommen, die auf das Schema passen. Bei der Kreuzung eines Exemplars des Schemas mit einem Exemplar, das nicht auf das Schema passt, liegt der Crossover-Punkt jedoch mit einer Wahrscheinlichkeit von $\frac{l(\xi)}{(l-1)}$ innerhalb der definierten Bits des Schemas und das Schema wird zerstört. Die Ausbreitungsrate eines Schemas ist daher abhängig von der durchschnittlichen Fitness sowie der definierenden Länge.

Die Mutation ist eine weitere potenzielle Quelle für den Verlust[11] von Exemplaren eines Schemas von Generation zu Generation. Wird als Mutationsoperation das zufällige Kippen eines Bits mit Wahrscheinlichkeit P_M durchgeführt, so wird ein Schema h mit einer Wahrscheinlichkeit von $P_M \bullet o(h)$ aufgebrochen [Hol92, S.111].

Ohne hier näher auf den Beweis des Schemata-Theorems einzugehen kann man seine zentrale Aussage damit folgendermaßen zusammenfassen: *Schemata mit überdurchschnittlicher Fitness, kurzer definierender Länge und niedriger Ordnung haben eine über die Generationen hinweg exponentiell ansteigende Anzahl von Vertretern* (vgl. [Gol89], S.41). Schemata, die diesen Anforderungen genügen, werden von Goldberg auch als *Building Blocks* bezeichnet. Nach seiner

[11] Natürlich können durch Crossover und Mutation zufällig auch *zusätzliche* Chromosome entstehen, die zum Schema passen. Die rechte Seite der obigen Gleichung stellt daher auch eine untere Schranke dar.

Building-Block-Hypothese lassen sich gute Gesamtlösungen durch Kombination guter Teillösungen konstruieren:

> „Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks." [Gol89, S.41]

Kritik und Folgerungen aus Schema-Theorem und Building-Block-Hypothese

Kritik am Schemata-Theorem und an der Building-Block-Hypothese basiert vor allem auf den folgenden Punkten:

Zunächst gilt das Schemata-Theorem eigentlich nur für unendliche Populationsgrößen. In realen Populationen kommt es sowohl bei der natürlichen Evolution als auch beim GA zu nicht vernachlässigbaren Abweichungen der Erwartungswerte für die Ausbreitung der Anteile der zu einem Schema passenden Individuen.

Holland geht des Weiteren von keiner bzw. nur geringer Epistasie der Entscheidungsvariablen aus: *„Die Vorstellung, gute Teillösungen lassen sich sukzessive aus Partiallösungen zusammensetzen, kann nur als erste Näherung akzeptiert werden"* [Nis94, S.111]. Gute Teillösungen können überhaupt nur identifiziert werden, wenn die *Wechselbeziehungen* zwischen den Genen im Bezug auf die Zielfunktion nicht zu stark sind. Diese Wechselbeziehungen werden auch als *Epistasie* bezeichnet und im nächsten Abschnitt näher erläutert. Je höher die Epistasie zwischen Genen, desto mehr nähert sich $\mu_\xi(t)$ für alle Schemata ξ an $\mu(t)$ an. Sind die Wechselbeziehungen dagegen sehr gering, so kann das Optimierungsproblem meist effizienter durch separate Optimierung der einzelnen Entscheidungsvariablen gelöst bzw. bearbeitet werden.

Die Aussagen zur Wahrscheinlichkeit des Auseinanderbrechens eines Schemas in Abhängigkeit von dessen definierender Länge implizieren, dass zusammengehörende Gene (d.h. Gene, die die gleiche Entscheidungsvariable kodieren oder auf andere Weite stark miteinander interagieren) möglichst nah nebeneinander auf dem Chromosom kodiert werden sollten. Dies gilt allerdings nur für den hier zugrunde gelegten Ein-Punkt-Crossover-Operator. Für andere Crossover-Operatoren müsste die definierende Länge durch ein entsprechendes Maß ersetzt werden.

Bei aller Kritik an Schemata-Theorem und Building-Block-Hypothese können doch sinnvolle Regeln für die Auswahl der Art der Kodierung und der Crossover-Operatoren daraus abgeleitet werden. Bei vielen realen Problemen ist eine Kodierung der Entscheidungsvariablen in Gene mit relativ geringer Epistasie auf einfache Weise möglich. Dann kann die Effizienz eines GA meist verbessert werden, wenn durch sinnvolle Kodierung und Auswahl von Crossover-Operatoren gute Teillösungen („building blocks") aus zusammen gehörenden oder stark interagierenden Genen möglichst nicht auseinander gerissen werden.[12] Dazu können aber keine *problemübergreifend* guten Operatoren angegeben werden, stattdessen sollten die Operatoren an das entsprechende Optimierungsproblem angepasst werden (vgl. [Gre87]). Für KOP mit Raumbezug gilt beispielsweise oft, dass Belegungen von Entscheidungsvariablen, deren zugrunde liegende Objekte geringe Distanzen aufweisen, möglichst gemeinsam vererbt werden sollten.

[12] Ein Beispiel für die Auswirkung Schemata-erhaltender bzw. -zerstörender Rekombinationsoperatoren findet sich im Anhang (A.4)

4.4 „Hardness“ von Optimierungsproblemen

Die Performanz aller universellen Optimierungsverfahren ist *gemittelt über alle Exemplare aller Optimierungsprobleme* gleich, so die zentrale Aussage der „*No Free Lunch*“-Theoreme, die im nächsten Abschnitt behandelt werden. Dennoch ist die Eignung bestimmter Verfahren für *bestimmte Probleme bzw. Problemexemplare* unterschiedlich. Man spricht hier von der „*hardness*“ eines Optimierungsproblems bzw. eines Exemplars für eine bestimmte Heuristik – nicht zu verwechseln mit der Komplexität des Optimierungsproblems selbst. Zur Beurteilung der *hardness* können unterschiedliche Maßzahlen herangezogen werden, einige davon werden in den folgenden Abschnitten eingeführt.

4.4.1 Fitness-Landschaften und der Zusammenhang zwischen Distanz und Lösungsgüte

Das Konzept der *Fitness-Landschaft* wurde vom Genetiker Wright eingeführt [Wri32], um den Zusammenhang zwischen Reproduktionsraten von Arten und ihren Gen-Kombinationen zu beschreiben. Die „Landschaft“ ergibt sich als Hyberebene, die durch die Gene und ihre möglichen Ausprägungen (Allele) aufgespannt wird. Die Arten mit ihren unterschiedlichen Gen-Kombinationen finden sich als Punkte in dieser Landschaft, ihre Höhe wird durch die jeweilige Reproduktionsrate bestimmt. Hoch angepasste Arten nehmen Bereiche nahe der Gipfel der Landschaft ein.

In der Optimierungstheorie wird das Konzept der Fitness-Landschaft zur Darstellung und Vorhersage des Verhaltens von Optimierungsalgorithmen – speziell von Verbesserungsheuristiken – verwendet.[13] Die Fitness-Landschaft setzt sich hier aus einer Menge von Punkten zusammen, die die Lösungen eines Problemexemplars darstellen. Die Zielfunktion, die jedem Punkt eine „Höhe“ zuordnet und die Nachbarschaftsfunktion definieren gemeinsam die räumliche Struktur der Fitness-Landschaft. Die Distanz zwischen zwei Lösungen x und y wird wieder auf Grundlage der Nachbarschaftsfunktion (vgl. 4.2.1) definiert, auf der das Verfahren beruht. Sie misst die Anzahl der Schritte, die das Verfahren benötigt, um von der Lösung x zur Lösung y zu gelangen. Insbesondere ergeben sich dadurch unterschiedliche Landschaften in Abhängigkeit der eingesetzten Operatoren [Jon95].

Gehen wir im Folgenden von einem Maximierungsproblem aus, so ist Ziel der Optimierung das Auffinden eines höchsten Berges in dieser Landschaft. Die Berge in der Fitness-Landschaft stellen lokale Optima dar. Wie bereits erwähnt, ist ein Nachteil von Verfahren, die auf lokaler Suche basieren, dass sie leicht in solchen lokalen Optima „hängen bleiben“ können. Die Anzahl (und die Qualität) lokaler Optima kann daher ein Kriterium für die Schwierigkeit von Problemexemplaren für entsprechende Verfahren sein.

Mit der räumlichen Autokorrelation [KL87], der Korrelationslänge [Wei90] und der Fitness-Distanz-Korrelation (FDC) [JF95] wird versucht, den Zusammenhang zwischen Distanz von Lösungen und ihrer Fitness zu messen. Nähere Erläuterungen zu diesen Maßen finden sich im Anhang (A.5).

[13] Stadler [Sta02] liefert einen Überblick über unterschiedliche Ansätze der Analyse von Fitness-Landschaften.

4.4.2 Rekombinierbarkeit von Teillösungen: Epistasie und Epistasievarianz

Der Begriff „Epistasie“ stammt aus der Genetik. Ein Gen, dessen Ausprägung die Wirkung eines oder mehrerer anderer Gene unterdrückt, wird als „epistatisches Gen“ bezeichnet. Treten epistatische Gene auf, so kann die Gesamtwirkung mehrerer Gene nicht mehr als Summe der Einzelwirkungen erklärt werden.

Analog wird der Begriff im Zusammenhang mit Optimierungsproblemen verwendet, wenn sich Auswirkungen der Belegungen von Entscheidungsvariablen im Hinblick auf die Gesamtgüte einer Lösung gegenseitig beeinflussen. Mit Epistasie-Maßen wird versucht, die Stärke der Nichtlinearitäten unter den Variablen zu messen.[14]

Liegt keine Epistasie unter den Entscheidungsvariablen vor, so ist die Kodierung (und damit auch das Problemexemplar) linear, d.h. die Zielfunktion kann als Summe von Termen ohne Produkte unter den Entscheidungsvariablen ausgedrückt werden und die optimalen Belegungen für die Entscheidungsvariablen können unabhängig voneinander gesucht werden. Kombinatorische Optimierungsprobleme sind jedoch im Allgemeinen nichtlinear, der Zielfunktionswert ist weniger von *einzelnen Belegungen* sondern von deren *Kombination* abhängig.

Beispiel TSP:
Wird für das TSP die in Abschnitt 2.3 vorgestellte Kodierung gewählt, so hat die Belegung einer einzelnen Entscheidungsvariable mit einem „Besuchsindex“ noch keine Auswirkung auf die zu erwartende Güte der Rundreise. Ob die Belegung günstig gewählt ist, ist davon abhängig, welche Variablen mit benachbarten Indizes belegt sind, da erst dadurch die Länge der so definierten Teilreise bestimmt werden kann.

Lassen sich Gruppen von Variablen bilden, so dass unter Variablen aus verschiedenen Gruppen keine Abhängigkeiten bestehen, dann ist das Problem in Subprobleme (über die Variablen jeweils einer Gruppe) *dekomponierbar*, die getrennt voneinander bearbeitet werden können. Die Teillösungen können dann zu einer Gesamtlösung kombiniert werden, deren Güte der Aggregation der Güte der Teillösungen entspricht. Vollständige Dekomponierbarkeit ist bei realen KOP nicht zu erwarten. Im Folgenden wird daher von *gewisser Dekomponierbarkeit* gesprochen, wenn die Abhängigkeiten unter Variablen verschiedener Gruppen *gering* sind und die Güte der Gesamtlösung *in etwa* der Summe der Güte der Teillösungen entspricht.

Die meisten populationsbasierten Ansätze setzen implizit auf eine solche *gewisse Dekomponierbarkeit* des Optimierungsproblems, da sie nicht nur den Lösungsraum in der Nähe bisher evaluierter Lösungen nach neuen Lösungen absuchen, sondern Teile von Lösungen – d.h. Belegungen von Teilmengen der Entscheidungsvariablen – zu neuen Gesamtlösungen kombinieren. Eine solche Rekombination der Belegungen ist nur dann erfolgversprechend, wenn die Abhängigkeiten der Variablen einer Teilmenge von den Variablen der anderen Teilmenge(n) nicht zu hoch sind. Ansonsten ist nicht zu erwarten, dass durch die Rekombination von Teilen guter Ausgangslösungen wieder gute neue Lösungen entstehen, das Optimierungsverfahren wird sich daher in der Performanz kaum von bloßer Zufallssuche unterscheiden. Als ein Maß für die Schwierigkeit der Bearbeitung eines Optimierungsproblems (bei gegebener Kodierung) mit populationsbasierten Ansätzen wie Genetischen Algorithmen kommt daher die Epistasie in Frage.

[14] Die Höhe dieses wechselseitigen Einflusses ist dabei abhängig von der Kodierung des Problems. Epistasie ist daher eine Eigenschaft der Kodierung und nicht des Optimierungsproblems an sich.

Es existieren mehrere Ansätze zur Bestimmung des Ausmaßes der Epistasie. Davidor [Dav90], der den Begriff ursprünglich im Zusammenhang mit Optimierungsproblemen einführte, schlägt als Maßzahl die *Epistasievarianz* vor, die auf dem Abstand der tatsächlichen Zielfunktion f von einer Näherung ξ mit voneinander unabhängigen (kodierten) Entscheidungsvariablen beruht und verwendet dieses Maß als Klassifikationskriterium für die Schwierigkeit eines Problemexemplars (mit gegebener Kodierung) für die Bearbeitung mit einem GA („GA-hardness“).

Er definiert zunächst die *Epistasie ϵ_s eines Lösungsvorschlags s* für ein Problembeispiel als den Abstand des Zielfunktionswertes f(s) der Lösung s vom Wert $\xi(s)$:

$$\epsilon_s = f(s) - \xi(s)$$

$\xi(s)$ berechnet sich aus den Mittelwerten der Fitness-Werte aller Lösungen t, bei denen jeweils eine der L Variablen mit dem gleichen Wert belegt ist, wie im Lösungsvorschlag s. ξ_f ist dann die *Approximation erster Ordnung* der Zielfunktion f, die f in euklidscher Distanz am nächsten ist.[15] Da ähnlich wie bei den unter 4.4.1 beschriebenen Maßen meist nicht alle diese Lösungen zur Berechnung der Epistasie herangezogen werden können, verwendet man statt dessen eine Stichprobe P von Lösungen:

$$\xi(s) = \sum_{i \in L} \frac{1}{|\{t \in P; t_i = s_i\}|} \sum_{t \in P; t_i = s_i} f(t) - \frac{l-1}{|P|} \sum_{t \in P} f(t)$$

Die *Epsistasievarianz $\epsilon_P^2(f(s))$ einer Funktion f* definiert Davidor als die euklidsche Distanz zwischen f und ξ_f:

$$\epsilon_P^2(f) = \frac{1}{|P|} \sum_{s \in P} \epsilon^2(s) = \frac{1}{|P|} \sum_{s \in P} \left((f(s) - \xi_p(s)\right)^2$$

Die Epistasievarianz berechnet sich also als Durchschnitt der quadrierten Abstände der Zielfunktion von ihrer besten Näherung erster Ordnung. Liegt keine Epistasie vor, so spricht man von einer *vollständig separablen Funktion*. In diesem Fall können die Entscheidungsvariablen der Zielfunktion getrennt optimiert werden. Nähere Erläuterungen zur Epistasie, insbesondere zu ihrer Eignung als Maß für die Schwierigkeit von Problemexemplaren sowie zu Kritikpunkten an Davidors Berechnungsmethode finden sich im Anhang (A.6)

Der Anwendungsbereich des hier entwickelten Verfahrens soll Probleme umfassen, die bei intuitiver Kodierung eine gewisse Dekomponierbarkeit der Zielfunktion aufweisen und daher eine sinnvolle partielle Bewertung von Lösungsteilen erlauben. Es ist also von einer relativ geringen Epistasie unter den Variablen verschiedener Lösungsteile auszugehen. Bei Problemen mit Raumbezug, die mit dem entwickelten Verfahren vorrangig bearbeitet werden sollen, zeigen die Variablen meist unterschiedlich starke Interaktionen in Abhängigkeit der Entfernung der jeweils zugrunde liegenden Objekte.

4.4.3 Kaufmanns NK-Fitness-Modell

Das NK-Fitness-Modell dient zur Analyse der Wechselbeziehungen zwischen Genen in der Biologie und wurde von Kauffman [Kau93] eingeführt. Im Modell trägt jeder der N Genorte i (i=1,...,N)

[15] Eine Funktion erster Ordnung kann als Summe partieller Funktionen geschrieben werden, die jeweils nur von einer Variablen abhängig sind. Sie enthält keine Terme höherer Ordnung (vgl. [NSV97], [NK98]).

mit einer eigenen Fitness-Funktion f_i additiv zur Gesamt-Fitness eines Individuums x bei:

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x_i, x_{i_1}, ..., x_{i_K})$$

Die Höhe des Beitrags ist dabei von der Ausprägung des Gens x_i sowie K weiteren, interagierenden Genen $x_{i_1}, ..., x_{i_K}$ abhängig. Die Fitness des Individuums ergibt sich durch die Mittelung der Fitnessbeiträge. Bei der theoretischen Analyse Evolutionärer Algorithmen wird das NK-Fitness-Modell herangezogen, um Kombinatorische Optimierungsprobleme unterschiedlicher „Schwierigkeitsgrade“ zu beschreiben. Während mit dem Parameter N die Problemgröße, d.h. die Anzahl der Entscheidungsvariablen, festgelegt wird, kann mit dem Parameter K die Stärke der Epistasie unter den Variablen eingestellt werden. Meist wird dem Modell eine binäre Kodierung der Individuen zugrunde gelegt. Für Werte von K zwischen 0 und N-1 können Autokorrelation und Korrelationslänge in Abhängigkeit von N und K bestimmt werden (vgl. [Mer00]). Für die Extremfälle $K = 0$ und $K = N - 1$ gilt [Kau93]:

K=0: Das Problem weist nur ein lokales (und damit zugleich globales) Optimum auf. Die Fitness-Landschaft ist eben, die räumliche Autokorellation sowie die FDC sind hoch.

K=N-1: Die Anzahl der zu erwartenden lokalen Optima beträgt $\frac{2^N}{N+1}$. Die Fitness-Landschaft ist extrem zerklüftet, räumliche Autokorellation sowie FDC sind niedrig.

Mit dem NK-Modell lässt sich die Eigenschaft der partiellen Bewertbarkeit von Teillösungen für die Anwendungsbeispiele des zu entwickelnden Optimierungsverfahrens beschreiben: Partiell evaluierbar sind Probleme, die im NK-Modell kleine K-Werte (und damit geringe Epistasie) aufweisen – und bei denen bekannt ist, von *welchen* Variablen die partiellen Fitness-Funktionen f_i jeweils abhängig sind. Die partiellen Fitness-Funktionen des NK-Modells sind mit den partiellen Bewertungsmethoden (vgl. 3.1.2) gleichzusetzen. Eine partielle Bewertung ist ebenfalls nur von einer kleinen Zahl von Variablen abhängig. Sie kann die Güte der Konfiguration eines Objekts messen, ohne das die Konfigurationen *aller* übrigen Objekte in die Berechnung einfließen.

4.5 No-Free-Lunch-Theoreme zu universellen Verfahren

4.5.1 No-Free-Lunch

1995 und 1997 veröffentlichten Wolpert und Macready ihre No-Free-Lunch-Theoreme zu Such- [WM95] und Optimierungsalgorithmen [WM97]. Zentrale Aussage der Theoreme ist, dass *„keine der modernen heuristischen Methoden im Bereich der Optimierung [...] den anderen eindeutig überlegen [ist]“* [Nis97, S.247]. Eine Beweisskizze für die Theoreme findet sich im Anhang (A.7).

Die Performanz des Algorithmus messen Wolpert und Macready in der Anzahl *unterschiedlicher* Lösungsvorschläge, die der Algorithmus benötigt, um zum Optimum zu gelangen [WM97, S.4]. Nach den Theoremen hat jeder Algorithmus, der zur Generierung neuer Lösungen nur die Zielfunktionswerte bisher evaluierter Lösungen verwendet (schwaches Verfahren), genau die gleiche Performanz, wenn man über alle Zielfunktionen mittelt:

> „If we do not take into account any particular biases or properties of our cost function, then the expected performance of all algorithms on that function are exactly the same. In short, there are no ‚free lunches' for effective optimization; any algorithm performs only as well as the knowledge concerning the cost function put into the cost algorithm." [WM95, S.2]

Es ist also nicht möglich, von der guten Performanz eines Algorithmus bei Problemen einer Klasse φ auf die Performanz bei Problemen einer anderen Klasse zu schließen. Da die mittlere Performanz über die Menge alle Zielfunktionen F gleich ist, gilt im Gegenteil:

> „If simulated annealing outperforms genetic algorithms on some set φ, genetic algorithms must outperform simulated annealing on $F\backslash\varphi$." [WM97, S.2]

4.5.2 Kritik und Erweiterungen

Hauptansatz der Kritik an den NFL-Theoremen ist die *Mittelung* über *alle* Optimierungsprobleme. Das entsprechende Szenario des gleich-wahrscheinlichen Auftretens aller möglichen Zielfunktionen ist in der Praxis unrealistisch - und zwar in zweierlei Hinsicht: Erstens werden in realistischen Szenarien nicht *alle möglichen Funktionen* zu optimieren sein, und zweitens werden die zu optimierenden Funktionen mit *unterschiedlichen Wahrscheinlichkeiten* auftreten.

In den letzten Jahren wurden die Theoreme diesbezüglich verallgemeinert. Schumacher, Vose und Whitley [SVW01] zeigen, dass die Aussage der NFL-Theoreme schon dann gilt, wenn die Menge der betrachteten Zielfunktionen unter den Permutationen der möglichen Zielfunktionswerte abgeschlossen ist. Die Bedingung des gleich-wahrscheinlichen Auftretens *aller* Zielfunktionen kann also zugunsten einer *unter Permutationen abgeschlossenen Menge* aufgegeben werden. Der Anteil solcher unter Permutationen abgeschlossenen Funktionenmengen unter den nicht-leeren Funktionenmengen insgesamt ist jedoch sehr klein [IT03]. Insbesondere sind nicht-triviale Nachbarschaftsbeziehungen im Suchraum nicht invariant gegenüber Permutationen. Realistische Funktionen basieren oft auf nicht-trivialen Nachbarschaftsbeziehungen. Koehler [Koe04] zeigt beispielsweise, dass nur triviale Unterklassen von Partitionierungs-, Rucksack- und Rundreiseproblemen unter Permutationen abgeschlossen sind. Igel und Toussaint [IT03] verallgemeinern die NFL-Theoreme so weit, dass in bestimmten Fällen beliebige Wahrscheinlichkeitsverteilungen für die zugrunde gelegten Funktionen angenommen werden können. Ihre Resultate sind jedoch – wie sie selbst zeigen – nur selten anwendbar.

4.5.3 Konsequenzen

Das langjährige Vorgehen vieler Forscher bestand darin, ihre (universellen) Algorithmen auf Mengen spezieller Problembeispiele - so genannter *test beds* - anzusetzen, um sie zu vergleichen und eventuell überlegene universelle Verfahren zu finden. Nach der Veröffentlichung der NFL-Theoreme stellen Whitley und Watson [WW05] dieses Vorgehen nun in Frage:

> „Given the NFL theorems, comparision is meaningless unless we proove (which virtually never happens) or assume (an assumption which is rarely made explicit) that the benchmarks used in a comparision are somehow representative of a particular sub-class of problems." (S. 18)

Durch Vergleiche der Algorithmen mittels Anwendung auf bestimmte Testprobleme kann also – wenn überhaupt – nur die Eignung der Algorithmen für unterschiedliche *Problemtypen*, d.h. Teilmengen der Menge aller Fitnessfunktionen aufgezeigt werden. Die Suche nach universell überlegenen Verfahren muss dagegen scheitern. Anstatt nach überlegenen universellen Verfahren zu suchen bietet es sich daher eher an, charakteristische Merkmale einer Problemklasse (und eventuell des konkret zu optimierenden Problembeispiels) zu identifizieren und im Anwendungsbereich eingeschränkte, spezialisierte Algorithmen zu entwickeln, die möglichst viele dieser Merkmale ausnutzen. Auf diese Weise kann der Zeit (und Platz-) Bedarf des Algorithmus bei der Anwendung auf Probleme dieser Klasse gegenüber universellen Verfahren verringert werden.

Für Algorithmen, die Wissen über die Struktur eines konkreten Problem(exemplar)s in die Produktion neuer Lösungsvorschläge einbeziehen, gelten die NFL-Theoreme nicht. Sofern dieses Wissen sinnvoll ausgenutzt wird, ist die Wahrscheinlichkeit höher, als die einer universellen Heuristik, durch die Kenntnis der Fitness-Werte bisheriger Lösungen eine gute Lösung zu produzieren. Wolpert und McReady nennen Branch-And-Bound-Algorithmen als Beispiel für Algorithmen, für die ihre Theoreme *nicht* gelten: Neben dem Wissen zu bereits evaluierten Lösungen nutzen diese zusätzlich problemspezifische Heuristiken für die Bewertung von Teil- (bzw. Rest-)lösungen [WM97, S.5]. Weil dadurch Bereiche mit „schlechten“ Lösungen bei der Suche ausgeklammert werden können, sind Branch-And-Bound-Algorithmen universellen Heuristiken überlegen.

4.6 Starke Verfahren: Die Integration problemspezifischen Wissens

Starke Optimierungsverfahren sind auf spezielle Optimierungsprobleme oder sogar auf konkrete Problembeispiele zugeschnitten und übertreffen schwache Verfahren in der Regel bei der Optimierung bezüglich Geschwindigkeit und Ergebnisgüte. Im Gegenzug sind sie jedoch nicht universell einsetzbar, d.h. ihre Anwendung auf einige andere Optimierungsprobleme bzw. Problembeispiele ist entweder gar nicht möglich, oder der Algorithmus schneidet hier schlechter ab als universelle Verfahren (vgl. Abschnitt 4.1.1). Für die Bearbeitung des TSP existieren beispielsweise viele problemspezifische Lösungsansätze, einige sind im Anhang zusammengestellt (A.8).

Um im Mittel schneller bessere Lösungen zu finden als ein universelles Verfahren, muss der Algorithmus bei der Generierung neuer Lösungen neben der Fitness bisher evaluierter Lösungen zusätzliches Wissen über das Optimierungsproblem einbeziehen (vgl. NFL, Abschnitt 4.5). Voraussetzung dafür ist natürlich, dass die zu bearbeitende Problemklasse überhaupt eine erkennbare Struktur besitzt und diese dem Anwender auch bekannt ist. Die bloße *Existenz* einer solchen Struktur, so Wolpert und McReady [WM97], rechtfertigt jedoch noch nicht die Wahl eines bestimmten (universellen) Algorithmus:

> „... while most classes of problems will certainly have some structure which, if known, might be exploitable the simple existence of that structure does not justify choice of a particular algorithm; that structure must be known and reflected directly in the choice of algorithm to serve as such a justification. In other words, the simple existence of structure per se, absent a specification of that structure, cannot provide a basis for preferring one algorithm over another.“ (S.9)

Es genügt also beispielsweise nicht, so Wolpert und McReady weiter, die Wahl eines bestimmten Algorithmus damit zu begründen, dass bei den zu bearbeitenden Problemen unterschiedliche Zielfunktionen nicht mit gleicher Wahrscheinlichkeit zu erwarten sind – Wissen über die speziellen Eigenschaften zu optimierender Probleme muss ausgenutzt werden. Viele problemspezifische Verfahren beruhen dazu auf den folgenden drei Prinzipien:

1. Insbesondere für Konstruktionsheuristiken bietet sich die *Eingrenzung des Suchbereichs* basierend auf einer Abschätzung der Güte von (Rest-)Lösungen oder der Auswertung von Nebenbedingungen an. Voraussetzung ist die Kenntnis einer Funktion zur Abschätzung der Güte oder zur Überprüfung der Einhaltung von Nebenbedingungen für *Teillösungen*.

2. Sowohl für Konstruktions- als auch für Verbesserungsheuristiken können *spezielle Konstruktions- oder Variationsoperatoren* bzw. *Nachbarschaftsdefinitionen* Verwendung finden, die Wissen über die Struktur des konkreten Problembeispiels ausnutzen und so das Suchverfahren schnell(er) in vielversprechende Bereiche des Lösungsraums lenken.

3. Die *Dekomposition des Problems* und die Anwendung von Divide-and-conquer-Techniken verspricht dann Erfolg, wenn nicht nur *Lösungen global*, sondern auch *Lösungsteile* (d.h. Belegungen von Teilmengen der Entscheidungsvariablen) *partiell* bewertet werden können – eine Eigenschaft, die möglichst geringe Abhängigkeiten unter den entsprechenden Entscheidungsvariablen voraussetzt (vgl. Abschnitt 4.4.2).

Im Folgenden werden diese Prinzipien genauer beschrieben und es wird untersucht, ob (und wenn ja, wie) sie auch für das im Zuge dieser Arbeit entwickelte Verfahren angewendet werden können.

4.6.1 Eingrenzung des Suchbereichs

Ist es möglich, Zielfunktionswerte bei fester Belegung oder Beschränkung des Werteintervalls einer *Teilmenge* von Entscheidungsvariablen nach oben hin abzuschätzen, so kann dieses Wissen für Konstruktions- und Verbesserungsheuristiken effizienzsteigernd eingesetzt werden.

Bei Verfahren der *impliziten Enumeration* werden Bereiche des Suchraums ausgegrenzt, wenn ausgeschlossen werden kann, dass sich darin das globale Optimum befindet. Das wohl bekannteste implizite Enumerationsverfahren ist das *Branch-and-Bound Verfahren*, das auf Land und Doig [LD60] zurückgeht. Die Bezeichnung Branch-and-Bound wurde von Little, Murty, Sweeney und Karel [LMSK63] in Verbindung mit ihrem Algorithmus zur Bearbeitung von TSP-Exemplaren geprägt. Beim Branch and Bound wird der der Suchraum durch einen *Suchbaum* repräsentiert. Jeder Knoten im Baum entspricht dabei einer Entscheidung für die Belegung einer bestimmten Entscheidungsvariablen, ein Pfad von der Wurzel bis zu einem Knoten somit einer Teillösung des Problems. Die Suche nach einem günstigsten Pfad von der Wurzel des Baumes bis zu einem Blatt korrespondiert somit mit einem Konstruktionsverfahren für die schrittweise Erstellung einer optimalen Lösung. Ausgehend von einer bisherigen Teillösung und einer bisher besten Gesamtlösung werden mit Hilfe von *Schätzfunktionen* jeweils untere Schranken für die Kosten der Lösung des Restproblems berechnet. Die Summe dieser unteren Schranke und der Kosten der bisherigen Teillösung werden dann mit den Kosten der momentan besten gefundenen Gesamtlösung verglichen. Teilbäume werden abgeschnitten, wenn darin enthaltene Gesamtlösungen (selbst bei niedrigstmöglichen Restkosten) die Kosten der momentan besten Gesamtlösung

nicht mehr unterschreiten können. Branch-and-Bound-Verfahren sind daher nur anwendbar, wenn Kosten für Teillösungen nach unten abgeschätzt werden können, wie dies etwa beim TSP der Fall ist. Implizit wird der *gesamte* Lösungsraum durchsucht, es handelt sich daher um exakte Verfahren (sofern gewährleistet ist, dass die Kosten für Restlösungen nie *überschätzt* werden). Einige Approximationsverfahren beruhen ebenfalls auf dem Prinzip, den Suchbereich schrittweise einzugrenzen, weil bestimmte Bereiche des Suchraums kein globales Optimum (mehr) enthalten können.

Grenzen *Nebenbedingungen* den Raum gültiger Lösungen stark ein, so bietet es sich eventuell an, auf Techniken der *Constraint Satisfaction* zurückzugreifen. Diesen Ansatz verfolgt beispielsweise Bruns [Bru96] mit seinem wissensbasierten Genetischen Algorithmus GAP. Er verknüpft einen GA mit speziellen Variationsoperatoren für die Bearbeitung von Constraint-Satisfaction-Problemen. Sein Algorithmus GAP wird in Abschnitt 4.8 ausführlicher dargestellt, da es sich um einen hybridisierten Genetischen Algorithmus handelt.

Das im Zuge dieser Arbeit entwickelte Verfahren beruht nicht auf der Eingrenzung des Suchbereichs, da der Anwendungsbereich auch Probleme umfasst, bei denen eine solche Einschränkung mangels Kenntnis einer entsprechenden Schätzfunktion für die Güte potenzieller Restlösungen nicht möglich ist. Durch Verwendung spezieller Variationsoperatoren kann jedoch ausgeschlossen werden, dass das Verfahren Lösungen produziert, die gegen Nebenbedingungen verstoßen.

4.6.2 Verwendung spezieller Operatoren und Nachbarschaftsdefinitionen

Bei starken Verfahren kommen häufig problemspezifische Konstruktions- oder Variationsoperatoren zum Einsatz, die spezielles Wissen über konkrete Problemstruktur, Belegung von Problemparametern oder Nebenbedingungen ausnutzen.

Problemspezifische Konstruktionsheuristiken wenden häufig eine so genannte *Greedy-Strategie* an. Sukzessive werden Entscheidungsvariablen so belegt, dass sich zusammen mit den bereits vorher belegten Entscheidungsvariablen eine möglichst gute momentane (Teil)lösung ergibt, d.h. die prognostizierte Güte der Gesamtlösung maximiert wird. Hier kann ebenso wie bei Branch-and-Bound und Divide-and-Conquer-Techniken Anwenderwissen über die *Güte von Teillösungen* ausgenutzt werden, das universelle Verfahren vernachlässigen.

Beim im Zuge dieser Arbeit entwickelten Verfahren handelt es sich nicht um ein Konstruktions- sondern um ein Verbesserungsverfahren: Es kann nur auf eingeschränktes Anwenderwissen zurückgegriffen werden, das unter Umständen keine ausreichend guten Schätzfunktionen für Restlösungen sowie Operatoren zur Konstruktion von Lösungen umfasst, um ein effizientes Kostruktionsverfahren zu realisieren. Dennoch können auch hier Greedy-Strategien angewendet werden, um beispielsweise zu entscheiden, *welche* Lösungsteile verändert werden sollen, um möglichst schnell zu Verbesserungen zu kommen.

Auf Seiten der Verbesserungsheuristiken wurden gerade für die bekannten Optimierungsprobleme wie das TSP viele Operatoren entwickelt, die die Suche nach dem Optimum durch Ausnutzen der speziellen Struktur – etwa von Rundreiseproblemen – und durch Ausnutzung der Kenntnisse über das konkrete Problembeispiel – z.B. der Entfernungen zwischen den Städten – zu beschleunigen versuchen. Für metrische TSP kommen beispielsweise Operatoren in Frage, die Überkreuzungen in Rundreisen erkennen und auflösen oder Operatoren, die in problemspezifisch definierten Nachbarschaften wie der k-change-Nachbarschaft nach neuen, möglicherweise besseren Lösungen

suchen. Für die Rekombination bieten sich Reihenfolge-erhaltende Operatoren („Order Based Crossover“) an, wie sie beispielsweise in [Dav91] beschrieben sind. Auch beim hier entwickelten Verfahren sollten Möglichkeiten für die Integration problemspezifischer Operatoren geschaffen werden, da der Anwender oft über entsprechendes Wissen verfügt (vgl. Abschnitt 3.1.2).

4.6.3 Partielle Bewertung und Dekomposition

Für die Lösung vieler Problemstellungen kann das *Divide-and-conquer-Prinzip* (Prinzip „Teile und herrsche“) angewendet werden. Dabei wird zunächst versucht, das Problem in kleinere (und dadurch in der Regel leichter zu optimierende) *Teilprobleme* zu dekomponieren, die dann einzeln gelöst werden und zusammengesetzt die Lösung des Gesamtproblems ergeben (vgl. Abschnitt 4.4.2). Die Teilprobleme werden dabei oft über mehrere „Stufen“ durch die gleiche Technik bearbeitet, so dass der Algorithmus rekursiv definiert werden kann [Hro03]. Ein Beispiel für Divide-and-conquer-Verfahren ist der Sortieralgorithmus *Quicksort*, bei dem die zu sortierende Liste rekursiv in Teillisten zerlegt wird, die dann sortiert und anschließend zu einer Gesamtlösung zusammengesetzt werden. Sortierprobleme stellen eine Unterklasse der Optimierungsprobleme dar. Auf diese Klasse spezialisierte Algorithmen wie *Quicksort* sortieren im Durchschnitt schneller als universelle Optimierungsverfahren, sind dafür aber für andere Probleme nicht geeignet.

Divide-and-Conquer-Verfahren bieten sich für Optimierungsprobleme dann an, wenn es möglich ist, die Entscheidungsvariablen so in Gruppen einzuteilen, dass der Einfluss von Belegungen aus einer Gruppe auf die Zielfunktion unabhängig von Belegungen aus anderen Gruppen ist, d.h. keine externe Epistasie unter den Gruppen auftritt (vgl. Abschnitt A.6). In diesem Fall kann ein globales Optimum des Gesamtproblems aus der Kombination der jeweils optimalen Belegung der Teilprobleme konstruiert werden. Notwendig ist dazu eine Aufspaltung der Zielfunktion in partielle Bewertungsfunktionen, die jeweils die Güte einer Teillösung bestimmen.

Leider ist eine vollständige Dekomposiotion eines Problemexemplars in Teilprobleme bei der Bearbeitung von KOP im Allgemeinen nicht möglich, da keine komplett unabhängigen Variablengruppen gebildet werden können. Dennoch kann versucht werden, das Problem durch die Kombination guter Teillösungen (d.h. Belegungen disjunkter Variablengruppen) zu optimieren, wenn das Problem(exemplar) *gewisse Dekomponierbarkeit* (vgl. Abschnitt 4.4.2) aufweist und entsprechende Funktionen zur Bewertung von Teillösungen bekannt sind. Rekombinationsoperatoren dienen dann zur Kombination von Lösungsteilen.

Die Dekomposition von Problem(exemplar)en spielt für das hier entwickelte Verfahren in zweierlei Hinsicht eine Rolle: Zunächst soll es – wie bei der Bearbeitung von Hand – möglich sein, lokales Verbesserungspotential von Lösungs*teilen* aufzuspüren und entsprechende Belegungen von Entscheidungsvariablen zielgerichtet zu modifizieren. Die Erkennung von Verbesserungspotential basiert dabei auf einer partiellen Bewertung der Lösungsteile, die in gewisser Hinsicht wiederum auf einer Zerlegung (*Dekomposition*) des Problems in einzelne Variablen bzw. Variablengruppen beruht. Andererseits sollen vom Anwender definierte, problemspezifische Rekombinationsoperatoren zum Einsatz kommen können, die die Lösungsteile zu Gesamtlösungen zusammensetzen. Dabei sollen ebenfalls partielle Bewertungen von Lösungsteilen herangezogen werden. Die partiellen Bewertungen werden von den *Agenten* eines *Multiagentensystems* übernommen. (Multi-)Agentensysteme sowie Möglichkeiten, mit diesen KOP zu bearbeiten, werden im nächsten Abschnitt eingeführt.

4.7 Multiagentensysteme

Das im Zuge dieser Arbeit entwickelte Optimierungsverfahren basiert auf einem GA, der mit einem so genannten *Multiagentensystem* (MAS) verknüpft wird. MAS bilden also gewissermaßen das zweite „Standbein“ des Verfahrens.

Die Forschung über MAS begann vor etwa 25 Jahren im Bereich der Verteilten Künstlichen Intelligenz (engl. Distributed Artificial Intelligence, DAI) und ist heute Hauptbestandteil dieses interdisziplinären Forschungszweiges, wie die Definition von Weiss im Prolog von [Wei99] verdeutlicht (S.1):

> „*DAI is the study, construction, and application of multiagent systems, that is, systems in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks.*“

Ein Multiagentensystem (MAS) besteht aus einer Menge *interagierender* autonomer Komponenten, genannt „Agenten“.[16] Die bloße Existenz mehrerer Agenten („Mehr-Agenten-System“, vgl. [Hes02]) reicht für die Klassifikation eines Systems als MAS nicht aus, zusätzlich wird meist gefordert, dass die Agenten in irgendeiner Form miteinander in Interaktion treten.

Agenten können dann als „interagierend“ bezeichnet werden, wenn die Aktionen, die sie zum Erreichen ihrer Ziele durchführen, beeinflusst werden durch Aktionen anderer Agenten oder die Kommunikation mit anderen Agenten. Dazu können die Agenten explizit z.B. über eine bestimmte Sprache (*agent communication language* [GK94]) miteinander kommunizieren.

Eine andere Möglichkeit besteht in der impliziten Kommunikation durch Beobachtung anderer Agenten oder deren Aktionen in einer gemeinsamen Umgebung. Bei so genannten *Blackboard Systemen*, deren Aufbau und Funktionsweise beispielsweise Corkill in [Cor91] beschreibt, findet die Kommunikation unter den Agenten (*knowledge sources*) *ausschließlich* über ein gemeinsames Wissensmedium, dem *blackboard* statt. Auf das *blackboard* steht zunächst die Beschreibung eines gemeinsam zu bearbeitenden Problems. Jeder Agent repräsentiert einen unabhängigen *Experten* für einen bestimmten Aspekt der Problemlösung und kann seine spezifische Fachkenntnis einbringen, in dem er eigene Beiträge (beliebiger Art) auf das *blackboard* schreibt. Ein Beitrag kann dabei Informationen beinhalten, die wiederum andere Agenten für eigene Beiträge verwenden. Auf diese Weise soll das Gesamtproblem nach und nach durch die Einzelbeiträge der Agenten gelöst werden. Charakteristisch für das *Blackboard*-Modell ist, dass die Agenten nie direkt miteinander in Interaktion treten.

Die Kontrolle ist in Multiagentensystemen typischerweise auf die Agenten dezentral verteilt, es existiert meist keine zentrale Instanz, die in Abhängigkeit des (Gesamt-)Systemzustandes Entscheidungen für die einzelnen Agenten trifft. Bei *Blackboard-Systemen* wird dagegen eine Kontrollkomponente benötigt die die Koordination der Beiträge der Agenten übernimmt [Cor91].

Besonderes Augenmerk wird bei der Forschung an MAS auf die Koordination der Handlungen der Agenten gelegt, die sie zum Erreichen ihrer im Allgemeinen unterschiedlichen und eventuell inkompatiblen Ziele durchführen [Wei99, Prolog S. 3]. Ebenso ist es möglich, Agenten zur Lösung einer gemeinsamen Aufgabe kooperieren zu lassen (*Kooperatives MAS*) [Vla03]. Genauso wie die

[16] Eine Zusammenstellung grundlegender Begriffe aus der Agententheorie findet sich im Anhang (A.9)

Kontrolle ist auch das Wissen über den Systemzustand in MAS oft verteilt, es existieren jedoch auch MAS mit gemeinsamer Wissensbasis.

Auch in Multiagentensystemen können Agenten lernen. Sen und Weiss [SW99] unterscheiden hier zwischen „*centralized learning*“ und „*decentralized learning*“ je nachdem, ob der Lernprozess nur von einem einzelnen Agenten ausgeführt wird und keine Interaktionen mit anderen Agenten notwendig sind („*centralized*“) oder ob verschiedene Agenten am Lernprozess beteiligt sind („*decentralized*“). Lernen in MAS kann beispielsweise dann sinnvoll sein, wenn das System großen, offenen Umgebungen ausgesetzt ist, bei denen der Nutzen bestimmter Aktionen nicht *a priori* vorhersehbar ist. In diesem Fall können die Agenten eventuell „selbstständig“ lernen, welche Aktionen in welchen Situationen auszuführen sind, um in Zukunft größtmöglichen Nutzen zu erzielen und sich so möglichst „rational“ zu verhalten.

Verschiedene Lernmethoden unterscheiden sich dabei unter anderem in der Art und Weise, wie der Nutzen bzw. Schaden bestimmter Aktionen, die ein Agent durchgeführt hat, ermittelt wird, und wie diese Informationen zur Verbesserung zukünftiger Aktionen eingesetzt werden. Von „*reinforcement learning*“ spricht man, wenn den Agenten nach der Durchführung von Aktionen regelmäßig ein „Feedback“ - auch „*reinforcement*“ genannt - übermittelt wird. Das Feedback spezifiziert den Nutzen durchgeführter Aktionen. Ziel eines Agenten ist es, seine zukünftigen Aktionen so zu wählen, dass dieser Nutzen maximiert wird [RN03]. Betrachtet man nur einen einzelnen Agenten, so besteht die Schwierigkeit darin, den Beitrag der im Einzelnen durchgeführten Aktionen zu diesem Feedback zu ermitteln. Man spricht in diesem Zusammenhang vom „*Credit Assignment Problem (CAP)*“. In MAS tritt dieses Problem sogar auf zwei Ebenen auf, da das Feedback eine Aggregation des Nutzens der Aktionen nicht nur eines einzelnen sondern mehrerer Agenten darstellt [SW99]. Dieses Problem kann jedoch umgangen werden, wenn ausgeschlossen wird, dass mehrere Agenten gleichzeitig Aktionen durchführen und wenn nach jeder einzelnen Aktion ein Feedback übermittelt wird. In Multiagentensystemen stellt sich zusätzlich die Frage, inwieweit Agenten nicht nur von den Ergebnissen eigener Aktionen, sondern auch von den Aktionen anderer Agenten lernen können. Gerade wenn die Agenten ähnlich aufgebaut sind, ähnliche Aktionen durchführen können und ähnliche Ziele verfolgen, kann es sinnvoll sein, auch das Feedback, das andere Agenten erhalten haben, in die eigene Aktionsauswahl einfließen zu lassen. Dies kann durch explizite Kommunikation unter den Agenten erreicht werden oder mit Hilfe einer zentralen Wissensbasis, die allen Agenten zur Verfügung steht, und in der Daten zum Nutzen bestimmter Aktionen in verschiedenen Situationen festgehalten werden.

Der Einsatz von MAS bietet sich insbesondere dann an, wenn eine komplexe Aufgabe in einfachere, möglichst unabhängige Teilaufgaben dekomponiert werden kann, die – möglicherweise parallel – bearbeitet werden können. Vorteile eines MAS liegen dabei in folgenden Punkten [Vla03]:

- Steigerung von Geschwindigkeit und Effizienz des Systems durch Parallelisierung.
- Flexible Skalierbarkeit durch die Möglichkeit der Aufnahme zusätzlicher Agenten.
- Eventuelle Kosteneinsparungen durch Reduzierung der Komplexität.
- Erhöhung der Wiederverwendbarkeit und der Transparenz durch modularen Aufbau.
- Erhöhung der Robustheit und Verlässlichkeit des Systems, wenn das Gesamtsystem Fehler einzelner Agenten toleriert.

Die wichtigsten Anwendungsgebiete von Multiagentensystemen liegen in der Simulation komplexer Systeme (beispielsweise menschlicher Gesellschaften) zu Analyse- und Prognosezwecken[17], der Bearbeitung als Stellvertreter für Menschen (etwa bei Computerspielen oder Auktionen im Internet) sowie in der verteilten Problemlösung[18]. Die Möglichkeit, durch ein MAS Probleme verteilt zu lösen, soll auch in der vorliegenden Arbeit ausgenutzt werden. Hier sollen *Optimierungsprobleme* kooperativ und verteilt bearbeitet werden.

4.7.1 MAS zur verteilten Bearbeitung von Optimierungsproblemen

Verteiltes Problemlösen (engl. *Distributed Problem Solving*) ist ein Teilbereich der Verteilten Künstlichen Intelligenz. Hier wird der Frage nachgegangen, wie Agenten in MAS kooperieren können, um gemeinsam übergeordnete Probleme zu lösen [Dur99]. Verteiltes Problemlösen mit MAS bietet sich natürlich vor allem dann an, wenn sich das zu bearbeitende Gesamtproblem in Teilprobleme dekomponieren lässt. Dann können diese Teilprobleme jeweils einem oder mehreren spezialisierten Agenten zur Lösung „überlassen“ werden. Wenn weder die Güte einer Teillösung von anderen Teillösungen abhängt, noch Teillösungen sich gegenseitig ausschließen, können die Teilprobleme sogar komplett parallel bearbeitet und die resultierenden Teillösungen zur Gesamtlösung zusammengesetzt werden (vgl. Abschnitt 4.4.2).

Verteilte Bearbeitung von Optimierungsproblemen

Für diese Arbeit interessiert vorrangig, wie Probleme der *kombinatorischen Optimierung* mit Techniken der verteilten Problemlösung sinnvoll bearbeitet werden können. Zwei grundsätzlich verschiedene Ansätze dazu werden in den folgenden Abschnitten näher vorgestellt:

- **Jeder Agent repräsentiert einen Lösungsvorschlag:**
  Dieser Ansatz wird beispielsweise bei *Ameisenalgorithmen* angewendet. Hier versuchen „voneinander lernende“ Agenten Problemexemplare durch iterative Anwendung einer Konstruktionsheuristik zu lösen. Dabei beruht die Heuristik einerseits auf Problemwissen sowie andererseits auf Erfahrungen der Agenten aus vorherigen Lösungsversuchen, die untereinander ausgetauscht werden. Ameisenalgorithmen werden in Abschnitt 4.7.2 näher erläutert.

- **Jeder Agent repräsentiert eine oder mehrere Entscheidungsvariable(n):**
  Beim zweiten Ansatz, der auch bei dieser Arbeit verfolgt wird, werden nicht die Lösungsvorschläge sondern die *Entscheidungsvariablen* von Agenten repräsentiert. Die Agenten versuchen, durch lokale Verbesserung der Belegung individuell verwalteter Variablen global gute Lösungen zu erreichen.

Einige andere Ansätze für die Bearbeitung Kombinatorischer Optimierungsprobleme mit Multiagentensystemen sollen hier ebenfalls erwähnt werden.

Bei der *Teilchenschwarmoptimierung* (*Particle Swarm optimization*), die Kennedy und Eberhart [KE95] einführen, repräsentiert wie bei Ameisenalgorithmen jeder Agent eine Lösung. Die

[17] Anwendungen des Autors stammen beispielsweise aus der geographischen Handelsforschung [Hes02] oder der Bevölkerungsgeographie [RHS04].

[18] Einige industrielle Anwendungen sind beispielsweise in [Par99] erläutert.

Agenten bewegen sich ähnlich den Individuen eines Vogel- oder Insektenschwarms im Suchraum: Sie passen ihre Bewegungsrichtung und -geschwindigkeit an die ihrer Nachbarn an und kommunizieren untereinander über die Güte bereits gefundener Lösungen. Das Verfahren ist – jedenfalls in seiner ursprünglichen Form – problemunabhängig, es wird kein zusätzliches Problemwissen ausgenutzt.

Die Optimierung der Verteilung bestimmter Ressourcen kann durch *virtuelle Marktplätze* modelliert werden. Gaspero et al. [GMS04] zum Beispiel beschreiben eine Multiagenten-Architektur zur verteilten Lösung von Stundenplanproblemen. Ziel ist es, bei der gleichzeitigen Erstellung der Stundenpläne *mehrerer Fakultäten* Ressourcen, d.h. Raumkapazitäten untereinander auszutauschen. Dazu werden jeder Fakultät drei kooperative Agenten zugeordnet. Ein *Solver* generiert Lösungen für ein einzelnes Department. Ein *Negotiator* nimmt an einem virtuellen Marktplatz (*RoomSlot MarketPlace, RSMP*) teil und versucht, ungenutzte Ressourcen zu „verkaufen“ bzw. zur Verbesserung der derzeitigen Lösung zusätzlich benötigte Ressourcen von anderen Departments zu „erwerben“. Ein *Manager* verwaltet die Preisangebote für die benötigten Ressourcen.

Agenten können auf unterschiedlichen *Ebenen der Problembearbeitung* eingesetzt werden. Milano und Roli [MR04] beschreiben beispielsweise die MAS-Architektur MAGMA (MultiAGent Metaheuristics Architecture), ein Framework für Metaheuristiken. Hier wird eine Aufgabentrennung unter den Agenten auf drei oder vier verschiedenen Ebenen vorgenommen: Agenten auf Ebene 0 bilden Lösungen, Agenten der Ebene 1 verbessern diese und Agenten auf Ebene 2 lenken den Suchprozess in vielversprechende Regionen. Zusätzliche Level-3-Agenten können außerdem für die Koordination *verschiedener* Metaheuristiken (Level-2-Agenten) sorgen, so dass diese beliebig miteinander kombiniert werden können.

4.7.2 Ameisenalgorithmen: Agenten repräsentieren Lösungsvorschläge

Ameisenalgorithmen gelten wie EA oder auch SA als naturanaloge Optimierungsverfahren. Bei der Untersuchung einer Ameisenkolonie stellt man fest, dass Ameisen auf Ihren Wegen Duftstoffe (Pheromone) hinterlassen. Nachfolgende Ameisen bevorzugen Wege mit hoher Pheromonkonzentration. Auf diesem Prinzip der Stigmergie (Kommunikation durch Veränderung der Umwelt) und Autokatalyse (selbstbeschleunigender Prozess) baut das Brückenexperiment von Deneubourg [DAGP90] auf, das als Ausgangspunkt für die Entwicklung der Ameisenalgorithmen gesehen werden kann. Im Experiment ist ein Ameisennest durch eine Brücke mit zwei Pfaden A und B von einer Futterstelle entfernt. Bisher haben i Ameisen die Brücke überquert, davon A_i Ameisen über Pfad A und B_i über Pfad B. Es zeigt sich, dass sich die Wahrscheinlichkeit P_A, dass die (i+1)-te Ameise den Pfad A wählt modellhaft berechnen lässt als:

$$P_A = \frac{(k + A_i)^n}{(k + A_i)^n + (k + B_i)^n} = 1 - P_B$$

Die Nichtlinearität der Attraktivitätssteigerung eines Weges durch die Anzahl vorausgegangener Ameisen wird durch den Exponent n modelliert, der Parameter k ist nötig, um auch den bisher pheromonfreien Wegen eine gewisse Attraktivität zuzuordnen und so das explorative Verhalten der Ameisen abzubilden.

Besonders interessant sind die Ergebnisse des Experiments, wenn die Pfade unterschiedliche Länge haben. Die Ameisen können den Weg vom Nest zur Futterstelle auf dem kürzeren Pfad

schneller zurücklegen, deshalb steigt nach einer kurzen, ausgeglichenen Phase die Pheromonkonzentration auf dem kürzeren Pfad schneller an und Ameisen wählen dann bevorzugt diesen kürzeren Pfad. Dadurch sind Ameisen dazu in der Lage, Netze aus jeweils näherungsweise kürzesten Wegen zwischen Nestern und Futterstellen zu bilden, obwohl keine der Ameisen über explizites, vollständiges Wissen verfügt.

AS und ACS

Die Ergebnisse dieses Experiments inspirierten Dorigo [DMC91, Dor92] dazu, das TSP mit Hilfe von Algorithmen zu bearbeiten, die das Verhalten der Ameisen simulieren (so genannte *Ameisenalgorithmen*): Virtuelle Ameisen, die als Agenten bezeichnet werden können, versuchen ein gegebenes Wegsucheproblem schrittweise individuell zu lösen, wobei sie Hinweise („Pheromone“) für die übrigen Ameisen hinterlassen. Die von Dorigo entwickelten Varianten von Ameisenalgorithmen werden als *Ant System (AS)* bezeichnet und laufen wie folgt ab: Zunächst werden die Ameisen auf die Städte verteilt. Ausgehend von einer Stadt besucht eine Ameise als nächstes eine andere noch nicht besuchte Stadt mit einer Wahrscheinlichkeit, die abhängig ist von der Entfernung zwischen den beiden Städten („Sichtbarkeit“) sowie der Anzahl der Ameisen, die den Weg zwischen den Städten vorher bereits „mit Erfolg“ zurückgelegt haben („Pheromone“).

Das Verfahren nutzt also im Voraus bekannte Informationen über das Problemexemplar (*a priori-Informationen*) sowie im Verlauf der Optimierung gesammelte Hinweise über die Güte von Teillösungen (*a posteriori-Informationen*). Die drei von Dorigo vorgeschlagenen Varianten unterschieden sich dabei in der Ermittlung der Menge der pro Ameise und besuchter Kante zu verteilenden Pheromone. Diese ist entweder konstant (*Ant-density*), umgekehrt proportional zur Länge der Kante (*Ant-quantity*) oder umgekehrt proportional zur Qualität der gesamten Rundreise (*Ant-cycle*) [DMC96]. Pheromone werden von den Ameisen direkt nach dem beschreiten einer Kante (*step by step*) und/oder nach Vervollständigung einer Rundreise verteilt (*online delayed pheromone update*, z.B. bei der *Ant-cycle*-Variante). Die Pheromone „verdunsten“ über die Zeit hinweg, um frühzeitige Konvergenz des Verfahrens zu vermeiden und die Exploration neuer Bereiche im Suchraum zu fördern [DC99]. Nach und nach entsteht eine Rundreise aus Kanten mit besonders hoher Pheromonkonzentration, die von (fast) allen Agenten „abgeschritten“ wird. Diese Rundreise liefert das Verfahren zurück.

Der weiterentwickelte Ameisenalgorithmus *ACS* (*Ant Colony System*) von Dorigo und Gamberdella [DG97] beschleunigt den Konvergenzvorgang durch eine andere Übergangsfunktion und zielgerichtetere Aktualisierung der Pheromonspuren. Eine andere Weiterentwicklung ist der *MMAS*-Ameisenalgorithmus (*Max-Min Ant System*) von Stützle und Hoos [SH00].

Ameisenalgorithmen zeigen besonders für kleine Exemplare des TSP meist vergleichbare oder bessere Ergebnisse als allgemeine heuristische Verfahren. Das überrascht nicht, da sie zusätzliches Problemwissen verwenden: Die Agenten wählen die nächste zu besuchende Stadt unter anderem in Abhängigkeit der „Sichtbarkeit“ aus, d.h. die Entfernungen der noch nicht besuchten Städte vom aktuellen Standpunkt aus gehen in die Lösungsfindung ein. Ameisenalgorithmen können jedoch nicht nur auf das TSP, sondern auch auf andere Optimierungsprobleme angewendet werden. Dazu zählen nicht nur Rundreise- und Routingprobleme (TSP, VRP usw.) sondern beispielsweise auch Quadratische Zuordnungsprobleme, Stundenplanprobleme und Constraint-Satisfaction-Probleme – eine Übersicht findet sich beispielsweise in [DC99] und [DS03]. In die Entscheidungsfindung der Agenten geht in der Regel problemspezifisches *a-priori*-Wissen mit ein. Ameisenalgorithmen sind insofern als *problemspezifische* Heuristiken zu bezeichnen.

Einordnung von Ameisenalgorithmen, Relevanz für die Arbeit

Ameisenalgorithmen stellen ein *Multiagentensystem* (Kapitel 4) dar, bei dem jeder Agent (jede Ameise) parallel nach einer eigenen guten Lösung sucht. Dabei kommunizieren die Agenten über (virtuelle) Pheromone und kooperieren so implizit bei der Suche nach der besten Lösung. Die Platzierung von Pheromonen anhand des Feedbacks nach der Vervollständigung einer Lösung (*Ant cycle*) sowie die entsprechende Anpassung der Wahrscheinlichkeitsfunktionen der Agenten im nächsten Durchlauf kann auch als „*Lernprozess*“ des Gesamtsystems verstanden werden. Man könnte Ameisenalgorithmen als iterative, populationsbasierte *Konstruktionsheuristiken* bezeichnen. Jede Ameise konstruiert sukzessive eine Gesamtlösung aus Teillösungen. Dabei werden bereits gefundene gute Teillösungen (Teilwege mit hoher Pheromonkonzentration) bevorzugt.

Für die vorliegende Arbeit interessant sind Ameisenalgorithmen, da hier Agentensysteme zur Lösung von Optimierungsproblemen eingesetzt werden die *a priori*-Informationen über konkrete Problemexemplare in Entscheidungssituationen zur Aktionsauswahl heranziehen: Durch die Einbeziehung der Entfernungen zwischen den Städten („Sichtbarkeit“) wird *zusätzliches Wissen* über das Problembeispiel in den Algorithmus eingebracht. Implizit wird dabei wiederum angenommen, dass sich gute Gesamtlösungen wenigstens ansatzweise aus der Kombination guter Teillösungen (Nacheinander-Aufsuchen von benachbarten Städten) ergeben.

4.7.3 „Greedy-MAS“: Agenten repräsentieren Entscheidungsvariablen

Ein alternativer Ansatz für die Bearbeitung von KOP durch MAS liegt darin, nicht die Lösungsvorschläge, sondern die *Entscheidungsvariablen* von Agenten verwalten zu lassen. Sycara et al. [SRSF91] beschreiben mit *DCHS (Distributed Constrained Heuristic Search)* einen solchen Ansatz – allerdings nicht zur Bearbeitung allgemeiner Kombinatorischer Optimierungsprobleme, sondern speziell für Constraint-Satisfaction-Probleme (CSP):

> „A distributed CHS problem is a problem where the variables are distributed among a set of agents. Each agent is responsible for a set of variables and their instantiation. [...] Agents make local decisions about assignments of values to particular variables at particular times during problem solving and a complete solution is formed by incrementally merging partial solutions.“ (S. 3f)

CSP stellen einen Spezialfall von Optimierungsproblemen dar, bei denen nur nach einer gültigen Lösung gesucht wird (konstante Zielfunktion). Sie eignen sich besonders für die Bearbeitung durch ein verteiltes (Agenten-)System, wenn die Nebenbedingungen jeweils nur eine begrenzte Zahl von Variablen betreffen. Dann können die Variablen auf die Agenten verteilt werden[19], und die Agenten können einen Lösungsvorschlag hinsichtlich der Verletzungen „ihrer“ Variablen *partiell* bewerten. Bei der Bearbeitung eines CSP mit einem Verbesserungsverfahren (*Iterative Improvement*) wird dadurch eine zielgerichtetere Suche nach einer besseren Lösung ermöglicht, als wenn nur die *Anzahl* der verletzten Nebenbedingungen *insgesamt* betrachtet wird. Ein Überblick zu Verfahren zur verteilten Lösung von Constraint Satisfaction Problemen findet sich in [YH00].

[19] Alternativ können auch partielle Constraintnetze, d.h. größere Teile des Gesamtnetzes jeweils einzeln durch einen Agenten bearbeitet und dann zur Gesamtlösung zusammengesetzt werden. Einen solchen Ansatz verfolgt beispielsweise Baumgärtel [Bau99] bei der verteilten Planung in einer Fließfertigung.

Der Ansatz, Entscheidungsvariablen durch Agenten verwalten zu lassen, kann auch bei der Bearbeitung von KOP mit nicht-konstanter Zielfunktion verfolgt werden. Grundsätzliche Idee dabei ist wiederum, Lösungsvorschläge nicht nur global zu evaluieren, sondern durch die Agenten individuelle, *partielle* Bewertungen durchzuführen zu lassen, mit denen jeweils die *Güte* der Belegung(en) der Entscheidungsvariable(n) gemessen wird, die ein Agent verwaltet. Individuelles Ziel der Agenten ist die Maximierung ihres partiellen Gütemaßes. Mit den partiellen Bewertungsfunktionen wird also gewissermaßen die „Zufriedenheit“ der Agenten gemessen. Die partielle Bewertungsfunktion ist natürlich von der Belegung der Variablen abhängig, die der Agent verwaltet. Haben die Belegungen der übrigen Variablen keine oder nur geringe Auswirkungen auf die Funktion (geringe Epistasie), dann kann der Agent versuchen, seine individuelle Zufriedenheit durch Variation der Belegung „seiner“ Variablen zu steigern. Ein MAS für die Bearbeitung von Optimierungsproblemen, bei dem die beteiligten Agenten auf diese Weise arbeiten, soll im Folgenden als „*Greedy-MAS*“ bezeichnet werden. Damit soll zum Ausdruck kommen, dass die Agenten jeweils (Teil-)lösungen anstreben, die die *individuell* größte Zufriedenheit versprechen.

Die Hoffnung bei der Anwendung eines solchen Greedy-MAS ist, dass eine gleichzeitige (möglicherweise parallele) Steigerung der Zufriedenheit der Agenten möglich ist, d.h. dass durch die *Maximierung der partiellen Bewertungen* schnell auch *insgesamt gute Gesamtlösungen* entstehen.

Polygongeneralisierung mit einem Greedy-MAS

Der Aufbau und Ablauf eines solchen Greedy-MAS soll durch ein Beispiel verdeutlicht werden: Galanda [Gal03] beschreibt einen Algorithmus zur Generalisierung von Polygonen für thematische und topographische Karten. Die Aufgabenstellung besteht darin, beim Übergang auf größere Maßstäbe die Übersichtlichkeit der Darstellung in der Karte beizubehalten. Dabei soll gleichzeitig der Informationsverlust minimiert werden.

Der Algorithmus muss dabei diverse Distanz- und Größenkonflikte lösen, die durch den Maßstabsübergang auftreten. Die Karte im Zielformat soll beispielsweise keine zu kleinen oder zu nahe beieinander liegenden Polygone enthalten, andererseits soll die Veränderung oder das Verdrängen von Polygonen nicht dazu führen, dass die Karte „zu ungenau“ wird.

Das Problem der Polygongeneralisierung ist zwar eigentlich wiederum als CSP einzuordnen, da nach einer Lösung gesucht wird, bei der alle der oben beschriebenen Bedingungen (constraints) eingehalten werden. Hier soll das Problem jedoch als KOP aufgefasst werden, bei dem die Zielfunktion die Güte einer Lösung hinsichtlich der Einhaltung der Bedingungen misst (vgl. [Gal03, S. 30f]). Gegenstand der Optimierung ist die Lage und Größe aller realen Objekte auf der Karte im Zielmaßstab, die Güte einer Lösung ergibt sich durch Aggregation von Maßzahlen, die den *Grad der Übertretung* der Bedingungen bestimmen.

Der Algorithmus, den Galanda beschreibt, verwendet ein Multiagentensystem mit Agenten unterschiedlicher Hierarchieebenen. Auf der untersten Ebene finden sich *line agents*, die einzelne Linien von Polygonen repräsentieren. *Polygon agents* fassen jeweils eine Menge solcher *line agents* zusammen. Die *polygon agents* werden wiederum in *group agents* zusammengefasst, die zusammen gehörende Gruppen von Polygonen repräsentieren. Auf der obersten Ebene findet sich ein *map agent*.

Die Agenten versuchen jeweils, bestimmte Größen- und Lagebedingungen (constraints) zu erfüllen und können diesbezüglich unterschiedliche Zufriedenheitsgrade erreichen, die sie mittels einer

partiellen („lokalen“) Bewertungsfunktion ermitteln. Der Grad der Zufriedenheit eines Agenten ist abhängig vom Grad der Übertretung der genannten Bedingungen sowie der Zufriedenheit untergeordneter Agenten. Im Allgemeinen müssen *mehrere* Bedingungen erfüllt werden, damit ein Agent zufrieden ist.

Um den eigenen Zufriedenheitsgrad zu steigern, kann ein Agent verschiedene Pläne ausführen. Welcher Plan zur Anwendung kommt, hängt dabei von der Art der nicht eingehaltenen Bedingungen sowie vom Grad der Übertretung ab. Beispielsweise überprüft ein *polygon agent* regelmäßig, ob das von ihm verwaltete Polygon auf der Karte mindestens eine Fläche von $4mm^2$ einnimmt. Ist die Fläche kleiner als $1mm^2$, kann er einen Plan ausführen, der zur vollständigen Eliminierung des Objekts auf der Karte führt. Bei einer weniger starken Übertretung der Größenbedingung (Fläche zwischen $1mm^2$ und $4mm^2$) versucht der Agent stattdessen zum Beispiel, die Fläche auf mindestens $4mm^2$ zu vergrößern [Gal03].

Bei der Ausführung eines solchen Plans werden entweder vom Agenten selbst verwaltete Entscheidungsvariablen wie die Koordinaten bestimmter Eckpunkte von Polygonen verändert oder entsprechende Prozesse bei untergeordneten Agenten angestoßen. Dazu kommen spezielle Operatoren zur Anwendung, die für eine Verbesserung der Situation des Agenten sorgen.

Übertragbarkeit auf andere KOP

Das Problem der Polygongeneralisierung eignet sich aus zwei Gründen besonders gut für die Bearbeitung durch ein MAS mit Agenten, die versuchen, durch die Suche nach *individuell* zufriedenstellenden Belegungen das *Gesamtproblem* zu lösen. Erstens ergibt sich die Güte des Gesamtvorschlags ohnehin aus der Aggregation der Gütewerte einzelner Teillösungen, d.h. durch Auswertung der Constraints und deren Übertretung durch die einzelnen Agenten. Daher entfällt die Dekomposition der Zielfunktion in partielle Gütefunktionen, die sonst notwendig wäre. Zweitens beeinflussen sich die Teillösungen kaum gegenseitig, insbesondere können unterschiedliche Kartenausschnitte – ja sogar unterschiedliche Polygone – fast immer unabhängig voneinander bearbeitet werden, da die Güte eines Ausschnitts (bzw. Polygons) kaum von der Güte anderer Ausschnitte (bzw. Polygone) abhängig ist. Grund dafür ist, dass die Constraints jeweils nur über die Konfiguration eines oder weniger Objekte definiert sind. Die nicht explizit angeführte Zielfunktion des Gesamtproblems hat daher nur geringe Epistasie, global gute Lösungen können durch die Kombination partiell günstiger Teillösungen produziert werden. Bei der Übertragung des Ansatzes auf andere KOP ergeben sich jedoch folgende Voraussetzungen:

1. Die Agenten benötigen partielle Bewertungsfunktionen für die Belegung „ihrer“ Entscheidungsvariablen, die – wie die Constraints bei der Bearbeitung von CSP – von möglichst wenigen zusätzlichen Variablen abhängig sind. Solche partiellen Bewertungsfunktionen sind jedoch im Allgemeinen gar nicht verfügbar (Black-Box).

2. Das Problem muss eine gewisse Dekomponierbarkeit aufweisen, sonst ist nicht zu erwarten, dass die global beste Lösung durch Kombination partiell optimaler Teillösungen zu erreichen ist (vgl. Abschnitte 4.4.2 und 4.6.1). Der Agent muss also wissen, welche Entscheidungsvariablen er ändern kann, ohne durch die Verbesserung der eigenen Zufriedenheit gleichzeitig die Zufriedenheit anderer Agenten maßgeblich zu verschlechtern. Ein MAS, dessen Agenten nur an partiell optimalen Belegungen interessiert sind, wird sonst schnell in

Zuständen gefangen bleiben, in denen Agenten sich gegenseitig „blockieren“ und daher keine globalen Optima finden.

3. Der Agent muss über entsprechende Operatoren verfügen, die die Konfiguration seines Objekts auch tatsächlich verbessern. Außerdem benötigt er ein Regelwerk, anhand dessen er in einer bestimmten Situation den richtigen Operator auswählen kann.

Beim Anwendungsbereich des im Zuge dieser Arbeit entwickelten Optimierungsverfahrens sind diese Voraussetzungen erfüllt: Dem Anwender sind partielle Bewertungsmethoden bekannt (1), die Probleme weisen eine gewisse Dekomponierbarkeit auf (2) und der Anwender kennt problemspezifische Operatoren zur Verbesserung von Lösungsvorschlägen sowie Kriterien für die Auswahl eines Operators in speziellen Situationen (3) (vgl. Abschnitte 3.1.2 und 3.5). Dennoch ergeben sich zwei Nachteile, wenn ausschließlich der beschriebene Ansatz verfolgt wird:

1. Eventuell ist das Anwenderwissen zu vage, als dass man sich *allein* auf dieses verlassen möchte. Die Unterstützung durch eine universelle Metaheuristik wäre daher wünschenswert.

2. Das MAS arbeitet nur an *einer* Lösung und sieht daher auch keine Rekombination von Teillösungen vor. Gerade für KOP mit Raumbezug ist eine Kombination von Teillösungen für unterschiedliche Teilräume jedoch erfolgversprechend und kann helfen, die frühzeitige Konvergenz in lokalen Optima zu vermeiden.

4.8 Hybridverfahren

Neben universellen und problemspezifischen Optimierungsverfahren können auch *Hybridverfahren* zum Einsatz kommen, in denen die Ansätze schwacher und starker Verfahren verknüpft werden. Diese Möglichkeit soll auch in der vorliegenden Arbeit genutzt werden – hier soll ein GA mit einem MAS verbunden werden.

Es bestehen unterschiedliche Möglichkeiten zur Verknüpfung von starken und schwachen Verfahren. Einerseits existieren Verfahren, die universelle und problemspezifische Heuristiken *sequentiell* verknüpfen. Problemspezifische Verfahren können zum Beispiel eingesetzt werden, um Startlösungen für universelle Heuristiken zu generieren oder um deren Ergebnisse „nachzuoptimieren“. Andererseits können an das spezielle Problem angepasste Algorithmen auch direkt in das universelle Verfahren integriert werden. GA werden oft hybridisiert, indem problemspezifische Operatoren und eine angepasste Lösungsrepräsentation verwendet werden [Dav91]. Auf diese Art der Hybridisierung Genetischer Algorithmen wird weiter unten näher eingegangen.

Die Vorteile von Hybridverfahren gegenüber universellen und rein problemspezifischen Verfahren stellen sich wie folgt dar:

1.) Sie sind universellen Verfahren überlegen:
Hybridverfahren nutzen problemspezifisches Wissen und sind daher universellen Verfahren (bei Anwendung auf Exemplare der entsprechenden Problemklasse) in der Regel bezüglich Performanz und Ergebnisgüte überlegen.

2.) Sie sind einfach zu erstellen und anzupassen:
Bestehende universelle Verfahren können gewissermaßen als Rahmen dienen, es muss kein kom-

plett neues Verfahren entwickelt werden. Stattdessen können spezielle Operatoren oder Heuristiken meist leicht in ein universelles Verfahren integriert werden, indem beispielsweise randomisierte Variationsoperatoren durch zielgerichtete, problemspezifische Operatoren ersetzt oder ergänzt werden. Soll das Verfahren auf Exemplare anderer Problemklassen angewendet werden, müssen nur die problemspezifischen Operatoren und Heuristiken angepasst bzw. ausgetauscht werden, während das universelle Verfahren als Rahmenverfahren bestehen bleiben kann.

3.) Sie bieten sich auch für Fälle mit geringer Problemkenntnis an:
Oft ist das Wissen über das Optimierungsproblem zu vage, als dass man sich *allein* auf problemspezifische Operatoren bzw. Heuristiken verlassen möchte: Es könnte die Gefahr bestehen, dass das Verfahren in lokalen Optima gefangen bleibt. In diesem Fall können die problemspezifischen Operatoren das universelle Verfahren mit seinen randomisierten Operatoren *ergänzen*, um ein günstiges Verhältnis zwischen *Exploration* (Vordringen in bisher nicht besuchte Bereiche des Lösungsraums) und *Exploitation* (Verstärkte Suche in Bereichen, in denen bereits gute Lösungen gefunden wurden) zu erreichen.

4.8.1 Hybridisierung genetischer Algorithmen

Bei einer Hybridisierung eines problemunabhängigen Verfahrens durch Anpassung der Suchoperatoren wird als Basisverfahren oft ein genetischer Algorithmus gewählt, der neben Mutations- auch Rekombinationsoperatoren vorsieht. Der ursprüngliche, kanonische GA wird dann hybridisiert, indem die randomisierten Variations- und Rekombinatoren durch problemspezifische Heuristiken ersetzt (oder ergänzt) werden. Dabei empfiehlt Davis [Dav91] statt der ursprünglichen binären Kodierung der Entscheidungsvariablen die Kodierung der integrierten Heuristik(en) bzw. Operatoren zu übernehmen, und zwar aus zwei Gründen:

> „First, it guarantees that the domain embodied in the encoding used by the current algorithm[20] will be preserved. Second it guarantees that the hybrid genetic algorithm will feel natural to the user, since the hybrid algorithm will be operating on the same kind of structures that the current algorithm is working on.“ (S. 56f)

Beispiel TSP:
Für die Bearbeitung von TSP kann beispielsweise ein GA herangezogen werden, der statt zufälligen Mutations- und Rekombinationsoperatoren problemspezifische Heuristiken wie *2-opt* und Reihenfolge-erhaltende Rekombinationsoperatoren verwendet. Es bietet sich eine Kodierung in natürlichen Zahlen an, wie sie beispielsweise in Abschnitt 2.3 dargestellt wurde.

Goldberg und Voessner [GV99] betonen, dass mittlerweile nahezu sämtliche praktische Anwendungen Evolutionärer Algorithmen mit problemspezifischen Suchverfahren hybridisiert sind. Als Beispiel sei der Ansatz von Bruns [Bru96] erwähnt, der einen Genetischen Algorithmus mit speziellen Lösungstechniken der Constraint-Programmierung hybridisiert, indem er die Lösungsrepräsentation anpasst und spezielle Operatoren einsetzt, die nach dem *select-assign-propagate*-Prinzip arbeiten.

[20] Mit *„current algorithm“* ist die problemspezifische Heuristik gemeint.

4.8.2 Memetische Algorithmen

Als *Genetic Local Search* [MAK07] oder auch *Memetic Algorithm* (memetischer Algorithmus) [Mos89] werden populationsbasierte Verfahren bezeichnet, bei denen die *Rekombination von Lösungsteilen* mit *lokale Suche* (vgl. Abschnitt 4.2.2) verknüpft wird: Nach jeder Rekombination wird die neue Lösung mittels lokaler Suche durch ein lokales Optimum ersetzt. In der Population befinden sich daher nur lokale Optima. Aus Sicht eines GA erhält man ein *Genetic Local Search*-Verfahren durch eine Hybridisierung, bei der der Mutationsoperator durch ein lokales Suchverfahren ersetzt wird, das nach jeder Rekombination angewendet wird.[21]

Die Bezeichnung *Memetischer Algorithmus* geht auf den Begriff des *Mems* zurück, den der Biologe Dawkins in seinem Buch „The Selfish Gene“ [Daw76] verwendet. Er betrachtet die kulturelle Evolution, die – ähnlich wie die genetische Evolution – auf der Weitergabe von Informationen beruht, jedoch um ein vielfaches schneller arbeitet. Die entsprechenden Informationseinheiten werden von Dawkins in Anlehnung an das griechische Wort *mimeme* (Imitation) als *Meme* bezeichnet, da sie durch Imitation weitergegeben werden.

Ebenso wie bei der natürlichen Evolution beruht der Prozess auf der Kombination, Variation und Übertragung dieser Informationseinheiten. Der grundlegende Unterschied besteht darin, dass die Meme vor ihrer Übertragung an andere durch den „Träger“ zielgerichtet weiterentwickelt, d.h. unter Einbeziehung seiner Erfahrungen und seines Wissens „verbessert“ werden.

Moscato [Mos89] schlägt daher die Bezeichnung Memetischer Algorithmus für EA-Hybride vor, die in dieser Beziehung eher Analogien zur kulturellen Evolution als zur genetischen Evolution zeigen. Verschiedene Anwendungen für solche Algorithmen finden sich z.B. in [CDG99]. Angelehnt an die Notation aus Abschnitt 4.3.3 könnte der Ablauf eines Memetischen Algorithmus wie folgt angegeben werden (vgl. [MAK07]):

1. **Initialisiere** zufällig eine Population von Lösungsvorschlägen
2. **Ersetze** jeden Lösungsvorschlag l der Population durch *LocalSearch(l)*
3. **Variiere** die Population iterativ. Wiederhole dazu (bis eine Abbruchbedingung zutrifft):
   (a) Erzeuge mittels **Rekombination** Nachkommen
   (b) **Ersetze** jeden Nachkommen n durch *LocalSearch(n)*
   (c) **Ersetze** Elemente der aktuellen Generation durch die Nachkommen

Merz [Mer00] untersucht Memetische Algorithmen für den Einsatz auf KOP und verwendet dabei ein erweitertes Schema, in dem Mutation und Rekombination als zwei eigenständige Variations-Operatoren erhalten bleiben. Die Lokale Suche wird nach jeder Variation auf den neuen Lösungsvorschlag angewendet. Er ergänzt den Ablauf zusätzlich um einen *Diversifikations-Schritt*: Sobald die Population konvergiert, wird nur das beste Mitglied unverändert beibehalten, auf alle anderen wird eine Mutation mit anschließender lokaler Suche angewendet. Er begründet diese Erweiterung mit dem Risiko der schnellen Konvergenz des Verfahrens: Aufgrund der Rechenintensität der lokalen Suche ist das Verfahren auf *kleine Populationen* begrenzt, die zu schneller Konver-

[21] Das hybridisierte Verfahren arbeitet auf einer problemspezifischen Lösungsrepräsentation. Es kommt ein entsprechend angepasster Rekombinationsoperator zum Einsatz.

genz neigen. Lokale Suche ist besonders rechenintensiv, wenn die Auswertung der Zielfunktion aufwändig ist wie in den Anwendungsbeispielen dieser Arbeit.

Als lokales Suchverfahren werden für praktische Anwendungen meist problemspezifische Algorithmen bzw. Definitionen der Nachbarschaft verwendet, um möglichst jegliches vorhandene Wissen über das Problem auszunutzen:

> „A key feature, presented in most MAs [=Memetic Algorithms, A.d.A.] implementations, is the use of a population-based search which intends to use all available knowledge about the problem“ [Mos99, S.1].

Dennoch können EA natürlich auch mit anderen *problemunabhängigen* Suchverfahren hybridisiert werden: Jakob [Jak04] versucht, die Konvergenzgeschwindigkeit eines EA unter Beibehaltung der Konvergenzsicherheit sowie der allgemeinen Anwendbarkeit des Verfahrens zu erhöhen, indem er mit dem *Rosenbrock*-Algorithmus und dem *Complex*-Verfahren ableitungsfreie, universelle Suchalgorithmen integriert. Er testet sein Hybridverfahren an fünf mathematischen Benchmarkfunktionen und drei praktischen Problemen und erreicht dabei sein Ziel, die Geschwindigkeit des EA durch Hybridisierung zu steigern. Nach den No-Free-Lunch-Theoremen sind jedoch im Mittel über alle Optimierungsprobleme keine besseren Ergebnisse oder Laufzeiteinsparungen zu erwarten, wenn nur mit problemunabhängigen Verfahren hybridisiert wird.

Für die Bearbeitung von KOP aus dem Anwendungsbereich des zu entwickelnden Verfahrens bieten sich Memetische Algorithmen nur dann an, wenn die Anzahl der jeweils für eine lokale Suche auszuwertenden Lösungsvorschläge begrenzt werden kann, da die Evaluation von Lösungsvorschlägen unter Umständen recht aufwändig werden kann. Anstatt *alle* Vorschläge aus der Nachbarschaft zu evaluieren, sollte daher versucht werden, aufgrund der *partiellen Bewertbarkeit* von Lösungsvorschlägen schnell *Lösungsteile mit Verbesserungspotential* zu finden und so *zielgerichtete* Variationen durchzuführen. Auf diese Weise arbeiten die in Abschnitt 4.7.3 beschriebenen Greedy-MAS. Beim im Zuge dieser Arbeit entwickelten Optimierungsverfahren handelt es sich daher um eine Hybridisierung eines GA mit einem Greedy-MAS.

Kapitel 5

Die Genetische Konferenz: Konzeption des entwickelten Verfahrens

Ziel der Arbeit ist die Entwicklung eines Optimierungsverfahrens für kombinatorische Optimierungsprobleme (KOP) – insbesondere Probleme mit Raumbezug – unter Ausnutzung vorhandenen Problemwissens. KOP lassen sich im Allgemeinen nicht effizient *exakt* lösen, deshalb müssen in der Praxis *Heuristiken* angewendet werden, die gute Lösungen in annehmbarer Zeit finden. Man unterscheidet hier universelle (schwache) Verfahren wie GA und problemspezifische (starke) Verfahren wie Greedy-MAS.

Außerdem besteht die Möglichkeit, verschiedene Verfahren miteinander zu verknüpfen (Hybridverfahren). Es gilt dabei: Je mehr Anwenderwissen über zu bearbeitende Problemexemplare in eine ursprünglich universelle Heuristik integriert werden kann, desto höhere Performanz und Ergebnisgüte ist zu erwarten. Für den Anwendungsbereich „KOP mit beschränktem Anwenderwissen“ soll ein Verfahren entwickelt werden, das aus einer Verknüpfung eines GA mit einem Greedy-MAS besteht. Dieses Kapitel beschreibt die Entwicklung dieses Verfahrens.

Zunächst werden die Eigenschaften von GA und MAS bei der Bearbeitung von KOP noch einmal zusammengefasst (5.1). Im Anschluss wird das Grobkonzept zur Verknüpfung der beiden Verfahren vorgestellt und die Namensgebung („Genetische Konferenz“) erläutert (5.2). Es folgt eine formale Definition des entwickelten Verfahrens (5.3). Im Anschluss werden die dem Verfahren zugrunde liegenden Konzepte im Detail erläutert (5.4). Zuletzt folgt eine Einordnung des entwickelten Verfahrens im Kontext von MAS und GA (5.5).

5.1 Zusammenfassung der Ausgangslage

Mit dem hier entwickelten Verfahren für KOP mit Raumbezug wird versucht, die Vorteile der beiden Ansätze Greedy-MAS und (Hybrid-)GA zu vereinigen. In diesem Abschnitt soll zunächst noch einmal zusammenfassend betrachtet werden, inwiefern sich die Verfahren *einzeln* zur Lösung der Probleme aus dem Anwendungsbereich des entwickelten Verfahrens eignen.

Ein kanonischer GA als problemunabhängige (schwache) Heuristik geht von mehreren aktuellen Lösungen aus und generiert neue Lösungen, indem er auf zufällige Weise (1) gute bisherige Lösungen *lokal variiert* und (2) Teile guter bisheriger Lösungen miteinander *kombiniert*. Dabei setzt man implizit voraus, dass einerseits (1) eine gewisse Korrelation besteht zwischen der Nähe zweier Lösungen im Lösungsraum und ihrer Güte und andererseits (2) die Entscheidungsvariablen keine allzu große Epistasie aufweisen. Ohne diese Eigenschaften der Zielfunktion vorauszusetzen, kann nicht davon ausgegangen werden, dass durch lokale Veränderung und Kombination der Belegungen schneller bessere Lösungen erreicht werden als durch bloße Zufallssuche.

Bei praktischen KOP mit Raumbezug treffen Eignungskriterien für GA wie *hohe Korrelation zwischen Nähe und Gütedifferenz von Lösungen* sowie *geringe Epistasie unter den Entscheidungsvariablen* meist zu, und GA schneiden daher in der Regel besser ab als bloße Zufallssuche. Als populationsbasierte Verfahren sind GA auf Problemen mit geringer Epistasie (wenigstens in der Theorie) dabei anderen Verbesserungsverfahren überlegen, die keine Kombination von Teillösungen vorsehen. Trotzdem arbeiten GA mit problemunabhängigen, zufallsbasierten Variationsoperatoren gerade auf „großen“ KOP mit vielen Entscheidungsvariablen und komplexen (und damit bei der Auswertung zeitintensiven) Evaluationsfunktionen oft langsam. Deshalb werden in der Praxis nahezu immer *mit problemspezifischen Operatoren bzw. Suchverfahren „hybridisierte“* genetische Algorithmen für die Bearbeitung von KOP herangezogen. Die Integration problemspezifischer Operatoren ist auf einfache Weise möglich.

Die Substitution bzw. Ergänzung der Suchoperatoren eines GA durch problemspezifische Operatoren soll im Folgenden als *klassische* Hybridisierung bezeichnet werden. Auf klassische Weise hybridisierte GA nutzen zwar bereits problemspezifisches Wissen, vernachlässigen jedoch das möglicherweise ebenfalls vorhandene Wissen des Anwenders über partielle Bewertungsmethoden und daraus resultierendes lokales Verbesserungspotential sowie Kenntnisse von Auswahlkriterien für anzuwendende Variationsoperatoren. Der GA *kennt* also vielversprechende Suchoperatoren für das Problem, weiß jedoch für einen gegebenen Lösungsvorschlag nicht, *welchen dieser Operatoren* er auf *welche Entscheidungsvariablen* anwenden soll, um den Vorschlag mit möglichst großer Wahrscheinlichkeit tatsächlich zu verbessern.

Anders gelagert sind Vor- und Nachteile eines MAS, das zur Bearbeitung des Optimierungsproblems die Entscheidungsvariablen von Agenten „verwalten“ lässt („*Greedy-MAS*“). Im einfachsten Fall arbeiten die beteiligten Agenten an nur einem Lösungsvorschlag, den sie parallel oder sequentiell an den „für sie relevanten“ Stellen verbessern, indem sie jeweils die Belegung(en) der von ihnen verwaltete(n) Entscheidungsvariable(n) ändern.

Ein solches Greedy-MAS stellt beispielsweise das System zur Generalisierung thematischer Karten dar, das Galanda [Gal03] beschreibt (vgl. Abschnitt 4.7.3). Das zugrunde liegende Konzept ist auch auf andere Probleme anwendbar unter der Voraussetzung, dass das Problem die partielle Evaluierung von Lösungsteilen zulässt – der Anwendungsbereich des hier zu entwickelnden Verfahrens umfasst gerade solche Probleme.

Die partiellen Bewertungsfunktionen dienen im Greedy-MAS zur Messung der Zufriedenheit der Agenten. Die Zufriedenheitswerte der Agenten werden benötigt, um zu entscheiden, ob der Vorschlag an der entsprechenden Stelle verändert werden soll. Dabei ist möglich, den Grad der Zufriedenheit von mehreren Kriterien abhängig zu machen. Anhand der Erfüllung der einzelnen Kriterien kann dann später entschieden werden, *mit welchem Operator* der Vorschlag variiert wird.

Hier wird also problemspezifisches *a-priori*-Wissen über die Erkennung von Verbesserungspotential und die in unterschiedlichen Situationen anzuwendenden Operatoren ausgenutzt, der Ansatz ist insofern als starke Methode (vgl. Abschnitt 4.1.1) zu bezeichnen. In der Analyse des Verbesserungspotentials der Belegungen einzelner Variablen und der damit einhergehenden *Auswahl der Variationsstelle und des Variationsoperators* liegt die Stärke des MAS-Ansatzes im Vergleich zu einem auf klassische Weise hybridisierten GA.

Hauptnachteil ist jedoch, dass nur an *einem* Lösungsvorschlag gearbeitet wird (Einpunkt-Verfahren). Sobald durch die Aktionen einzelner Agenten, d.h. die lokale Veränderung *einzelner* Entscheidungsvariablen keine Verbesserung des Vorschlags mehr erreicht werden kann, konvergiert das Verfahren. Deshalb werden zwar schnell *lokale* Optima gefunden, der Algorithmus kann aber leicht in diesen „gefangen“ bleiben und findet dann nicht das *globale* Optimum.

Um lokale Optima verlassen zu können, müssten entweder temporäre Verschlechterungen zugelassen werden, oder die „gleichzeitige“ Veränderung der Belegungen *mehrerer* Entscheidungsvariablen müsste ermöglicht werden – etwa wie bei GA durch Kombination von Lösungsvorschlägen. Die Verwendung „allmächtiger“ Agenten, die über globales Problemwissen verfügen und möglicherweise alle Entscheidungsvariablen gleichzeitig verändern können, widerspricht dagegen dem Ansatz der Dekomposition des Problems und seiner verteilten Lösung.

Weiterer Nachteil eines Greedy-MAS ist, dass es bei der Anwendung auf ein anderes Optimierungsproblem jeweils entsprechend angepasst werden muss – gleiches gilt natürlich auch für Hybrid-GA. Für ein vielseitig anwendbares Optimierungsverfahren sollten daher ein Rahmenwerk (*Framework*) mit entsprechenden Möglichkeiten der einfachen Adaption vorgesehen werden.

Tabelle 5.1 zeigt Eigenschaften von kanonischem GA, klassischem Hybrid-GA (mit problemspezifischen Variationsoperatoren) und Greedy-MAS im Überblick.

	kanonischer GA	**„Klassischer“ Hybrid-GA**	**Greedy-MAS**
Integration problemspezifischer Operatoren	nein	ja	ja
Verwendung partieller Bewertungsfunktionen	nein	nein	ja
Universelle Einsetzbarkeit des Verfahrens	ja	nein	nein
Anfälligkeit für lokale Optima	gering	gering	hoch

Tabelle 5.1: *Eigenschaften von kanonischem GA, Hybrid-GA und Greedy-MAS*

5.2 Grobkonzeption des Verfahrens

5.2.1 Die Grundidee

Das Grundgerüst des entwickelten Verfahrens bildet ein *Lösungsvorschlags-Pool*, der dem Chromosomenpool eines Standard-GA entspricht und ein *Multiagentensystem*, das – ähnlich wie beim in Abschnitt 4.7.3 beschriebenen Greedy-MAS – die Variation und Bewertung von Lösungsvorschlägen übernimmt.

Die Objekte, die dem zu bearbeitenden Problem zugrunde liegen, werden individuell durch Agenten verwaltet, die über Problemwissen verfügen und im Sinne einer verteilten Problemlösung nach einem globalen Optimum, d.h. einem Kompromiss für die Konfigurationen ihrer Objekte suchen.

Dazu entscheiden sie anhand partieller Bewertungsfunktionen, *welche* Lösungsvorschläge *an welchen Stellen* und *mit welchen Operatoren* modifiziert bzw. *welche* Vorschläge *auf welche Weise* zu neuen Vorschlägen kombiniert werden sollen.

Die wesentliche Erweiterung zu „klassischen“ Hybrid-GA besteht dabei in der *partiellen Bewertung von Lösungsvorschlägen* und der dadurch ermöglichten *Situations-spezifischen Auswahl eines Verbesserungsoperators* sowie der *Variationsstelle*, d.h. der Entscheidungsvariablen, auf die der Operator angewendet wird.

Abbildung 5.1 zeigt die wesentlichen Beziehungen zwischen dem Vorschlagspool und dem Multiagentensystem.

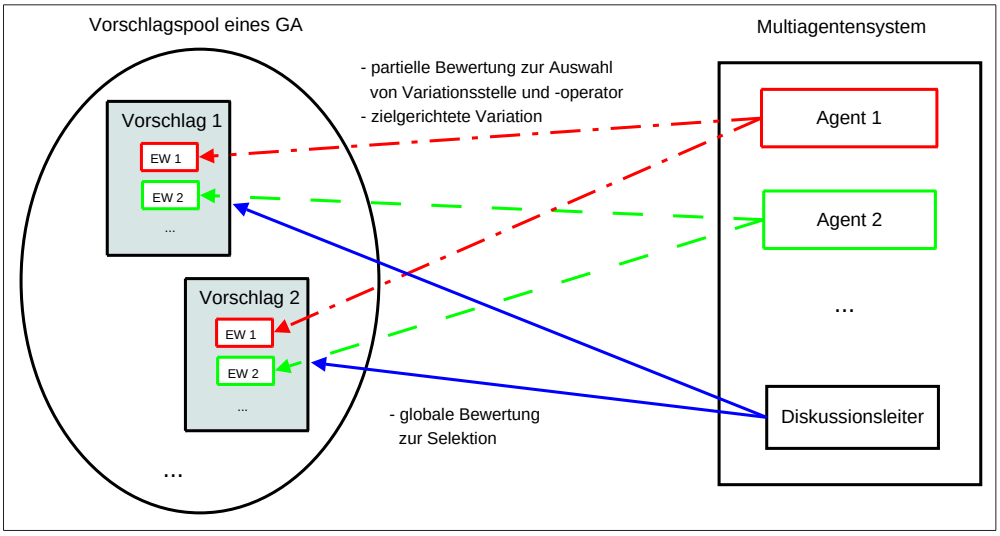


Abbildung 5.1: *Die Kombination von MAS und GA*

Der Ablauf des Verfahren besteht aus den vier Schritten *Initialisierung*, *Bewertung*, *Selektion* und *Variation*, die Analogien zu denen eines klassischen GA aufweisen, jedoch nun von den Agenten unter Einbeziehung ihres Problemwissens durchgeführt werden.

5.2.2 Der Ablauf des Verfahrens

Schritt 1: Initialisierung
Anfangs initialisieren die Agenten den Vorschlagspool (vgl. Abbildung 5.2). Dazu produzieren sie zufällig oder besser unter Einbeziehung problemspezifischen Wissens Lösungsvorschläge – beispielsweise durch Anwendung einer speziellen Konstruktionsheuristik, die im Vorfeld vom Anwender spezifiziert wurde.

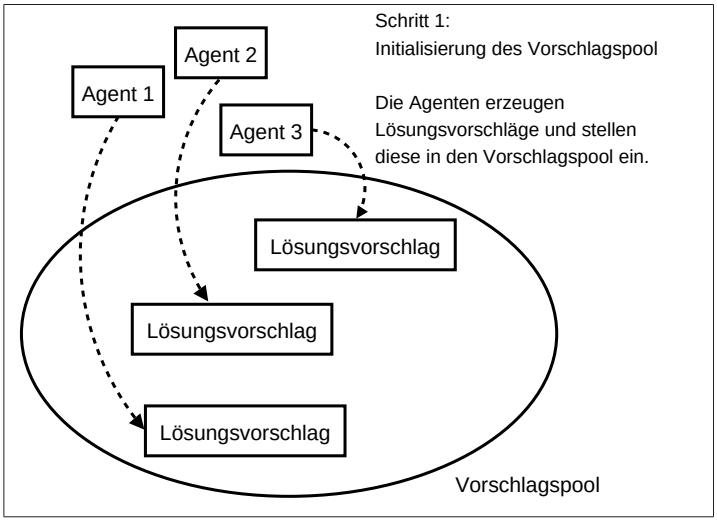


Abbildung 5.2: *Ablauf des kombinierten Verfahrens (Schritt 1: Initialisierung)*

Schritt 2: Bewertung
Im Anschluss bewertet jeder Agent jeden Vorschlag im Pool *partiell*, d.h. er betrachtet jeweils nur die Güte der Konfiguration des von ihm verwalteten Objekts. Dazu sollte es möglich sein, die *globale* Bewertungsfunktion des zu bearbeitenden Problems in *partielle Bewertungsfunktionen* aufzuteilen, die von den Agenten „vor Ort“ durchgeführt werden können. Die globale Bewertung als Maß für die Güte des Lösungsvorschlags *insgesamt* lässt sich dann wiederum berechnen durch Aggregation der partiellen Bewertungen aller Agenten sowie möglicherweise einer „Restfunktion“, die ein zentraler „Diskussionsleiter“ vornimmt (vgl. Abbildung 5.3). Die Möglichkeit der Dekomposition der Zielfunktion ist zwar nicht *Voraussetzung* für die Anwendbarkeit des Verfahrens, da die Restfunktion theoretisch auch die gesamte Bewertung übernehmen kann. Allerdings können die partiellen Bewertungen die Grundlage bilden für die folgenden Berechnungen der Agenten, die auf die Auswahl der Variationsstelle und den gezielten Einsatz problemspezifischer Heuristiken abzielen – und somit zur Erfüllung wichtiger Anforderungen an das Optimierungsverfahren (vgl. Abschnitt 3.5) beitragen: Die Agenten berechnen – in der Regel basierend auf der partiellen Bewertung – zusätzlich für jeden Vorschlag *lokale Güte-Koeffizienten*, die die Qualität der momentanen Konfiguration des durch den Agenten verwalteten Objekts messen, und zwar möglicherweise hinsichtlich mehrerer unterschiedlicher Kriterien. Die Koeffizienten sollen einerseits später Anhaltspunkte liefern, durch welche Operationen die derzeitige Konfiguration des verwalteten Objekts verbessert werden kann. Andererseits können die Agenten durch Auswertung einer *Zufriedenheitsfunktion* das lokale Verbesserungspotential jedes Vorschlags abschätzen. Die

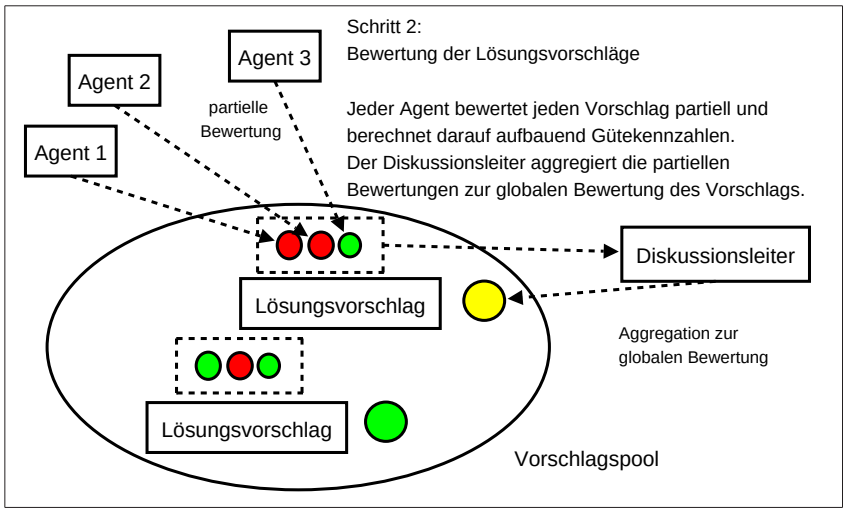


Abbildung 5.3: *Ablauf des kombinierten Verfahrens (Schritt 2: Bewertung)*

Zufriedenheitsfunktion beruht auf der Aggregation der Güte-Koeffizienten und dient zur späteren Identifikation von Teilen des Lösungsvorschlags (d.h. Konfigurationen von Objekten), die möglicherweise noch verbessert werden können („Verbesserungsstelle“).

Schritt 3: Selektion

Wenn alle Vorschläge von allen Agenten bewertet sind, wird wie beim klassischen GA ein gewisser Anteil (insgesamt) schlechter bewerteter Vorschläge aus dem Pool entfernt, um Platz für neue Vorschläge zu schaffen – diese Aufgabe übernimmt wiederum der Diskussionsleiter (vgl. Abbildung 5.4). Im Unterschied zum klassischen GA „merken“ sich die einzelnen Agenten jedoch Vorschläge in individuellen „Gedächtnissen“, die zwar global nicht (mehr) ausreichend gut bewertet wurden, um sie im Pool zu halten, lokal jedoch besonders hohe Zufriedenheitswerte hervorgerufen hatten. Teilinformationen solcher Vorschläge können später mittels Rekombination mit anderen Vorschlägen wiederverwendet werden.

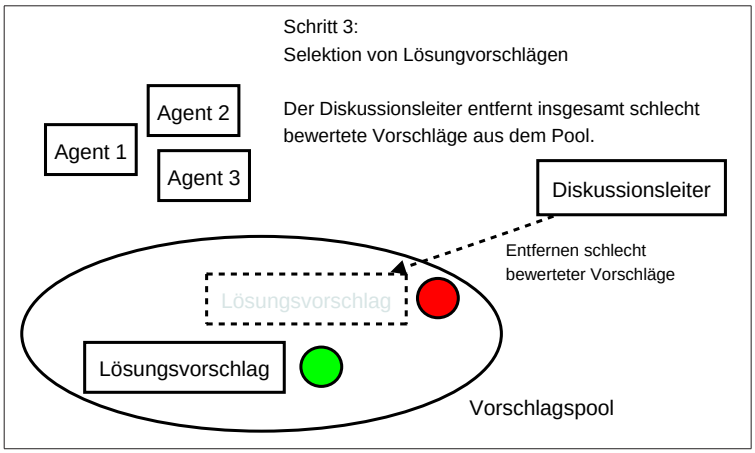


Abbildung 5.4: *Ablauf des kombinierten Verfahrens (Schritt 3: Selektion)*

Schritt 4: Variation

Im nächsten Schritt erzeugen die Agenten neue Vorschläge, indem sie unabhängig voneinander variierte Kopien bisheriger Vorschläge in den Pool einbringen (vgl. Abbildung 5.5). Zunächst wird dazu jeweils wie beim klassischen GA ein Lösungsvorschlag aus dem Pool ausgewählt, der variiert werden soll. Der Vorschlag wird einem Agenten zur Variation übergeben. Der Agent entscheidet dann je nach individueller Zufriedenheit mit dem Vorschlag, ob er ihn selbst durch Änderungen an den Belegungen der von ihm verwalteten Entscheidungsvariablen modifizieren will, oder an einen weiteren Agenten delegieren soll. Die Variation soll möglichst von einem Agenten durchgeführt werden, bei dem die Auswertung der Zufriedenheitsfunktion auf hohes Verbesserungspotential schließen lässt (siehe Schritt 1). Im Unterschied zum klassischen GA wird also nicht nur der zu variierende Lösungsvorschlag anhand der *globalen Güte* ausgewählt, sondern zusätzlich anhand der *lokalen Zufriedenheitsmaße* ermittelt, „*an welcher Stelle*“, d.h. durch Variation welcher Entscheidungsvariablen der Vorschlag verändert werden soll. Diese Auswahl der *Variationsstelle* basiert auf den partiellen Gütekriterien und damit auf dem Problemwissen des Anwenders.

Bei der konkreten Variation geht dann – soweit möglich – weiteres Problemwissen in die Handlungen der Agenten ein. Der mit der Variation beauftragte Agent kann einerseits versuchen, den Vorschlag durch Ausführung problemspezifischer Operationen lokal an der ausgewählten Variationsstelle zu verbessern. Er versucht dabei zu erreichen, dass der neue Vorschlag eine höhere partielle – und dadurch möglicherweise auch globale - Bewertung erhält (zielgerichtete, punktuelle Verbesserung). Alternativ kann der Agent den Vorschlag mit einem weiteren Vorschlag kombinieren. Die Auswahl des zweiten Ausgangsvorschlags basiert wiederum auf der jeweiligen Zufriedenheit des beauftragten Agenten. Hier kann der Agent auch auf Vorschläge aus seinem Gedächtnis zurückgreifen.

Sobald der Vorschlagspool wieder angefüllt ist, unterbricht ein „Diskussionsleiter“ den Variationsschritt, überprüft die Abbruchbedingungen und stoppt das Verfahren oder veranlasst die Agenten, wiederum zur Bewertung der Vorschläge (Schritt 2) überzugehen.

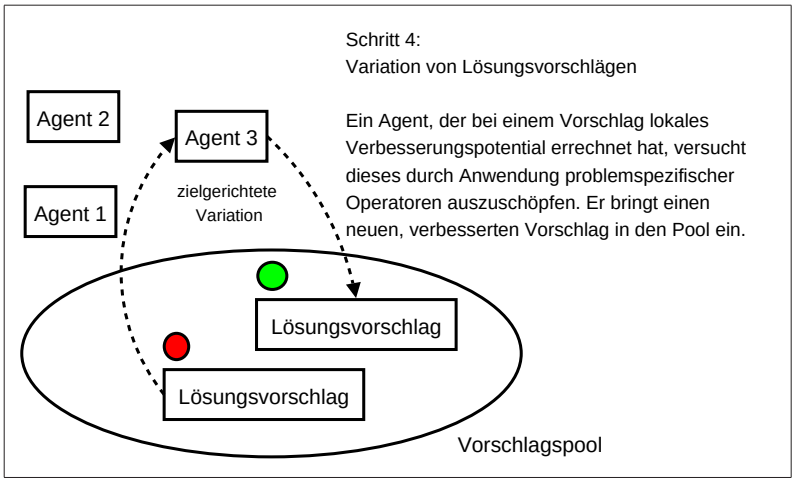


Abbildung 5.5: *Ablauf des kombinierten Verfahrens (Schritt 4: Variation)*

5.2.3 Die Bezeichnung „Genetische Konferenz“

Den Ablauf des kombinierten Verfahrens könnte man mit einer *Konferenz* von Vertretern („Agenten“) unterschiedlicher Interessensgruppen vergleichen, die zum Ziel haben, zu einem bestimmten Thema einen für alle Agenten möglichst zufriedenstellenden Beschluss zu fassen:

Die Agenten bringen zunächst Vorschläge ein, bei denen jeweils die eigenen Interessen im Vordergrund stehen (*Initialisierung*).

Im Anschluss wandeln die Agenten jeweils bereits eingebrachte Vorschläge anderer Agenten nach eigenen Interessen ab oder kombinieren mehrere Vorschläge miteinander (*Variation*). Dabei verändert ein Agent insbesondere dann einen Vorschlag, wenn er (noch) nicht mit ihm zufrieden war, weil seine eigenen Interessen nur ungenügend berücksichtigt waren. Zwischenzeitlich wird immer wieder über Vorschläge abgestimmt (*Bewertung*).

Die Güte eines Vorschlags ergibt sich aus der Aggregation individueller Bewertungen (beispielsweise Auszählung von Handzeichen oder Summierung vergebener Punkte). Vorschläge, die insgesamt nur auf geringe Zustimmung stoßen, werden ausgesondert (*Selektion*).

Aufgrund dieser Analogien – und in Anlehnung an das zugrunde liegende Konzept der Genetischen Algorithmen – wird das entwickelte Optimierungsverfahren als „*Genetische Konferenz*“ bezeichnet.

5.2.4 Gentechnik als Metapher

Versucht man, passende Metaphern für das beschriebene Vorgehen aus dem Bereich der Biologie und der Genetik zu finden, so kann man die Ansätze bei der Bewertung und insbesondere der Variation von Vorschlägen am ehesten mit Methoden der *Gentechnik* vergleichen, d.h. der *zielgerichteten* Veränderung bestimmter Gene aufgrund der *Kenntnisse ihrer individuellen Wirkungsweise*.

Ein (kanonischer) GA setzt bei der Rekombination und Mutation vorrangig auf den Zufall, die „Heranzüchtung“ guter Individuen erfolgt allein durch die *Selektion*. Bei der Genetischen Konferenz wird dagegen in den Lösungsvorschlägen („Chromosomen“) zielgerichtet nach Stellen („Genen“) mit Verbesserungspotential gesucht, die dann ausgetauscht werden.

Die Agenten des Optimierungsverfahrens könnten als „Gentechniker“ angesehen werden, deren Ziel es ist, möglichst *schnell* eine qualitativ hochwertige Züchtung zu erreichen. Ein Gentechniker ist dabei jeweils für eine kleine Zahl von Genorten zuständig und kann die Wirkungsweise am Genort auftretender Allele einschätzen. Aus einer Population von Individuen sucht er sich nun immer wieder diejenigen heraus, bei denen er die auf den verwalteten Genorten auftretenden Gene als weniger günstig einstuft und sorgt für deren Modifikation. Die anderen Gentechniker gehen parallel ebenso vor.

Auch bei der Kreuzung von Individuen nutzen sie ihre Kenntnisse über die Auswirkungen bestimmter Gene und versuchen, die günstigen Eigenschaften von Individuen möglichst zu verbinden. Voraussetzung ist dabei einerseits natürlich die Kenntnis der Bedeutung und Auswirkung bestimmter Allele sowie andererseits möglichst geringe Epistasie (Wechselwirkungen zwischen den Genen).

5.2.5 Abgrenzung von anderen Ansätzen

Blackboard-Systeme und die verteilte Bearbeitung von Problemen

Die *Konferenz*-Metapher mag zunächst an die altbekannten *Blackboard*-Systeme (vgl. Abschnitt 4.7) erinnern. Das *Blackboard*-Architekturmodell ist sehr weit gefasst und beschreibt verteilte Systeme, deren Einzelkomponenten (*knowledge sources*) zur gemeinsamen Problemlösung nicht direkt, sondern über ein gemeinsames Medium (*blackboard*) kommunizieren. Sowohl mit *Blackboard*-Systemen als auch mit der *Genetischen Konferenz* sollen Probleme auf verteilte Weise bearbeitet werden. Sonst lassen sich jedoch nur wenige Gemeinsamkeiten finden. Die Genetische Konferenz ist nicht als Architekturmodell für die Form der Interaktion eines verteilten Systems, sondern als neuer, integrierter Lösungsansatz für die Bearbeitung von KOP anzusehen.

Während das *blackboard*-Modell weder die Art des zu bearbeitenden Problems, noch der Beiträge der Agenten genauer festlegt, ist der Anwendungsbereich der Genetischen Konferenz viel spezifischer und die möglichen Aktionen der Agenten sind genau festgelegt: Die Agenten arbeiten hier auf einem Pool von Lösungsvorschlägen für ein KOP und führen dazu Aktionen aus, die mit den wesentlichen Schritten eines Genetischen Algorithmus korrespondieren. Außerdem findet bei der Delegation von Lösungsvorschlägen und der Bildung von Koalitionen (die später eingeführt wird) auch eine direkte Kommunikation unter den Agenten statt.

Die Beiträge der Agenten in *Blackboard-Systemen* dienen vor allem der Unterstützung anderer Agenten bei der gemeinsamen Lösungsfindung („*If you can draw it, I can use it*“ [Cor91, S.2]). Die Agenten sind hier also als äußerst kooperativ zu bezeichnen. Das stellt sich bei der Genetischen Konferenz ebenfalls anders dar: Hier versucht ein Agent, insbesondere die Konfiguration des von ihm verwalteten Objekts in einer Lösung (lokal) zu verbessern. Solche Variationen haben oft eine Minderung der Güte anderer Konfigurationen – und damit auch der Zufriedenheit anderer Agenten zur Folge. Die Agenten konkurrieren in diesem Fall um bestmögliche Repräsentation in den Lösungen. Besonders anschaulich wird dies, wenn das Anwendungsbeispiel der Kurszuteilung für die Klinische Rotation betrachtet wird: Dort konkurrieren die Agenten um Kursplätze für die von ihnen repräsentierten Studierenden (vgl. Abschnitt 7.4).

Memetische Algorithmen und die Einbeziehung problemspezifischen Wissens in GA

Die Beschleunigung eines Genetischen Algorithmus durch die Integration problemspezifischen Wissens sehen auch andere Ansätze vor. Bei den in Abschnitt 4.8.2 näher beschriebenen Memetischen Algorithmen werden beispielsweise bekannte problemspezifische lokale Suchheuristiken in einen GA integriert. Mit diesen Heuristiken wird jedes neue Individuum zunächst in ein lokales Optimum umgewandelt, bevor es in die Population eingestellt wird.

Die Genetische Konferenz sieht zwar ebenfalls die Nutzung problemspezifischer Operatoren vor, allerdings müssen diese nicht unbedingt lokale Optima erzeugen. Da der Anwendungsbereich des Verfahrens insbesondere auch Probleme umfasst, bei denen eine Auswertung von Lösungsvorschlägen aufwändig ist, soll die Nachbarschaft nicht komplett nach einer besseren Lösung abgesucht werden. Ziel ist stattdessen, die Anwendungsstelle eines Variationsoperators im Lösungsvorschlag durch Abschätzung lokalen Verbesserungspotentials ebenso zielgerichtet auszuwählen, wie den Operator selbst. Im Unterschied zu memetischen Algorithmen werden dazu *partielle Bewertungen* durchgeführt, die von den Agenten eines MAS übernommen werden. Die Genetische Konferenz nutzt dabei *zusätzliches* Anwenderwissen, das sich auf die partielle Bewertung von Lösungsteilen und die Situations-spezifische Auswahl von Operatoren bezieht.

5.3 Formale Definition der Genetischen Konferenz

Nach der Darstellung des Grundkonzepts der Genetischen Konferenz soll in diesem Abschnitt zunächst eine formale Definition der Komponenten und Operationen sowie eine formale Beschreibung des Ablaufs der Genetischen Konferenz geliefert werden. Im nächsten Abschnitt werden die eingeführten Konzepte dann im Detail besprochen.

5.3.1 Die Genetische Konferenz (*GENETIC-CONFERENCE*)

Die Genetische Konferenz (*GENETIC-CONFERENCE*) kann – in einer vereinfachten Form[1] – formal definiert werden als Quadrupel aus Agentensystem (*AGENTSYSTEM*, vgl. 5.3.5), Vorschlagspool (*POOL*, vgl. 5.3.2), Diskussionsleiter (*MODERATOR*, vgl. 5.3.7) und Statusvariablen (*STATUS*, vgl. 5.3.4)):[2]

$$GENETIC\text{-}CONFERENCE = (AGENTSYSTEM,\ POOL,\ MODERATOR,\ STATUS)$$

Ziel ist es, eine günstigste Kombination der Konfiguration von Objekten zu finden (vgl. 2.5). Die Konfiguration eines Objekts wird durch die *Ausprägung* mehrerer *Merkmale* bestimmt. Die Merkmale können in Merkmalssätzen zusammengefasst werden, so dass jedem Objekt für jeden Merkmalssatz genau ein Wert zugewiesen werden muss (nähere Erläuterungen finden sich im Abschnitt 5.4.1). Jedes Objekt wird von einem individuellen Agenten des Agentensystems verwaltet.

5.3.2 Der Vorschlagspool (*POOL*)

Der Vorschlagspool *POOL* entspricht einer Zuordnungstabelle und besteht aus einer Menge von Schlüssel-Wert-Paaren. Das erste Element jedes Paars ist ein Vorschlag, das zweite der entsprechende Zielfunktionswert:

$$POOL = \{(PROPOSAL, \mathbb{R})\}_{0..\phi}$$

Wurde der Vorschlag noch nicht bewertet, so wird er zusammen mit dem Wert $-\infty$ abgelegt.[3] ϕ bezeichnet die maximale Anzahl von Vorschlägen, die sich im Pool befinden kann. Mit den Indizes $M_{0..\phi}$ wird ausgedrückt, dass die Menge M *bis zu* ϕ Elemente beinhalten kann.

5.3.3 Der Vorschlag (*PROPOSAL*)

In einem Vorschlag (*PROPOSAL*) wird *jedem* Agenten (*AGENT*, vgl. 5.3.6) für *jeden* Merkmalssatz (*ATTRIBUTE*) eine Ausprägung (*VALUE*) zugeordnet:

$$PROPOSAL = VALUE^{|\beta|\times|\alpha|}$$

[1] Der besseren Lesbarkeit halber werden einige der in den folgenden Kapiteln näher erläuterten Konzepte wie *Koalitionen* und *Lernen* in dieser formalen Definition noch nicht berücksichtigt.

[2] Bei der formalen Definition kennzeichnen griechische Buchstaben ganzzahlige Parameter, neu definierte Typen werden in GROSSCHRIFT angegeben, mit kleinen Buchstaben werden jeweils einzelne Exemplare solcher Typen bezeichnet. Operationen werden in gemischter Gross- und Kleinschrift notiert.

[3] Im Folgenden wird von der Bearbeitung von Maximierungsproblemen ausgegangen.

Dabei bezeichnet α die Anzahl der beteiligten Agenten und β die Anzahl der Merkmalssätze. Jeder Merkmalssatz umfasst β Merkmale, d.h. für jeden der α Agenten genau eines. Die Anzahl der Entscheidungsvariablen des Optimierungsproblems beträgt somit $\beta \times \alpha$.

5.3.4 Die Statusvariablen (*STATUS*)

Die Genetische Konferenz verfügt über ein Feld von σ Statusvariablen, in die Informationen über den Stand der Optimierung eintragen werden können, und auf deren Grundlage über ein mögliches Ende des Optimierungslaufs entschieden wird.

$$STATUS = \mathbb{R}^{\sigma}$$

5.3.5 Das Agentensystem (*AGENTSYSTEM*)

Das Agentensystem *AGENTSYSTEM* ist ein Quadrupel aus der Menge der an der Konferenz beteiligten Agenten (*AGENT*), den zu verwaltenden Merkmalen (*ATTRIBUTE*), möglichen Ausprägungen (*VALUE*) sowie Plänen (*PLAN*), die zur Verbesserung eines Vorschlags ausgeführt werden können.

$$AGENTSYSTEM = (\{AGENT\}_{\alpha}, \{ATTRIBUTE\}_{\beta}, \{VALUE\}_{\xi}, \{PLAN\}_{\delta})$$

Die Anzahl der möglichen Ausprägungen der Merkmale eines Merkmalssatzes beträgt ξ, die Anzahl der Pläne, die zur Verbesserung der Vorschläge herangezogen werden können ist δ.

Die Anzahlen der beteiligten Agenten und der Merkmalssätze wurde oben bereits mit α respektive β festgelegt. Als Datentyp der Merkmalssätze und Pläne können Zeichenketten verwendet werden, der Datentyp der Ausprägungen ist beliebig.

5.3.6 Der Agent (*AGENT*)

In Anlehnung an die spätere objektorientierte Modellierung wurde zugunsten der besseren Lesbarkeit darauf verzichtet, die Operationen und Attribute der Agenten in die Definition des Agentensystems aufzunehmen. Stattdessen werden diese Operationen direkt hier beim Agenten definiert. Ein Agent (*AGENT*) ist ein 11-Tupel:

$$AGENT = (\textit{initProp, evalProp, calcRatios, calcSat, requireImp, selSatProp, selPlan, alterProp, combineProps, MEMORY, rememberProp})$$

Um zu kennzeichnen, für welchen Agenten eine Operation ausgeführt wird bzw. das Gedächtnis welches Agenten gemeint ist, werden Indizes verwendet. $initProp_a$ bezeichnet demnach die Initialisierungsmethode für Vorschläge des Agenten a.

initProp ist eine Operation, mit der ein Agent einen neuen Vorschlag erstellt:

$$initProp_{AGENT} : \varnothing \rightarrow PROPOSAL$$

Mit der Operation *evalProp* kann der der Agent einen Vorschlag partiell evaluieren. Der Vorschlag wird auf ϵ *partielle Evaluationswerte* abgebildet:

$evalProp_{AGENT} : PROPOSAL \rightarrow \mathbb{R}^{\epsilon}$

Mit der Funktion *calcRatios* berechnet der Agent γ partielle Güte-Koeffizienten, die zur Ermittlung des lokalen Verbesserungspotentials und zur Auswahl möglicher Verbesserungsoperatoren dienen. Die Koeffizienten sind zwischen 0 (ungüstig) und 1 (günstig) skaliert. Die Berechnung der Koeffizienten kann auf Basis der partiellen Evaluationswerte oder auch direkt auf Grundlage der Belegungen des Vorschlags durchgeführt werden: $calcRatios_{AGENT} : \mathbb{R}^{\epsilon} \times PROPOSAL \rightarrow [0..1]^{\gamma}$

Mit der lokalen Zufriedenheitsfunktion *calcSat* kann der Agent einen aus den Koeffizienten aggregierten Zufriedenheitswert zwischen 0 (nicht zufrieden) und 1 (zufrieden) berechnen. Dieser Wert gibt gleichzeitig das lokale Verbesserungspotential des Vorschlags wieder:

$calcSat_{AGENT} : [0..1]^{\gamma} \rightarrow [0..1]$

Mit der Funktion *requireImp* entscheidet der Agent anhand des berechneten Zufriedenheitswerts, ob er Verbesserungspotential für einen Vorschlag meldet. Dabei vergleicht er möglicherweise den Zufriedenheitswert mit den entsprechenden Werten, die er für Vorschläge in seinem Gedächtnis berechnet hat.

$requireImp_{AGENT} : [0..1] \times MEMORY_{AGENT} \rightarrow \{true, false\}$

selSatProp ist eine Operation, mit der der Agent aus der Menge von Vorschlägen im Pool oder aus den Vorschlägen in seinem Gedächtnis einen auswählt, mit dem er besonders zufrieden ist und der zur Rekombination herangezogen werden soll:

$selSatProp_{AGENT} : POOL \times MEMORY_{AGENT} \rightarrow PROPOSAL$

Mittels *selPlan* wählt der Agent unter den verfügbaren Plänen einen aus, dessen Ausführung zur Verbesserung eines Vorschlags führen soll. Die Auswahl erfolgt auf Basis der Gütekoeffizienten, die für den Vorschlag berechnet wurden:

$selPlan_{AGENT} : [0..1]^{\gamma}, \{PLAN\} \rightarrow PLAN$

alterProp ist eine Operation, mit der der Agent auf Basis eines Ausgangsvorschlags und eines Variationsplans einen neuen Vorschlag erstellt:

$alterProp_{AGENT} : PROPOSAL \times PLAN \rightarrow PROPOSAL$

Mit der Operation *combineProps* kombiniert ein Agent einen Vorschlag mit einem weiteren Vorschlag zu einem neuen Vorschlag:

$combineProps_{AGENT} : PROPOSAL \times PROPOSAL \rightarrow PROPOSAL$

MEMORY ist das Gedächtnis eines Agenten. Es besteht aus einer Menge mit höchstens μ Elementen. Die Elemente sind jeweils Paare aus Vorschlägen und den entsprechenden Zufriedenheitswerten des Agenten (vgl. POOL, 5.3.2):

$MEMORY_{AGENT} = \{(PROPOSAL, [0..1])\}_{0..\mu}$

rememberProp ist eine Operation, mit der der Agent entscheidet, ob ein Vorschlag in sein Gedächtnis aufgenommen werden soll. Dazu vergleicht der Agent seine Zufriedenheit mit dem Vorschlag mit den entsprechenden Werten der Vorschläge im Gedächtnis:

$rememberProp_{AGENT} : [0..1] \times MEMORY_{AGENT} \rightarrow \{true, false\}$

5.3.7 Der Diskussionsleiter (*MODERATOR*)

Der Diskussionsleiter (*MODERATOR*) wird als 5-Tupel definiert:

$$MODERATOR = (aggregateEvaluation,\ selectProp,\ chooseModifier,\ updateState,\ checkTermination,\ rejectProps)$$

aggregateEvaluation ist eine Aggregationsfunktion, mit der der Diskussionsleiter aus den lokalen Evaluationswerten aller Agenten (*evalProp*) einen globalen Fitness-Wert aggregiert, wobei er optional einen zusätzlichen Rest-Evaluationswert einfließen lässt, den er für den Vorschlag berechnet (vgl. 5.2.2).

$aggregateEvaluation_{MODERATOR} : (\mathbb{R}^{\epsilon})^{\alpha} \times PROPOSAL \rightarrow \mathbb{R}$

selectProp ist eine Funktion, mit der der Diskussionsleiter einen zu verbessernden Vorschlag aus dem Vorschlagspool auswählt:

$selectProp_{MODERATOR} : POOL \rightarrow PROPOSAL$

chooseModifier ist eine Funktion, mit der der Diskussionsleiter für einen Vorschlag einen Agenten auswählt, der diesen variieren soll. Dazu werden die Zufriedenheitsfunktionen der Agenten ausgewertet:

$chooseModifier_{MODERATOR} : PROPOSAL \times \{AGENT\}_{\alpha} \rightarrow AGENT$

removeProps bezeichnet eine Selektionsfunktion, mit der der Diskussionsleiter Vorschläge aus dem Vorschlagspool entfernt, um Platz für neue Vorschläge der nächsten Generation zu schaffen. Im Pool verbleiben nur die λ besten Vorschläge (Elite):

$removeProps_{MODERATOR} : POOL \times \lambda \rightarrow POOL$

Die Methode *updateStatusAndCheckTermination* führt der Diskussionsleiter durch, wenn die Evaluation von Lösungsvorschlägen abgeschlossen ist. Er passt damit die Statusvariablen an und überprüft, ob Abbruchbedingungen zutreffen.

$updateStatusAndCheckTermination_{MODERATOR} :$
$POOL \times STATUS \rightarrow STATUS \times \{true, false\}$

5.3.8 Ablauf der Genetischen Konferenz als formalisierter Algorithmus

Zusammenfassend kann der Ablauf einer Genetischen Konferenz wie folgt angegeben werden:

Ablauf der Genetischen Konferenz
 # (0) Deklaration und Initialisierung
 # (1) Erzeugen von Initialvorschlägen durch die Agenten
while abbruch $\neq$ true
    # (2) Bewertung der Vorschläge durch die Agenten und
    # Aggregation durch den Moderator
    # (3) Selektion von Vorschlägen durch den Moderator
    # (4) Variation von Vorschlägen durch die Agenten
    # (5) Statusvariablen setzen und Abbruchbedingungen überprüfen
    durch den Moderator
end while

Der Deklarations und Initialisierungsabschnitt könnte verkürzt etwa wie folgt aussehen:

0: Deklaration und Initialisierung (verkürzt)
$a_1..a_\alpha : AGENT$;
$mem_{a_1}..mem_{a_\alpha} : MEMORY$
$at_1..at_\beta : ATTRIBUTE$;
$v_1..v_\xi : VALUE$;
$pl_1..pl_\delta : PLAN$
$asys : AGENTSYSTEM(\{a_1..a_\alpha\}, \{at_1..at_\beta\}, \{v_1..v_\xi\}, \{pl_1..pl_\delta\})$
$pool : POOL$
$status : STATUS$
$mod : MODERATOR$;
$gk : GENETIC - CONFERENCE(asys, pool, mod, status)$
[...]
abbruch = false

Die genetische Konferenz beginnt mit der Erzeugung von Initialvorschlägen durch die Agenten. Dazu wird ϕ mal ein zufälliger Agent ag ausgewählt, der Mittels der Operation $initProp_{ag}$ einen Vorschlag erzeugt und diesen in den Pool $pool$ einbringt:

1: Erzeugen von Initialvorschlägen
(0)
while $|pool| < \phi$
    ag = select arbitrarily an Agent from $a_1..a_\alpha$
    $pool = pool \cup (initProp_{ag}(), -\infty)$
end while
# (2)-(5)

Nach der Initialisierung des Vorschlagspool wird eine Schleife ausgeführt, bis eine der Abbruchbedingungen zutrifft. Dabei werden zunächst alle Vorschläge aus dem Vorschlagspool von allen Agenten partiell bewertet und (individuell) besonders günstig eingestufte Vorschläge in den Gedächtnissen der Agenten abgelegt:

2: Bewertung von Vorschlägen
# (0),(1)
abbruch = false
while abbruch ≠ true
  foreach (p, r) in *pool* **do**
    if $(r == -\infty)$
      $val = aggregateEvaluation_{mod}(evalProp_{a_1}(p), ..., evalProp_{a_\alpha}(p), p)$;
      $pool = pool \setminus (p, r)$
      $pool = pool \cup (p, val)$
      foreach AGENT ag in asys **do**
        $sat = calcSat_{ag}(calcRatios(p, evalProp_{ag}(p)))$
        if $rememberProp_{ag}(sat, mem_{ag})$
          $mem_{ag} = mem_{ag} \cup (p, sat)$
        end if
      end do
    end if
  end do
  # (3)-(5)
end while

Im Anschluss werden alle bis auf die λ besten Vorschläge (*Elite*) mittels der Methode *removeProps* aus dem Pool entfernt, um Platz für neue zu schaffen:

3: Selektion von Vorschlägen
# (0),(1)
abbruch = false
while abbruch ≠ true
  # (2)
  if $|pool| > \lambda$
    $pool = removeProps_{mod}(pool, \lambda)$
  end if
  # (4),(5)
end while

Die Agenten erzeugen dann durch Variation der bisherigen Vorschläge neue Vorschläge und bringen diese in den Pool ein. Dabei wird zunächst ein zufälliger Agent mit der Variation eines ausgewählten Vorschlags beauftragt. Meldet der Agent kein Verbesserungspotential an (*requireImp*), wird der Variationsauftrag an einen anderen Agenten delegiert, den der Diskussionsleiter mittels $chooseModifier$ anhand der Zufriedenheitswerte auswählt:

4: Variation von Vorschlägen
# (0),(1)
abbruch = false
while abbruch $\neq$ true
    # (2),(3)
    while $|pool| < \phi$
        $p = selectProp_{mod}(pool)$
        a = select arbitrarily an Agent from $a_1..a_\alpha$
        unless $requireImp_a(calcSat_a(calcRatios(p, evalProp_a(p))), mem_a)$
            $a = chooseModifier_{mod}(p)$
        end unless
        $pnew = alterProp_a(p, selPlan(calculateRatios_a(p), \{pl_1..pl_\delta\}))$
            or $combineProps_a(p, selSatProp_a(pool, mem_a))$
        $pool = pool \cup (pnew, -\infty)$
    end while
    # (5)
end while

Zu den in Abschnitt 5.2.1 angegebenen Schritten kommt noch die Überprüfung von Abbruchbedingungen hinzu. Nach der Variation von Vorschlägen[4] werden die Statusvariablen angepasst. Der Algorithmus endet, wenn beispielsweise ein Generationenzähler unter den Statusvariablen die maximale Generationenzahl überschreitet oder eine andere Abbruchbedingung zutrifft. Andernfalls wird wieder zur Evaluation übergegangen:

5: Anpassen der Statusvariablen, Überprüfen der Abbruchbedingung
# (0),(1)
abbruch = false
while abbruch $\neq$ true
    # (2)-(4)
    $status, abbruch = updateStatusAndCheckTermination_{mod}(pool, status)$
end while

Aus Gründen der besseren Lesbarkeit wurde der Ablauf des Algorithmus hier in sequentieller Form angegeben. Ein paralleler Ablauf der Bewertung und der Variation von Vorschlägen wird besonders unter dem Gesichtspunkt der Effizienz in Abschnitt 6.2.4 noch diskutiert.

[4]Theoretisch wäre die Überprüfung der Abbruchbedingungen auch bereits nach der Evaluationsphase möglich, sie wurde hier aus rein ästhetischen Gründen an das Ende der Schleife gelegt.

5.4 Konzepte im Detail

5.4.1 Verwaltung der Entscheidungsvariablen durch Agenten

Mit der Genetischen Konferenz sollen Kombinatorische Optimierungsprobleme bearbeitet werden. Bei KOP besteht die Aufgabenstellung in der Anordnung, Zuordnung, Gruppierung oder Auswahl von Objekten (vgl. 2.5). Bei einer intuitiven Lösungsrepräsentation und Kodierung der Entscheidungsvariablen kann jede Variable genau einem dieser Objekte zugeordnet werden (vgl. 3.1.3). Die Genetische Konferenz basiert auf der individuellen Repräsentation der Objekte durch die Agenten des beteiligten Multiagentensystems: Jeder Agent verwaltet die Entscheidungsvariable(n), die die Konfiguration „seines“ Objekts bestimmt (bzw. bestimmen).

Beispiel TSP:
TSP-Exemplare können mit der Genetischen Konferenz beispielsweise auf Grundlage der in Abschnitt 2.3 vorgeschlagenen Lösungsrepräsentation bearbeitet werden. Jedem Agent A_i wird eine Stadt s_i zugeordnet und er verwaltet somit die entsprechende Entscheidungsvariable $pos(s_i)$. Beim TSP wird nur eine einzelne Entscheidungsvariable (die Positionsnummer in der Rundreise) zur Konfiguration eines Objekt (einer Stadt) benötigt.

Auch wenn im obigen Beispiel ein „Satz“ von Entscheidungsvariablen ausreicht, könnten auch mehrere Variablen zur Beschreibung der Konfiguration eines Objekts in einem Lösungsvorschlag benötigt werden. In diesem Fall muss der Agent mehrere Entscheidungsvariablen verwalten.

Betrachtet man den allgemeinen Fall eines KOP mit n Objekten und p Entscheidungsvariablen, die (bei intuitiver Kodierung) zur Beschreibung *eines einzelnen Objekts* notwendig sind, so ergibt sich eine Zielfunktion $f(x_{11}, ..., x_{1p}, ...x_{n1}, ..., x_{np})$ mit $n \cdot p$ Entscheidungsvariablen. Die Agenten der Genetischen Konferenz repräsentieren die Objekte, es werden also n Agenten $A_1, ..., A_n$ benötigt und jeder dieser Agenten A_i verwaltet die p Entscheidungsvariablen $x_{i1}, ..., x_{ip}$.

5.4.2 Ablaufschritt 1: Initialisierung von Lösungsvorschlägen

Zur Initialisierung des Vorschlagspool erstellen die Agenten Lösungsvorschläge, bei denen jeweils die Konfiguration des individuell verwalteten Objekts möglichst günstig ist. Dadurch kann eine große Variationsbreite im Pool erreicht werden. Stehen entsprechende Methoden nicht zur Verfügung, können alternativ auch zufällige Lösungen eingebracht werden. Im Unterschied zu den übrigen Ablaufschritten erfolgt die Initialisierung nur einmal zu Beginn des Verfahrens.

5.4.3 Ablaufschritt 2: Bewertung der Lösungsvorschläge durch die Agenten

In der Genetischen Konferenz werden Lösungsvorschläge nicht (nur) global, sondern auch partiell bewertet (Ablaufschritt 2). Die partielle Bewertung wird von den Agenten durchgeführt und dient drei Zielen:

Ziel 1: Durch die partielle Bewertung soll das Verbesserungspotential eines Lösungsvorschlags ermittelt werden, das sich bei Variation der Konfiguration des vom Agenten repräsentierten Objekts ergibt.

Ziel 2: Die partielle Bewertung hinsichtlich unterschiedlicher Kriterien soll die Auswahl eines Variationsoperators ermöglichen, mit dem der Vorschlag verbessert werden kann.

Ziel 3: Durch Aggregation der partiellen Bewertungen aller Agenten soll die Güte des Vorschlags insgesamt berechenbar sein.

Dabei ist zu beachten, dass die Evaluation von Lösungsvorschlägen eventuell aufwändig ist, und die mehrfache Berechnung derselben Zwischenergebnisse daher möglichst vermieden werden sollte. Aus diesem Grund findet die Evaluation in mehreren *Teilschritten* statt, wobei die Zwischenergebnisse der Schritte jeweils gespeichert werden, um sie für spätere Schritte wiederverwenden zu können. Abbildung 5.6 zeigt die Schritte sowie Verwendungsmöglichkeiten der jeweiligen Zwischenergebnisse. Die Schritte werden zunächst im Überblick dargestellt und im Anschluss detaillierter erläutert.

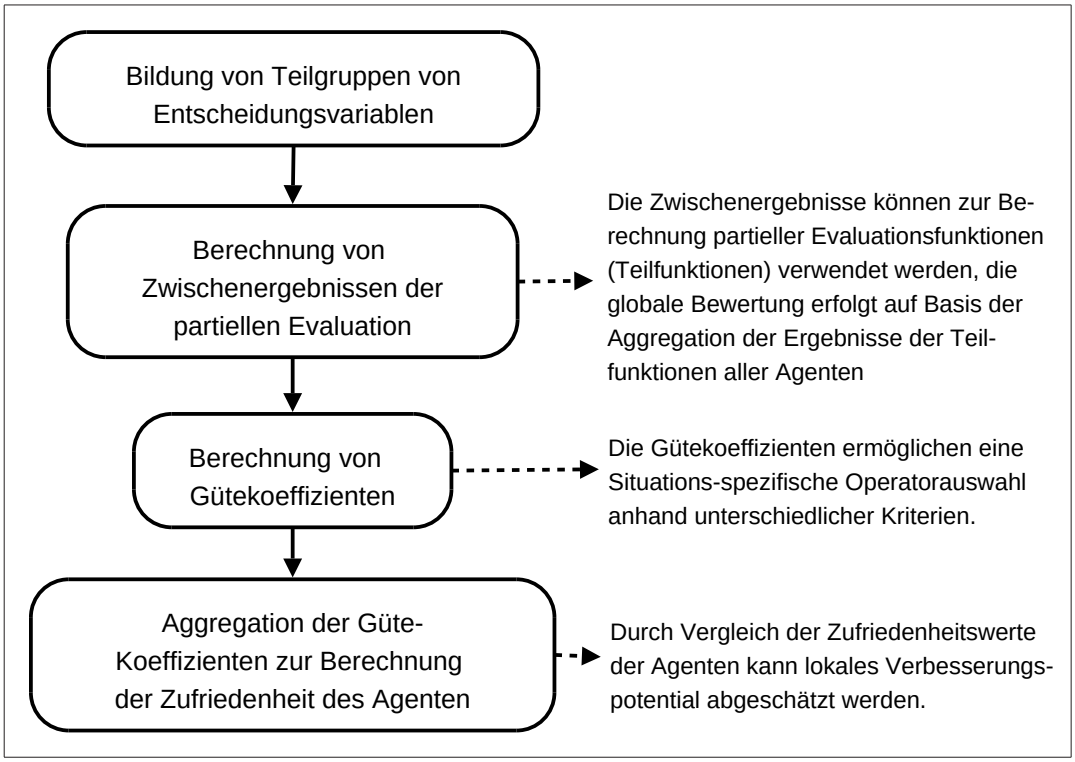


Abbildung 5.6: *Schritte bei der Bewertung von Lösungsvorschlägen*

Evaluationsschritt 1: Die Zielfunktion $f(x_{11}, .., x_{np})$ wird in *Teilfunktionen* $t_1(T_1), ..., t_n(T_n)$ über möglichst kleine Teilgruppen T_i von Entscheidungsvariablen mit $\{x_{i1}, ..., x_{ip}\} \in T_i$ zerlegt, so dass die Zielfunktion $f(x_{11}, .., x_{np})$ (zum großen Teil) als Aggregation von Teilfunktionen $t_i(T_i)$ über diese Teilgruppen berechenbar ist. Diese Zerlegung ermöglicht eine nebenläufige *partielle Evaluation* von Lösungsvorschlägen durch die Agenten zur Auswertung der Zielfunktion (Ziel 3). Aber auch die Erkennung von Verbesserungspotential (vgl. Ziel 1) und die Auswahl von Variationsoperatoren (Ziel 2) kann auf Grundlage der Teilgruppe durchgeführt werden.

Evaluationsschritt 2: Teilweise müssen zum Erreichen der drei Ziele die selben Berechnungsschritte durchgeführt werden. Daher werden die Teilfunktionen nicht direkt, sondern in zwei Schritten berechnet, wobei die *Zwischenergebnisse* des ersten Berechnungsschritts auch für die Ziele 2 und 3 weiterverwendet werden können. Diese *Evaluations-Zwischenergebnisse* $z_{i1}, ..., z_{ik}$ werden von den Agenten durch Auswertung von *Abbildungen* a_i berechnet, wobei $(z_{i1}, ..., z_{ik}) = a_i(T_i)$ und $t_i(T_i) = \sum_j z_{ij}$.

Evaluationsschritt 3: Die Agenten berechnen *Güte-Koeffizienten* $g_{i1}..g_{il}$ durch Anwendung einer *Abbildung* $(g_{i1}..g_{il}) = g_i(z_{i1}..z_{ik})$ auf die Evaluations-Zwischenergebnisse. Mit jedem Gütekoeffizienten wird die Zufriedenheit des Agenten mit der Konfiguration seines Objekts *bezüglich eines Kriteriums* gemessen. Durch die Berechnung und den Vergleich der Güte-Koeffizienten wird eine *spezifische Auswahl eines passenden Variationsoperators* möglich, die Güte-Koeffizienten können andererseits zu individuellen (Gesamt-)Zufriedenheitswerten *aggregiert* werden.

Evaluationsschritt 4: Die Agenten berechnen *individuelle Zufriedenheitswerte* für die Vorschläge im Lösungspool durch Aggregation der entsprechenden Güte-Koeffizienten. Dadurch kann das lokale *Verbesserungspotential* abgeschätzt werden.

Evaluationsschritt 1: Dekomposition der Zielfunktion

Als Grundlage für die spätere Berechnung der Güte-Koeffizienten (und des lokalen Verbesserungspotentials) bietet es sich an, einen möglichst großen Teil der Zielfunktion $f(x_{11}, ..x_{np})$ des Problemexemplars zunächst als Summe von Teilfunktionen $t_1, .., t_n$ aufzufassen, deren Funktionswert hauptsächlich von den Entscheidungsvariablen $x_{i1}, ..., x_{ip}$ abhängig ist, die ein einzelner Agent A_i verwaltet – notfalls kann jeweils eine *möglichst kleine* Untermenge U_i weiterer Entscheidungsvariablen einfließen.[5] Die Variablen, von denen t_i abhängt, sollen im Folgenden als *Teilgruppe* T_i bezeichnet werden. Die Berechnung der Teilfunktionen $t_i(T_i)$ wird von jeweils einem einzelnen Agenten übernommen, und die Gesamtgüte eines Vorschlags ergibt sich entsprechend aus der Aggregation dieser Einzelbewertungen aller Agenten und einer möglichen Restfunktion:

$$\underbrace{f(x_{11}, ..., x_{np})}_{\text{Zielfunktion}} = \underbrace{t_1(T_1) + ... + t_n(T_n)}_{\text{partielle Bewertungsfunktionen (Teilfunktionen)}} + \underbrace{f_{REST}(u_{REST})}_{\text{Restfunktion}}$$

$$T_i = \{x_{i1}, ..., x_{ip}\} \cup U_i;\ U_i \subseteq \{x_{11}, ..., x_{np}\};\ |U_i| \text{ möglichst klein}$$

In der Regel sind die Teilfunktionen nicht vollständig voneinander unabhängig – dies wäre auch nur möglich, wenn die Teilgruppen tatsächlich disjunkt wären ($\forall i : U_i = \varnothing$) und keine Nebenbedingungen an die Kombination der Belegungen von Entscheidungsvariablen aus unterschiedlichen Gruppen gestellt werden.

Die Dekomposition der Zielfunktion stellt die Grundlage der folgenden Evaluationsschritte dar und muss im Unterschied zu diesen nicht explizit bei der Bewertung jedes Lösungsvorschlags neu durchgeführt werden.

[5] Eine Teilfunktionen t_i ist mit dem Fitness-Beitrag f_i für einen Genort im NK-Fitness-Modell von Kaufmann [Kau93] vergleichbar. Dort ist der Fitness-Beitrag f_i jeweils vom Gen x_i sowie K weiteren der insgesamt N Gene abhängig (vgl. 4.4.3). Für Probleme mit geringer Epistasie (wie den Anwendungsbeispielen des zu entwickelnden Verfahrens) ist K eher klein gegenüber N.

Evaluationsschritt 2: Berechnung von Evaluations-Zwischenergebnissen

Die partiellen Bewertungsfunktionen $t_i(T_i)$ werden von den Agenten nicht direkt berechnet, sondern wiederum als Aggregation von *Evaluations-Zwischenergebnissen* $z_{i1}, ..., z_{ik}$ aufgefasst, die der entsprechende Agent durch Auswertung einer Abbildung $a_i : T_i \mapsto z_{i1}, ..., z_{ik}$ ermittelt. In der formalen Definition der Genetischen Konferenz wurden die Abbildungen zur Berechnung der Zwischenergebnisse mit *evalProp* bezeichnet. Im einfachsten Fall ergibt sich der Wert der Teilfunktion als *Summe*[6] der Zwischenergebnisse:

$$t_i(T_i) = \sum_{j=1}^{k} z_{ij} \text{ mit } (z_{i1}, ..., z_{ik}) = a_i(T_i)$$

Die bei der Berechnung der Teilfunktionen anfallenden und gespeicherten Zwischenergebnisse können für auch für die Berechnung der Güte-Koeffizienten herangezogen werden. Die Aufspaltung der Zielfunktion in partielle Einzelfunktionen, die unabhängig voneinander ausgewertet werden können, ermöglicht außerdem die *Parallelisierung der Evaluation einzelner Lösungsvorschläge* (vgl. Abschnitt 6.2.4).

Zur globalen Bewertung eines Lösungsvorschlags werden die Ergebnisse der Teilfunktionen aller Agenten aggregiert. Zur Variation werden vorrangig Lösungsvorschläge herangezogen, die *insgesamt* gut evaluiert wurden – ebenso wie beim Standard-GA kann dies beispielsweise durch ein Turnierverfahren gelöst werden. Eine Erweiterung zum Standard-GA ist die zielgerichtete Auswahl der Variationsstelle (Ziel 1) und des Variationsoperators (Ziel 2), die auf Basis von Güte-Koeffizienten und Zufriedenheitswerten der Agenten erfolgt (Schritte 3 und 4).

Beispiel TSP:
Wieder soll die unter 2.2 vorgeschlagene Kodierung verwendet werden. Die Agenten übernehmen – wie in 5.4.1 beschrieben – jeweils eine Stadt und verwalten die entsprechende Positionsnummer. Die Zielfunktion lässt sich darstellen als Summe von Teilfunktionen, bei denen jeweils nur die Entfernung von einer Stadt s_i zu den in der Rundreise direkt zuvor und direkt im Anschluss besuchten Städten $pred(s_i)$ und $succ(s_i)$ berechnet wird. Die Teilgruppe T_i besteht also jeweils nur aus den Positionsnummern dreier Städte s_i, $pred(s_i)$ und $succ(s_i)$ und ist für verschiedene Lösungsvorschläge unterschiedlich besetzt.
Der Agent A_i berechnet zunächst zwei Evaluations-Zwischenergebnisse z_{i1} und z_{i2}, und zwar die Entfernung zur Vorgängerstadt $z_{i1} = dist(s_i, pred(s_i))$ und die zur Nachfolgerstadt $z_{i2} = dist(s_i, succ(s_i))$. Diese Zwischenergebnisse werden für die spätere Berechnung von Güte-Koeffizienten zwischengespeichert.
Die vom Agenten zu berechnende Teilfunktion T_i steuert zur Gesamtbewertung eines Vorschlags jeweils die Summe der Zwischenergebnisse $z_{i1} + z_{i2}$ bei. Da so jede Entfernung in der Rundreise doppelt gezählt wird, sollte jedoch noch mit dem Faktor 1/2 normiert werden. Eine Restfunktion wird beim TSP nicht benötigt.

[6] Es sind natürlich auch andere Formen der Aggregation (beispielsweise gewichtete Summen) denkbar.

Evaluationsschritt 3: Berechnung von Güte-Koeffizienten für die spezifische Auswahl von Variationsoperatoren

Die Auswahl des Variationsoperators erfolgt auf Grundlage von *Güte-Koeffizienten*, die der mit der Variation eines Vorschlags beauftragte Agent für die aktuellen Belegungen der Variablen seiner Teilgruppe ermittelt. Dazu kann er auf die vorher berechneten Evaluations-Zwischenergebnisse zurückgreifen. Die dafür zuständige Funktion wurde in der formalen Definition *calcRatios* genannt.

Mit den Güte-Koeffizienten kann die Qualität der Konfiguration des vom Agenten verwalteten Objekts hinsichtlich unterschiedlicher Kriterien gemessen werden, sie müssen zwischen 0 (minimale Qualität) und 1 (maximale Qualität) skaliert sein.

Bei der Berechnung der Güte-Koeffizienten kann vorhandenes *a-priori*-Wissen das Anwenders eingebracht werden. Die Güte-Koeffizienten dienen einerseits zur *Auswahl möglicher Variationsoperatoren*. Je nach Ausprägung der unterschiedlichen Koeffizienten kann später der Operator passend zur jeweiligen Situation gewählt werden. Da die Evaluations-Zwischenergebnisse gespeichert werden, wird dabei kein zusätzlicher Rechenaufwand nötig. Andererseits dienen die Koeffizienten zur Ermittlung der Zufriedenheit des Agenten und darauf aufbauend der Abschätzung des lokalen Verbesserungspotentials (Schritt 4).

Beispiel TSP:
Beim TSP kann ein Agent beispielsweise je einen Güte-Koeffizienten für die Entfernung zur Vorgängerstadt und Nachfolgerstadt berechnen. Der Anwender bringt hier sein *a-priori*-Wissen ein, dass verhältnismäßig lange An- und Abreisewege für die Konstruktion einer möglichst kurzen Rundreise hinderlich sind.
Der Agent bestimmt für die Berechnung der Koeffizienten zunächst (unabhängig vom konkreten zu bewertenden Vorschlag) die Entfernung $d_{sec-min}$ zur zweitnächsten Nachbarstadt *überhaupt*.
Der Güte-Koeffizient für den Anreiseweg nimmt genau dann den Wert 1 an, wenn die Entfernung zur Vorgängerstadt höchstens so lang ist, wie diese Entfernung – d.h. die Vorgängerstadt unter den beiden nächsten Nachbarn ist. Ansonsten wird ihr das Verhältnis zwischen $d_{sec-min}$ und $dist(s_i, pred(s_i))$ zugewiesen. Analog wird die zweite Koeffizient für den Abreiseweg berechnet. Die Distanzen zu Vorgänger- und Nachfolgerstadt liegen bereits als Evaluations-Zwischenergebnisse vor.
Die Berechnung der beiden Güte-Koeffizienten für An- und Abreiseweg ermöglicht eine spezifische Auswahl des Variationsoperators. Je nach Ausprägung der Koeffizienten kann hier beispielsweise entschieden werden, ob bei Kantentausch eher die Verbindung zur Vorgänger- oder zur Nachfolgerstadt „gekappt“ werden sollte.

Evaluationsschritt 4: Berechnung von Zufriedenheitswerten zur Abschätzung lokalen Verbesserungspotentials

Für die Auswahl der *Variationsstelle*, d.h. der zu variierenden Entscheidungsvariablen, sollen die Agenten das *lokale Verbesserungspotential* eines Vorschlags abschätzen, d.h. die Güte der Konfiguration des vom Agenten verwalteten Objekts.

Ein Agent ermittelt seine *Zufriedenheit* mit einem Vorschlag durch Auswertung einer *Zufriedenheitsfunktion*, die die individuellen Güte-Koeffizienten aggregiert und wiederum einen Wert zwischen 0 (nicht zufrieden) und 1 (zufrieden) zurück liefert. Diese Zufriedenheitsfunktion – in der formalen Definition *calcSat* genannt – spiegelt gleichzeitig das lokale Verbesserungspotential wieder. Vorschläge werden vorrangig von Agenten variiert, die Verbesserungspotential melden.

Die Zufriedenheitsfunktion wird auch herangezogen, um zu entscheiden, ob ein Vorschlag in das Gedächtnis eines Agenten aufgenommen werden soll. In Abschnitt 5.4.6 wird näher auf das Konzept „Evolution mit Gedächtnis“ eingegangen.

Beispiel TSP:
Werden die Güte-Koeffizienten auf die oben vorgeschlagene Weise berechnet, so kann die Zufriedenheitsfunktion einfach das Produkt der beiden Koeffizienten zurück liefern. Der Agent ist genau dann vollständig zufrieden, wenn die Stadt zwischen den nächsten Nachbarn liegt. In diesem Fall nimmt die Zufriedenheitsfunktion den Wert 1 an und der Vorschlag birgt kein lokales Verbesserungspotential durch Variation der vom Agenten verwalteten Variablen.
Die Zufriedenheitsfunktion nimmt umso niedrigere Werte an, je länger An- und Abreiseweg im Verhältnis zum zweitnächsten Nachbarn sind. Ist ein Agent mit einem Vorschlag unzufrieden, so besteht lokales Verbesserungspotential, d.h. der durch den Agenten verwalteten Stadt wurde *einzeln betrachtet* noch keine optimale Positionsnummer in der Rundreise zugewiesen.

5.4.4 Ablaufschritt 3: Selektion von Lösungsvorschlägen

Die Selektion von Lösungsvorschlägen für den Generationenübergang wird in der formalen Definition von der Funktion *removeProps* des Diskussionsleiters übernommen (Ablaufschritt 3). Der Ansatz entspricht dem so genannten *Generational Replacement mit Elite:* Ein oder mehrere Vorschläge, die vorher die beste aggregierte Bewertung erhalten hatten, werden in die nächste Generation übernommen (in der formalen Definition beträgt die Anzahl λ). Der Pool wird dann mit neuen Vorschlägen aufgefüllt.

Möglich wäre auch ein vollständiges Ersetzen der Vorgängergeneration *(Generational Replacement)*, bei dem alle Vorschläge der alten Generation gelöscht werden und der Vorschlagspool nur mit neuen Vorschlägen gefüllt wird ($\lambda = 0$). Nachteilig ist dann, dass die Güte des besten Lösungsvorschlags im Pool nicht mehr monoton steigt.

Die Verdrängung ähnlicher Lösungsvorschläge

Um die Streuungsbreite unter den Lösungsvorschlägen im Vorschlagspool zu erhöhen, wurde darüber hinaus mit Methoden zur Verdrängung ähnlicher Lösungsvorschläge experimentiert.[7] Die Agenten können jeweils entscheiden, ob ein neuer Lösungsvorschlag in der Konfiguration des von ihnen verwalteten Objekts signifikante Abweichungen gegenüber anderen Vorschlägen im Pool aufweist. Ist die Anzahl der Abweichungen gering, so kann man den insgesamt schlechter evaluierten Vorschlag verdrängen. Wie bei den in Abschnitt 4.3.5 beschriebenen Nischentechniken ist das Ziel, frühzeitige Konvergenz auf eines oder wenige lokale Optima zu vermeiden. Mit diesem

[7] Diese Technik wird auch als *crowding* bezeichnet (vgl. z.B. [MF00, S.322]).

Konzept konnten teilweise Beschleunigungen des Verfahrens erreicht werden, daher wurde es bei der Implementierung der Genetischen Konferenz berücksichtigt.

Der Schutz „innovativer“ Vorschläge

Des Weiteren wurden Experimente zur Priorisierung „innovativer“ Vorschläge gemacht. Unter einem „innovativen“ Vorschlag soll dabei ein solcher verstanden werden, der bezüglich der *partiellen* Bewertung bestimmter Agenten allen oder sehr vielen anderen Vorschlägen überlegen ist. Die Messung der „Durchsetzungsfähigkeit“ beim Konkurrenzkampf der Lösungsvorschläge um die Plätze im Pool beschränkt sich dann nicht nur auf *einen* (möglicherweise aus partiellen Bewertungen) aggregierten Wert, wie das beim Standard-GA der Fall ist. Stattdessen können Vorschläge, die partiell besonders vielversprechende, ansonsten selten vorkommende Eigenschaften (Belegungen von Entscheidungsvariablen) aufweisen, zusätzlich bevorzugt werden. Die Idee erinnert an Pareto-basierte Ansätze für Multikriterielle Probleme (vgl. Abschnitt 4.3.5), wenn man die individuellen Bewertungen der Agenten als Teilziel-Funktionen für die einzelnen „Kriterien“ ansieht.

Das Konzept wurde zwar testweise umgesetzt, allerdings konnte keine signifikante Beschleunigung des Verfahrens bewirkt werden. Im Gegenteil scheint die Priorisierung innovativer Vorschläge die Produktion insgesamt guter Lösungen eher zu behindern. Andere mögliche Erweiterungen der Selektion speziell für die Bearbeitung multikriterieller Probleme werden im Ausblick (Abschnitt 9.2.2) beschrieben.

5.4.5 Ablaufschritt 4: Variation von Lösungsvorschlägen

Die Variation von Lösungsvorschlägen (Ablaufschritt 4 der formalen Definition) erfolgt in vier (Unter-) Schritten:

Variationsschritt 1: Auf Basis der Zielfunktions-Werte wird ein zu variierender Vorschlag ausgewählt.

Variationsschritt 2: Auf Basis des individuell bestimmten lokalen Verbesserungspotentials wird ein Agent ausgewählt, der mit der Variation des Vorschlags beauftragt wird. Implizit wird dadurch eine Entscheidung darüber getroffen, welche Entscheidungsvariablen verändert werden sollen.

Variationsschritt 3: Der beauftragte Agent entscheidet, ob der Vorschlag durch Anwendung eines lokalen Variationsoperators („*Mutation*“) oder durch Rekombination mit einem zweiten Vorschlag („*Crossover*“) variiert werden soll.

Variationsschritt 4a: Zur Variation des Vorschlags durch *Mutation* wählt der Agent auf der Basis der zuvor berechneten Güte-Koeffizienten und einer Situation-Plan-Matrix einen Variationsplan aus und führt diesen durch. Der Plan besteht in der Anwendung eines oder mehrerer lokaler Verbesserungsoperatoren.

Variationsschritt 4b: Zur Variation des Vorschlags durch *Rekombination* wählt der Agent einen weiteren Ausgangsvorschlag, mit dem er selbst zufrieden ist, und kombiniert diesen mit dem ersten.

Variationsschritt 1: Auswahl eines Elternvorschlags zur Variation

Die Auswahl eines zu variierenden Vorschlags entspricht der Selektion von Individuen für den *mating pool* (vgl. A.3) beim klassischen GA. Dort werden *insgesamt* bessere Lösungen mit höherer Wahrscheinlichkeit rekombiniert und mutiert als schlechtere. Dabei kommen beispielsweise Roulette-Wheel-Verfahren oder rangbasierte Selektionsverfahren zur Anwendung. Auch bei der Genetischen Konferenz wird zunächst *ein* zu variierender Vorschlag anhand eines Turnierverfahrens basierend auf der (aggregierten) Güte der Vorschläge ausgewählt. Die entsprechende Funktion wurde bei der formalen Definition *selectProp* genannt. Die Auswahl des bei einer Rekombination benötigten zweiten Ausgangsvorschlags erfolgt erst später (Schritt 4b.).

Variationsschritt 2: Auswahl des Variationsorts

Im Unterschied zum klassischen GA wird die Variationsstelle nicht zufällig, sondern auf Grundlage des lokalen Verbesserungspotentials bestimmt. Zwar wäre eine statische Auswahl des jeweils „unzufriedensten“ Agenten für die Variation des Vorschlags möglich, diese könnte jedoch zu unerwünschter frühzeitiger Stagnation des Verfahrens führen (weil die Vorschläge im Pool immer von denselben Agenten variiert werden). Stattdessen wird folgendermaßen vorgegangen:

Ein zur Variation ausgewählter Vorschlag wird zunächst einem zufälligen Agenten zur Variation überlassen. Der Agent prüft (in der formalen Definition mit der Methode *requireImp*), ob er noch Verbesserungspotential erkennt. Dazu zieht er den aus den Güte-Koeffizienten aggregierten Zufriedenheitswert heran.[8]

Wenn der Agent kein Verbesserungspotential meldet, wird der Vorschlag an einen anderen, unzufriedeneren Agenten delegiert. Dieser Agent wird wiederum – ähnlich wie bei der Auswahl des Vorschlags selbst – auf Basis eines Turnierverfahrens bestimmt. Mittels einer speziellen Funktion, die in der formalen Definition *chooseModifier* genannt wurde, wählt der Diskussionsleiter aus einer zunächst zufällig bestimmten *Teilmenge* der Agenten denjenigen aus, der die geringste Zufriedenheit mit dem Vorschlag (und damit das größte Verbesserungspotential) meldet. Dieser Agent wird mit der Variation des Vorschlags beauftragt.

Auf diese Weise ist nicht nur für Selektionsdruck im Pool der Lösungsvorschläge gesorgt. Zusätzlich wird die Chance, einen selektierten Vorschlag auch tatsächlich verbessern zu können, durch die vorherige Identifikation einer vielversprechenden Verbesserungsstelle erhöht.

Beispiel TSP:
Einem Agenten soll bei der Bearbeitung des TSP bevorzugt eine Rundreise zur Variation überlassen werden, die zwar insgesamt relativ kurz ist, bei der jedoch die von ihm verwaltete Stadt mit einem vergleichsweise langen Anreise- bzw. Abreiseweg angebunden ist, wenn man die Entfernungen benachbarter Städte betrachtet. In diesem Fall meldet der Agent Verbesserungspotential, ansonsten wird der Vorschlag an einen anderen Agenten delegiert.

[8] Den Zufriedenheitswert kann er zusätzlich mit *dem bisher maximal erreichten* Zufriedenheitswert vergleichen, der ja in seinem Gedächtnis abgelegt ist. Verbesserungspotential besteht, wenn die Differenz der Werte hoch ist. So kann wiederum der beschriebenen Gefahr der frühzeitigen Stagnation beigekommen werden, weil nur dann Verbesserungspotential gemeldet wird, wenn die Erreichbarkeit eines höheren Zufriedenheitswerts bereits (empirisch) nachgewiesen wurde.

Variationsschritt 3: Entscheidung über Mutation oder Crossover

Beim Standard-GA finden Mutationen (pro Bit) meist nur sehr selten, und nur in Verbindung mit einer Rekombination statt. Bei der Genetischen Konferenz werden die Variationsoperatoren „*Mutation*“ und „*Crossover*“ stattdessen getrennt voneinander als eigenständige Suchoperatoren aufgefasst.[9] Der mit der Variation eines Vorschlags beauftragte Agent entscheidet „selbstständig“ (anhand vom Anwender vorgegebener Wahrscheinlichkeiten), ob ein neuer Vorschlag durch Mutation eines Lösungsvorschlags oder durch Rekombination zweier Lösungsvorschläge[10] aus der Elterngeneration erzeugt werden soll.

Variationsschritt 4a: Auswahl und Anwendung von Variationsoperatoren zur lokalen Verbesserung

Die Variation eines Lösungsvorschlags durch Anwendung lokaler Variationsoperatoren erfolgt in vier Teilschritten:

4a-1. *Initialisierung des neuen Vorschlags:* Zunächst kopiert der Agent den ihm zur Variation überlassenen Ausgangsvorschlag. Durch die später ausgeführten lokalen Variationen wird er in der Kopie die Belegungen einiger Variablen ersetzen, die von ihm verwaltet werden. Zusätzlich wird er meist einige Belegungen anderer Variablen anpassen müssen, die von „benachbarten“ Agenten verwaltet werden, um einen gültigen und möglicherweise auch global besseren Vorschlag zu produzieren. Dabei handelt es sich beispielsweise um Variablen, die sich zusätzlich in der Teilgruppe für die partielle Evaluation befinden. Bei der Bearbeitung von KOP mit Raumbezug sind das oft Entscheidungsvariablen *räumlich benachbarter Objekte*. Die Agenten, die benachbarte Objekte verwalten, werden daher hier und im Folgenden als „*benachbarte Agenten*“ bezeichnet.

4a-2. *Bestimmung der individuellen Situation:* Im nächsten Schritt ermittelt der Agent aufgrund der Auswertung der Güte-Koeffizienten, die er für den zu variierenden Lösungsvorschlag berechnet hat, in welcher „*Situation*“ er sich befindet. Die möglichen Situationen sowie die Art und Weise ihrer Bestimmung in Abhängigkeit der Güte-Koeffizienten werden vom Anwender im Vorfeld spezifiziert.

4a-3. *Auswahl eines Verbesserungsplans:* Der Anwender macht außerdem Vorgaben, welche *Pläne* die Agenten zur Verbesserung von Vorschlägen verfolgen können. Des Weiteren spezifiziert er mittels einer Matrix die jeweiligen *Wahrscheinlichkeiten* für die Durchführung der Pläne *in Abhängigkeit der Situationen*. Diese Wahrscheinlichkeits-Matrix wird im Folgenden als *Situations-Plan-Matrix (SP-Matrix)* bezeichnet. Tabelle 5.2 entspricht einer exemplarischen SP-Matrix für die Bearbeitung von TSP-Exemplaren. Der Agent wählt anhand der Wahrscheinlichkeiten einen durchzuführenden Plan aus.

4a-4. *Ausführung des Verbesserungsplans:* Weiterhin muss der Anwender angeben, welche konkreten lokalen Variationsoperatoren bei der Durchführung eines bestimmten Plans zur Anwendung kommen sollen. Der Agent führt die Operationen dann in der angegebenen Rei-

[9] Diese Vorgehensweise findet man auch bei Memetischen Algorithmen (vgl. 4.8.2), so z.B. in [Mer00].

[10] Momentan ist die Rekombination auf zwei „Elternvorschläge“ beschränkt - eine Erweiterung auf n-Elter-Rekombinationen wäre jedoch möglich.

henfolge aus. Anschließend trägt er in speziell vorgesehenen Feldern des Vorschlags eine Beschreibung seiner ursprünglichen Situation sowie den zur Verbesserung des Vorschlags ausgeführten Plan ein. Diese Protokoll-Informationen dienen später dazu, Auswertungen über die Nützlichkeit bestimmter Pläne in spezifischen Situationen machen zu können, und die Wahrscheinlichkeitsmatrix eventuell entsprechend anzupassen (vgl. Kapitel 4.7, „lernen“).

	Kreuzung auflösen	neuen Vorgänger suchen	neuen Nachfolger suchen	Zufälligen Positionstausch durchführen
Agent ist zufrieden	0.25	0.25	0.25	0.25
Distanz zum Vorgänger ist vergleichsweise lang	0	0.75	0	0.25
Distanz zum Nachfolger ist vergleichsweise lang	0	0	0.75	0.25
Anbindung hat Überkreuzung	0.75	0	0	0.25

Tabelle 5.2: *Eine exemplarische Situations-Aktions-Matrix für das TSP*

Beispiel TSP:
Für die Variation eines Lösungsvorschlags könnten die Agenten unter den Plänen „zufälliger Positionstausch zweier Städte“, „Einfügen einer neuen Vorgängerstadt“, „Einfügen einer neuen Nachfolgerstadt“ und „Auflösen einer Kreuzung“ wählen. Situationen der Agenten könnten „zufrieden“, „Distanz zum Vorgänger vergleichsweise lang“ „Distanz zum Nachfolger vergleichsweise lang“ und „Anbindung hat Überkreuzung“ sein. Als SP-Matrix käme dann beispielsweise die Tabelle 5.2 Frage.

Können vom Anwender keine verschiedenen, in unterschiedlichen Situationen der Agenten einzusetzenden Operatoren angegeben werden, so kann die situationsbedingte Auswahl der Variationsoperatoren natürlich auch zugunsten eines einzelnen Operators entfallen. Dieser eventuell sogar rein zufallsbasierte Operator wird dann in jedem Fall angewendet. Auch in diesem Fall sind theoretisch Beschleunigungen gegenüber einem „klassischen“ GA zu erwarten, da ja wenigstens bei der Auswahl der Variations*stelle* Problemwissen einbezogen wurde. In Testläufen des entwickelten Verfahrens (vgl. Abschnitt 8.2) konnten solche Beschleunigungen für zwei untersuchte TSP-Exemplare auch empirisch nachgewiesen werden.

Variationsschritt 4b: Auswahl eines zweiten Ausgangsvorschlags und Durchführung der Rekombination

Für die Variation eines Vorschlags durch Rekombination muss der Agent zunächst einen zweiten Ausgangsvorschlag auswählen. Wiederum versucht er dabei, sein Wissen über die Qualität der Konfiguration des von ihm verwalteten Objekts bei diesem Vorgang einzubeziehen: Neben dem

Vorschlag, der dem Agenten zur Variation übergeben wurde (und der lokales Verbesserungspotential aufweist) zieht er zur Rekombination einen Vorschlag heran, bei er mit der Konfiguration seines Objekts besonders zufrieden ist.

Dieser zweite Vorschlag wird von einer – in der formalen Definition *selSatProp* genannten – Funktion entweder ebenfalls aus dem Vorschlagspool oder aus dem Gedächtnis der Agenten ausgewählt. Zusätzlich besteht die Möglichkeit, den zweiten Vorschlag aus den Koalitionen zu rekrutieren, die der Agent vorher eingegangen ist. Diese Möglichkeit wurde bei der formalen Definition der Genetischen Konferenz aus Gründen der Übersichtlichkeit vorerst nicht eingeführt und wird in Abschnitt 5.4.7 näher besprochen.

Der Agent versucht dann, das Verbesserungspotential des ersten Vorschlags auszuschöpfen, indem er die Belegungen der von ihm (und eventuell von benachbarten Agenten) verwalteten Entscheidungsvariablen des ersten Vorschlags durch die des zweiten Vorschlags ersetzt. Der Anwender muss für die Rekombination einen Crossover-Operator spezifizieren, der die Belegungen aus dem ersten Vorschlag auf diese Weise mit denen des zweiten Vorschlags kombiniert – in der formalen Definition wurde diese Funktion *combineProps* genannt.

Beispiel TSP:
Mit Abbildung 5.7 soll das Vorgehen eines Agenten bei der Rekombination von Lösungen eines TSP-Exemplars verdeutlicht werden. Der mit der Variation beauftragte Agent befindet sich oben (schwarz ausgefüllt). Dem Agenten wird zur Rekombination bevorzugt eine insgesamt kurze aber lokal, d.h. „für ihn selbst“ ungünstige Rundreise (analog zur oben beschriebenen Auswahl zur Mutation) überlassen (1). Als zweiten Ausgangsvorschlag wählt er eine lokal günstige Rundreise, in der die von ihm verwaltete Stadt über relativ geringe Entfernungen „angebunden“ ist (2). Er versucht dann, diese so zu kombinieren, dass die lokal ungünstige „Anbindung“ im ersten Vorschlag durch die lokal günstige Teilreise des zweiten Vorschlags ersetzt wird. Dazu kopiert er den ersten Vorschlag und setzt die „günstigen“ Kanten des zweiten Vorschlags ein (3). Um einen gültigen Vorschlag zu erhalten, müssen gleichzeitig einige der „alten“ Kanten entfernt werden (in der Zeichnung gestrichelt dargestellt). Zuletzt muss unten eine zusätzliche Kante eingefügt werden, um eine gültige Lösung zu erhalten (4).

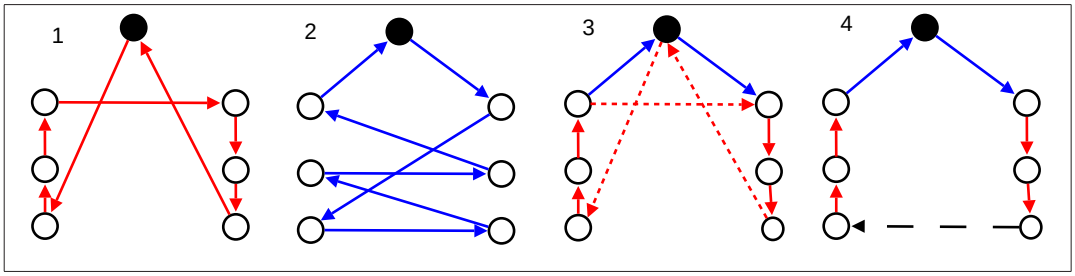


Abbildung 5.7: *Die Rekombination von Vorschlägen am Beispiel des TSP*

Selbstverständlich ist es für einen Agenten nicht immer möglich, das Verbesserungspotential des Ausgangsvorschlags durch Mutation oder Rekombination auszuschöpfen, ohne dass sich die partiellen Bewertungen anderer Agenten (und damit eventuell auch der Zielfunktionswert des neuen

Vorschlags insgesamt) verschlechtern, da die partiellen Bewertungsfunktionen im Allgemeinen ja nicht vollständig unabhängig voneinander sind (siehe auch Abschnitt 4.4.2, Epistasie). In Tests für zwei TSP-Exemplaren ergeben sich dennoch deutliche Beschleunigungen, wenn die Agenten lokales Verbesserungspotential für die Konfiguration ihrer Objekte berechnen und Lösungen primär an den Stellen variieren, an denen solches besteht (vgl. Abschnitt 8.3).

Erweiterungen

Erweiterungen des Ablaufschritts 4 stellen die Konzepte „Evolution mit Gedächtnis“, „Bildung von Koalitionen“ und „lernende Agenten“ dar, die in den nächsten Unterabschnitten besprochen werden. Die beiden letzteren Konzepte wurden bei der formalen Definition zunächst nicht berücksichtigt.

5.4.6 Evolution mit Gedächtnis (Ablaufschritt 2 und 4)

Das Konzept der „Evolution mit Gedächtnis“ drückt sich in der formalen Definition der Genetischen Konferenz in den Schritten 2 und 4 aus: Bei der Bewertung der Vorschläge können individuell besonders günstige Vorschläge von den Agenten speziell gespeichert werden und später im Variationsschritt (Schritt 4) – im Unterschied zum Standard-GA – auch dann noch zur Rekombination herangezogen werden, wenn sie nicht (mehr) im Lösungspool enthalten sind.

Ein Standard-GA dagegen verfügt (wie die natürliche Evolution) über keinerlei Gedächtnis und arbeitet immer nur auf den Lösungen der derzeitigen Generation. Einmal verdrängte Lösungsvorschläge können also nur wieder in die Population gelangen, wenn sie zufällig neu erzeugt werden. Dadurch können auch *günstige* Teillösungen – d.h. Konfigurationen einer Teilmenge der Objekte – verloren gehen, bei denen die Konfiguration anderer Objekte zu einer schlechten (Gesamt-)Bewertung geführt hatte. Je nach Stärke der *Epistasie* könnten solche Teillösungen – kombiniert mit anderen guten Teillösungen – aber eigentlich noch zu guten Gesamtlösungen des Problems beitragen. Ein Verlust guter Teillösungen verlangsamt dann den Optimierungsprozess.

Bearpark und Keane [BK05] beschreiben den Nutzen eines *gemeinsamen* Gedächtnisses der Individuen, um den Verlust guter Teillösungen bei der *Genetischen Programmierung* zu vermeiden. Im zentralen Gedächtnis werden vom besten Lösungsvorschlag – der bei der Genetischen Programmierung ja ein *Programm* repräsentiert – Sub-Programme bis zu etwa 20% der maximalen Länge der Populationsmitglieder abgelegt. Diese Sub-Programme stehen allen Individuen zur Verfügung und können als Mutation – statt des sonst verwendeten zufälligen Materials – in andere Programme eingefügt werden.

Bei der Genetischen Konferenz wird ein ähnlicher Ansatz verfolgt, allerdings verfügt hier jeder Agent über ein *eigenes*, *individuelles* Gedächtnis. Die Agenten können Vorschläge, mit denen sie besonders zufrieden waren, in ihrem Gedächtnis ablegen. So können sie auch nach der Verdrängung der Vorschläge aus dem Vorschlagspool auf die entsprechenden Variablenbelegungen zurückgreifen, um sie mit den Belegungen der übrigen Variablen aus einem anderen Vorschlag zu kombinieren (Ablaufschritt 4 der formalen Definition). Abgelegt werden zwar ganze Vorschläge, bei der Rekombination wird jedoch primär auf die Belegung der Variablen zurückgegriffen, die der Agent selbst oder einer seiner „Nachbarn“ verwaltet (Siehe 5.4.5, Schritt 4b). Ob ein Vorschlag in das Gedächtnis eines Agenten aufgenommen wird, entscheidet eine in der formalen

Definition *rememberProp* genannte Funktion, die am Ende des Ablaufschritts 2 (Bewertung) zur Ausführung kommt. Als Kriterium für die Aufnahme dient dabei die *Zufriedenheit* des Agenten mit dem Vorschlag.

Betrachtet man das zu bearbeitende KOP als multikriterielles Problem (vgl. Abschnitt 2.3) und fasst die Konfiguration eines Objekts als jeweils eigenständiges Kriterium auf (vgl. Abschnitt 5.4.4), so lassen sich die Zufriedenheitsfunktionen der Agenten jeweils als Teilziel-Funktionen für die einzelnen Kriterien interpretieren. Im Gedächtnis eines Agenten werden dann Vorschläge abgelegt, die hinsichtlich *einer* dieser Teilziel-Funktionen besonders gut evaluiert wurden. Da die Gedächtnisse der Agenten jeweils eigene Sub-Populationen von Vorschlägen darstellen, weist die Genetische Konferenz diesbezüglich Analogien auf zum VEGA-Ansatz von Schaffer [Scha85], der bei der Selektion ebenfalls eigene Subpopulationen für jedes Kriterium vorsieht (vgl. 4.3.5), die er im Anschluss zu einer gemeinsamen Population zusammenfasst.

Beim implementierten Prototyp kann sowohl die Länge der Gedächtnisliste als auch die Wahrscheinlichkeit, mit der auf einen Vorschlag aus dem Gedächtnis anstatt aus dem Vorschlagspool zurückgegriffen werden soll, durch einen Optimierungsparameter vom Anwender gesteuert werden. Durch die Nutzung von Gedächtnissen konnte die Performanz des Optimierungsverfahrens in Testläufen vor allem zu Beginn der Optimierung gesteigert werden (vgl. Kapitel 8).

5.4.7 Bildung von Koalitionen (Erweiterung der Ablaufschritte 2 und 4)

Das Konzept der Bildung von Koalitionen unter den Agenten wurde bei der formalen Definition der Genetischen Konferenz aus Gründen der Übersichtlichkeit zunächst nicht eingeführt. Es stellt eine Erweiterung dar, die ähnlich wie das Konzept der Gedächtnisse auf der Speicherung lokal besonders günstiger Vorschläge und ihrer späteren Verwendung als Ausgangsvorschlag für Rekombinationen beruht.

Ziel des Konzepts ist es, günstige *Kombinationen* von Belegungen zusammengehörender Entscheidungsvariablen möglichst zu erhalten und nicht (beispielsweise durch ungünstige Rekombinationsoperationen) „versehentlich“ aufzuspalten. Im Sinne der Building-Block-Hypothese (vgl. Abschnitt 4.3.6) soll also versucht werden, die Konstruktion einer günstigen Gesamtlösung durch Kombination günstiger Teillösungen zu unterstützen.

Bei Problemen aus dem Anwendungsbereich der Genetischen Konferenz kann eine gewisse Dekomponierbarkeit vorausgesetzt werden. Die Dekomposition der Zielfunktion basiert jedoch bisher nur auf den Teilgruppen, die zum großen Teil aus den Entscheidungsvariablen *eines* Objekts bestehen. Möglicherweise können neben dieser Partitionierung auch Dekompositionen in *größere* Gruppen von Entscheidungsvariablen vorgenommen werden, so dass unter diesen Gruppen nur geringe *äußere* Epistasie besteht (vgl. Kapitel A.6). Beispielsweise ist bei KOP mit Raumbezug der wechselseitige Einfluss in Bezug auf die Zielfunktion unter Entscheidungsvariablen, die zu entfernten Objekten gehören, oft gering. Um eine solche Form der Dekomponierbarkeit auszunutzen, kann eine *Gruppe* von Agenten versuchen, eine besonders günstige *Kombination* der Konfigurationen ihrer Objekte durch Gründung einer *Koalition* zu erhalten.

Der Anwender muss dazu im Vorfeld spezifizieren, welche Agenten überhaupt untereinander Koalitionen eingehen dürfen. Im Laufe des Verfahrens entscheiden diese Agenten dann, ob eine Koalition gegründet werden soll, indem sie regelmäßig ihre Zufriedenheitswerte für neu eingebrachte Vorschläge aggregieren. Liegt ein Vorschlag vor, bei dem die aggregierte Zufriedenheit

von Kandidaten einer Koalition besonders hoch ist[11], so wird eine Koalition gegründet. Der Vorschlag wird als „Koalitionsbasis“ gesondert gespeichert. Die enthaltenen Belegungen können im späteren Verlauf der Optimierung direkt mit anderen Vorschlägen rekombiniert werden, auch wenn sich der Koalitionsbasis-Vorschlag nicht mehr im Vorschlagspool befindet.

Ähnlich wie beim Konzept der Gedächtnisse konnte die Performanz des Optimierungsverfahrens bei der Anwendung auf TSP-Exemplare in Testläufen zum Teil deutlich gesteigert werden (vgl. Abschnitt 8.3).

5.4.8 Lernende Agenten (Erweiterung der Ablaufschritte 2 und 4)

Eine Erweiterung der Auswahl des Verbesserungsplans (Variationsschritt 4a-3) stellt das Konzept der *lernenden Agenten* dar. Ziel dabei ist, das Ausmaß des Erfolgs oder Misserfolgs der Ausführung eines Plans in einer bestimmten Situation in die Auswahl zukünftiger Pläne einzubeziehen. Dieses Konzept wurde aus Gründen der Übersichtlichkeit in der formalen Definition zunächst nicht eingeführt.

In Kapitel 4 wurde auf Möglichkeiten eingegangen, die Agenten eines Multiagentensystems durch Feedback-Informationen *lernen* zu lassen, welche Aktionen sie in bestimmten Situationen durchführen sollten, um möglichst großen Nutzen zu erzielen.

Bei der Genetischen Konferenz ändert immer nur *ein* Agent *einen* Vorschlag ab, indem er zunächst seine Situation bestimmt und dann anhand einer Wahrscheinlichkeitstabelle (SP-Matrix) einen Plan auswählt, den er an diesem Vorschlag durchführt. Die Güte des Vorschlags wird vor und nach der Ausführung des Plans gemessen (Evaluation durch die Agenten). Die resultierende Güte-Differenz kann als Nutzen des ausgeführten Plans in der entsprechenden Situation aufgefasst werden und dem Agenten nach der Ausführung als „Feedback“ zurückgeliefert werden. Der Agent hat nun die Möglichkeit, die Werte in der Matrix anzupassen, so dass in bestimmten Situationen bisher besonders erfolgreiche Pläne in Zukunft öfter und weniger erfolgreiche Pläne seltener ausgeführt werden. Die Intensität der Anpassung kann durch die Spezifikation einer Lernrate variiert werden. Da weder mehrere Pläne ausgeführt werden, noch mehrere Agenten an den Änderungen beteiligt waren, tritt kein *Credit-Assignment-Problem* auf (vgl. Abschnitt 4.7).

Die Anpassung der Wahrscheinlichkeiten in der SP-Matrix kann sich einerseits zur Ermittlung günstiger Ausgangswerte zur Bearbeitung weiterer Problemexemplare der gleichen Problemklasse als nützlich erweisen – insbesondere dann, wenn dem Anwender nicht bekannt ist, welche Pläne in einer Situationen vorrangig ausgeführt werden sollen. Dazu kann das Verfahren mit einer Matrix gestartet werden, in der die Wahrscheinlichkeit für jeden Plan in jeder Situation gleich ist. Nach einer gewissen Zahl von Generationen pendeln sich die Werte im Regelfall bei einem stabilen Verhältnis ein. In folgenden Durchläufen können dann diese neuen Werte verwendet werden.

Durch die Möglichkeit der Agenten, die Wahrscheinlichkeiten im Laufe der Optimierung anzupassen, könnte andererseits auch der Tatsache Rechnung getragen werden, dass sich der Nutzen unterschiedlicher Operatoren während der Optimierung ändern kann. Nähere Untersuchungen hierzu wurden jedoch bisher noch nicht durchgeführt.

[11] Dabei wird ein vom Anwender einstellbarer Schwellenwert herangezogen.

5.5 Einordnung der Genetischen Konferenz im Kontext von MAS und GA

Abschließend soll die Genetische Konferenz noch einmal unter dem Blickwinkel der Ähnlichkeiten und Unterschiede mit den Basisalgorithmen *MAS* und *GA* betrachtet werden.

5.5.1 Die Genetische Konferenz als GA

Betrachtet man den Vorschlagspool sowie die Veränderungen, die an den Vorschlägen vorgenommen werden, kann die Genetische Konferenz als GA eingeordnet werden, der durch problemspezifische partielle Bewertungsfunktionen und Modifikationsoperatoren erweitert („hybridisiert“) wurde. Vom kanonischen GA wird konkret in folgenden Punkten abgewichen:

Lösungsrepräsentation: Die Lösungsrepräsentation erfolgt im Unterschied zum kanonischen GA nicht binär, sondern je nach Problem als beliebiges Datum, z.B. als natürliche oder rationale Zahl oder als Zeichenkette. Dadurch wird die Integration problemspezifischer Modifikationsoperatoren erleichtert.

Bewertung von Vorschlägen: Anders als beim Standard-GA wird die Güte eines Vorschlags nicht nur global, sondern auch partiell bezüglich der Belegungen einzelner Entscheidungsvariablen gemessen. Dies setzt eine gewisse Dekomponierbarkeit der Zielfunktion voraus. Die errechneten Güte-Koeffizienten helfen bei der Situations-spezifischen Operatorauswahl und können zur Ermittlung lokalen Verbesserungspotentials herangezogen werden.

Lösungspool: Die Genetische Konferenz arbeitet nicht nur auf einem zentralen Lösungspool. Zusätzlich werden Vorschläge in individuellen Gedächtnissen und bei der Bildung von Koalitionen zwischengespeichert. Damit sollen Vorschläge, die besonders günstige Konfigurationen für eines (oder wenige) der dem KOP zugrunde liegenden Objekte vorsehen, vor der Verdrängung bewahrt werden.

Modifikation von Vorschlägen: Bei der Modifikation von Vorschlägen ergeben sich wesentliche Unterschiede zum kanonischen GA. Zwar können wiederum Mutationen und Rekombinationen durchgeführt werden, allerdings werden diese als getrennte Suchoperatoren aufgefasst. Die Stelle im Lösungsvorschlag, auf die ein Mutationsoperator angewendet werden soll, wird nicht zufällig, sondern zielgerichtet auf Basis des lokalen Verbesserungspotentials bestimmt. Die Auswahl des Operators selbst erfolgt Situations-spezifisch anhand partieller Bewertungskoeffizienten, die ein Agent für die derzeitige Konfiguration seines Objekts berechnet. Bei einer Rekombination werden die zu übernehmenden Lösungsteile aus den Elternvorschlägen ebenfalls anhand der partiellen Bewertungen ausgewählt. Im Gegensatz dazu werden beim kanonischen GA nur randomisierte Crossover- und Mutationsoperatoren verwendet. Während man bei der Anwendung herkömmlicher GA also implizit annimmt, dass sich Lösungsvorschläge rekombinieren lassen und dass sich durch kleine Änderungen guter Lösungsvorschläge wieder gute Vorschläge produzieren lassen, kommt bei der Genetischen Konferenz explizites *a-priori*-Wissen über mögliches Verbesserungspotential und erfolgversprechende Variationsoperatoren zur Anwendung.

5.5.2 Die Genetische Konferenz als MAS

Die Agenten der Genetischen Konferenz können als lernende, zielbasierte Agenten mit internem Zustand (und Gedächtnis) eingeordnet werden. Abschließend folgt eine Betrachtung ihrer wesentlichen Eigenschaften:

Umgebung: Die Agenten sind in einer problemspezifischen Umgebung eingebettet (situiert), die aus den übrigen Agenten, dem Moderator und dem Vorschlagspool besteht.

Situation / Zustand: Zum *Zustand* eines Agenten kann zunächst sein individuelles Gedächtnis gezählt werden. Bei der Variation eines Vorschlags nimmt er außerdem eine *Situation* entsprechend der Güte-Koeffizienten ein, mit denen er die vorgeschlagene Konfiguration „seines“ Objekts hinsichtlich unterschiedlicher Kriterien bewertet hat. Der Variationsplan wird passend zur Situation ausgewählt.

Autonomie: Die Handlungen der Agenten bestehen in der (partiellen) Bewertung bisheriger Vorschläge, der Erzeugung neuer Vorschläge und der Bildung von Koalitionen. Sie werden *autonom* ausgeführt. Die Bewertungen erfolgen anhand individueller Maßstäbe. Bei der Erzeugung neuer Vorschläge steht insbesondere die Konfiguration des „eigenen“ Objekts im Vordergrund.

Proaktivität: Die Proaktivität der Agenten ist dadurch eingeschränkt, dass mit dem Diskussionsleiter eine zentrale Steuerungsinstanz existiert. Die Agenten können sich ihre Arbeit also nicht „selbst suchen“, die Aufgaben werden zentral verteilt. Allerdings können die Agenten Aufträge zur Variation von Vorschlägen an andere Agenten delegieren, wenn sie selber wenig Verbesserungspotential erkennen.

Ziele: Die Agenten verfolgen das Ziel, jeweils die individuelle Zufriedenheit mit den Vorschlägen im Lösungspool zu maximieren. Dazu können sie durch die Ausführung bestimmter Pläne (vorrangig) die Konfiguration des individuell verwalteten Objekts variieren. Die Ziele der einzelnen Agenten können sich widersprechen, wenn beispielsweise eine optimale Verteilung bestimmter Ressourcen Gegenstand der Optimierung ist.

Interaktion: Direkte Kommunikation unter den Agenten erfolgt durch Abfrage und Vergleich der Zufriedenheit mit bestimmten Vorschlägen bei der Delegation von Vorschlägen und der Bildung von Koalitionen. Die Interaktion über den gemeinsamen bearbeiteten Lösungspool weist gewisse Analogien mit der *Blackboard-Architektur* auf, allerdings werden ausschließlich vollständige Lösungen in den Pool eingestellt. Auftretende Konflikte bei der Verfolgung widersprüchlicher Ziele werden nicht durch Verhandlung unter den Agenten gelöst. Stattdessen können mehrere Lösungen parallel bearbeitet werden, wobei die Bewertungs- und Selektionsmechanismen eines Genetischen Algorithmus zum Einsatz kommen.

Lernfähigkeit: Die Lernfähigkeit der Agenten beschränkt sich auf die Anpassung der Wahrscheinlichkeitsmatrix, anhand derer sie in einer bestimmten Situation einen passenden Plan zur Verbesserung eines Lösungsvorschlags auswählen. Die Menge der ausführbaren Pläne kann bisher nicht erweitert werden.

Kapitel 6

Modellierung und Implementierung

Dieses Kapitel beschreibt die Modellierung und Implementierung des entwickelten Optimierungsverfahrens in einem Framework.

Der erste Abschnitt (6.1) erläutert kurz die Wahl des Objektorientierten Paradigmas.

Im Abschnitt 6.2 wird die Modellierung des Systempakets „Genetic Conference“ behandelt, das die Genetische Konferenz realisiert. Das Paket und seine Komponenten werden hier im Überblick beschrieben, detailliertere Informationen zu Inhalt und Aufbau der wichtigsten beteiligten Klassen des Pakets findet sich im Anhang.

In Abschnitt 6.3 wird die Einbettung des Pakets „Genetic Conference“ in ein Gesamtsystem und dessen prototypische Implementierung behandelt.

Das Kapitel schließt mit einigen Informationen zur JAVA-Implementierung (Abschnitt 6.4).

6.1 Verwendung des Objektorientierten Paradigmas

Die Modellierung erfolgt nach dem *objektorientierten* Paradigma. Dieses mittlerweile ohnehin als Standard zu bezeichnende Paradigma bietet sich im vorliegenden Fall unter anderem aus folgenden Gründen an:

- Im Hinblick auf die spätere *objektorientierte Implementierung* des Framework kann auf diese Weise von der Modellierung über den Systementwurf bis hin zur Implementierung ein einheitliches Konzept verwendet werden.
- Mit *Kapselung* und *Vererbung* bietet die objektorientierte Modellierung Konzepte, die notwendige problemspezifische Anpassungen des Framework für die Anwendung auf konkrete Problem(exemplar)e auf einfache Weise ermöglichen.
- Die *Identifikation der Klassen und Objekte* – sonst eher eine schwierige Aufgabe bei der objektorientierten Modellierung – ist in diesem Falle einfach. Sie entsprechen im Wesentlichen den Typen, die bereits in der formalen Definition verwendet wurden.
- Für die Modellierung eines Multiagentensystems bietet sich der objektorientierte Ansatz in besonderer Weise an, da sich die beteiligten Haupt-Entitäten (die Agenten) gut als Objekte modellieren lassen.

6.2 Das Paket *Genetic Conference*

In diesem Abschnitt werden zunächst grundsätzliche Überlegungen zur Modellierung eines Systempakets behandelt, das das entwickelte Optimierungsverfahren realisiert. Detailliertere Erläuterungen zu den einzelnen Klassen dieses Pakets finden sich im Anhang.

6.2.1 Schichten und Komponenten im Überblick

Die Genetische Konferenz wird als *ein Paket* eines *Softwaresystems zur Optimierung von KOP* aufgefasst. Das Paket trägt den Namen *Genetic Conference (GC)* und ist für die Durchführung von Optimierungsläufen verantwortlich. Ein weiteres Paket des Systems stellt die Benutzerschnittstelle *(User Interface, UI)* dar.[1] Die einzelnen Komponenten des GC-Pakets sind auf den drei Schichten *Datenschicht*, *Ausführungsschicht* und *Steuerungsschicht* angesiedelt:

In einer *Datenschicht* (*data layer*) finden sich die Komponenten „Vorschlags-Daten“ (*POOL*) und „Problemumgebung“ (*ENV*). In der Komponente *POOL* werden die Daten bereitgestellt, die den aktuellen Stand eines Optimiervorgangs repräsentieren. Zentrale Elemente dieser Komponente sind die Vorschlags-Objekte (*Proposal*), die in einem Vorschlagspool (*ProposalPool*) organisiert werden. Die Komponente stellt eine Schnittstelle *ManipulatablePool* bereit, über die auf den Vorschlagspool zugegriffen werden kann. Die Komponente ist problemunabhängig realisierbar. Problemspezifische Informationen werden dagegen in der Umgebungs-Komponente *ENV* gehalten und über eine Schnittstelle *DataSupply* bereitgestellt. Die zentrale Klasse der *ENV*-Komponente heißt *ProblemEnvironment*.

[1] Abschnitt 6.3 beschreibt die Einbettung der Pakete GC und UI in das Gesamtsystem

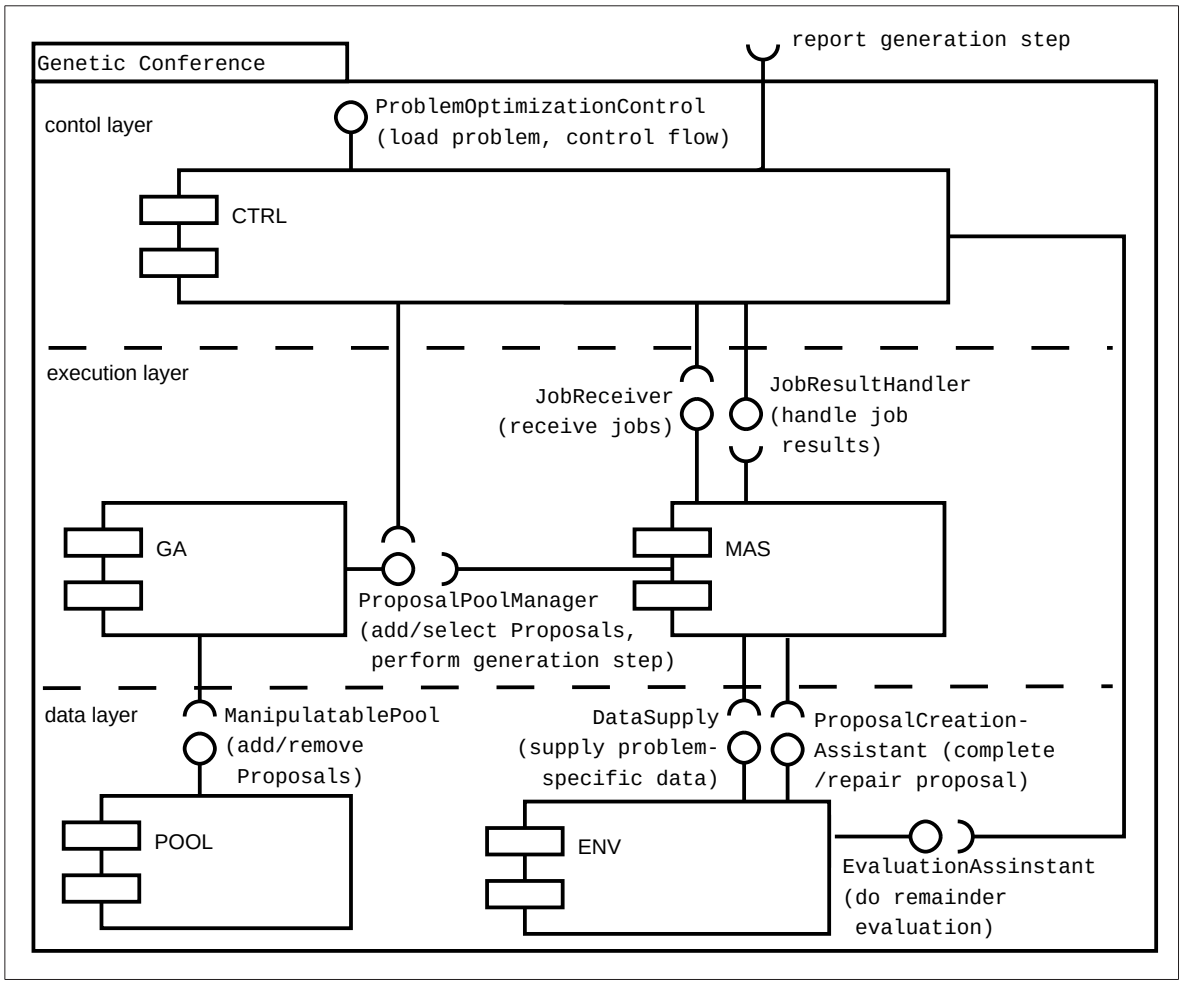


Abbildung 6.1: *Komponentendiagramm des Pakets Genetic Konference*

Der *Ausführungsschicht* (*execution layer*) sind wiederum zwei Komponenten zugeordnet. Eine GA-Operationen-Komponente (*GA*) stellt über die Schnittstelle *ProposalPoolManager* Operationen wie Selektion, Einfügen und Entfernen von Vorschlägen bereit. Bei der Ausführung solcher Operationen greifen die Klassen der Komponente auf die Schnittstelle *ManipulateablePool* der *POOL*-Komponente zu. Das Multiagentensystem findet sich als *MAS*-Komponente ebenfalls in der Ausführungsschicht. Die *MAS*-Komponente greift auf die Schnittstellen der *ENV*-Komponente zu und wird so mit problemspezifischen Informationen versorgt. Die Objekte der zentralen Agenten-Klasse dieser Komponente (*Agent*) nehmen die Initialisierung, Variation und Bewertung von Lösungsvorschlägen vor. Für diese Aktionen können Aufträge an die Komponente übertragen werden, die Ergebnisse werden über eine Schnittstelle bereitgestellt.

In der *Steuerungsschicht* (*control layer*) befindet sich die Steuerkomponente (*CTRL*). Die Notwendigkeit einer Steuerungsinstanz ergibt sich insbesondere aus der Modellierung des Ablaufs der Genetischen Konferenz in nebenläufigen Prozessen und den daraus resultierenden Synchronisationsaufgaben, die in Abschnitt 6.2.4 erläutert werden. Die Steuerkomponente steuert den Ablauf der Optimierung, indem ein zentrales Diskussionsleiter-Objekt (*Moderator*) über eine Schnitt-

stelle *JobReceiver* Aufträge an die Agenten vergibt. Die Ergebnisse solcher Aufträge werden vom Diskussionsleiter über eine zweite Schnittstelle *JobResultHandler* wieder entgegengenommen. Über Zugriffe auf die Schnittstelle *ProposalPoolManager* der *GA*-Komponente können Vorschläge in den Pool eingebracht sowie der Generationsübergang angestoßen werden. Die *CTRL*-Komponente bietet des Weiteren eine Schnittstelle *ProblemOptimizationControl* für die Initialisierung eines Problemexemplars, die Steuerung eines Optimierungslaufs (Starten und Stoppen der Optimierung) und den Abruf von Ergebnissen oder Zwischenständen der Optimierung. Diese Schnittstellen werden von Objekten des UI-Pakets genutzt (vgl. Abschnitt 6.3.3)

6.2.2 Problemspezifische Komponenten und deren Initialisierung

Das beschriebene Programmpaket *Genetic Conference* soll später in ein Framework zur Lösung beliebiger KOP mit Raumbezug eingebunden werden. Um die Problemunabhängkeit des Framework zu gewährleisten, müssen zwei Klassen des Pakets zunächst abstrakt gehalten werden, der Anwender muss Methoden dieser Klassen dann problemspezifisch implementieren.[2]

Betroffen sind die Klassen Agent (*MAS*-Komponente) und ProblemEnvironment (*ENV*-Komponente). Hier finden sich einige Methoden (wie beispielsweise die partiellen Bewertungsfunktionen der Agenten), die nicht ohne entsprechendes *a-priori*-Wissen über das zu bearbeitende Problem implementiert werden können. Alle übrigen Klassen der Komponenten des Pakets *Genetic Conference* sind problemübergreifend implementiert.

Die Initialisierung der Komponenten erfolgt über die Schnittstelle *ProblemOptimizationControl* der Komponente CTRL. Einer speziellen Methode wird dazu eine XML-Datei mit problemspezifischen Informationen übergeben, die zur Bearbeitung des Problems benötigt werden. Dazu gehören einerseits Optimierungsparameter, die für die Initialisierung der Klassen der Komponenten GA und POOL bekannt sein müssen (z.B. die Maximalzahl von Vorschlägen im Lösungspool). Die Parameter werden in Abschnitt 6.3.4 genauer beschrieben. Andererseits müssen die Namen nachzuladender Klassen angegeben werden. Die abstrakten Klassen *Agent* und *ProblemEnvironment* können *nicht* direkt instanziert werden. Stattdessen müssen problemspezifische Klassen geladen werden, die diese erweitern und ihre abstrakten Methoden implementieren.[3] Dabei darf nicht vergessen werden, dass die Komponente *Genetic Conference* in ein Gesamtsystem zur Optimierung *unterschiedlicher* KOP eingebunden werden soll. Welches Problembeispiel (bzw. welche Problembeispiele) mit diesem System bearbeitet wird (werden), sollte beim Start des Systems jedoch nicht immer bereits feststehen müssen. Daher können die problemspezifischen Klassen nicht bereits beim Systemstart geladen werden.[4] Um die Klassen für die Bearbeitung eines konkreten Problembeispiels nachladen zu können, müssen deren Namen in der XML-Datei spezifiziert werden (vgl. Abbildung 6.2).

[2] Neben diesen beiden Klassen des *Genetic Conference*-Programmpakets muss noch eine weitere Klasse der Benutzerschnittstelle problemspezifisch implementiert werden, die für die graphische Darstellung von Lösungsvorschlägen verantwortlich ist.

[3] Bei der detaillierten Beschreibung der einzelnen Klassen des Modells im nächsten Abschnitt wird noch genauer auf abstrakte Methoden und ihre problemspezifische Implementierung eingegangen. Im nächsten Kapitel findet sich außerdem ein Vorgehensmodell für die Implementierung der abstrakten Methoden.

[4] Es bietet sich daher eine Implementierung in einer Programmiersprache wie *Java* an, die das dynamische (Nach-)laden von Klassen (*Dynamic Class Loading*) erlaubt.

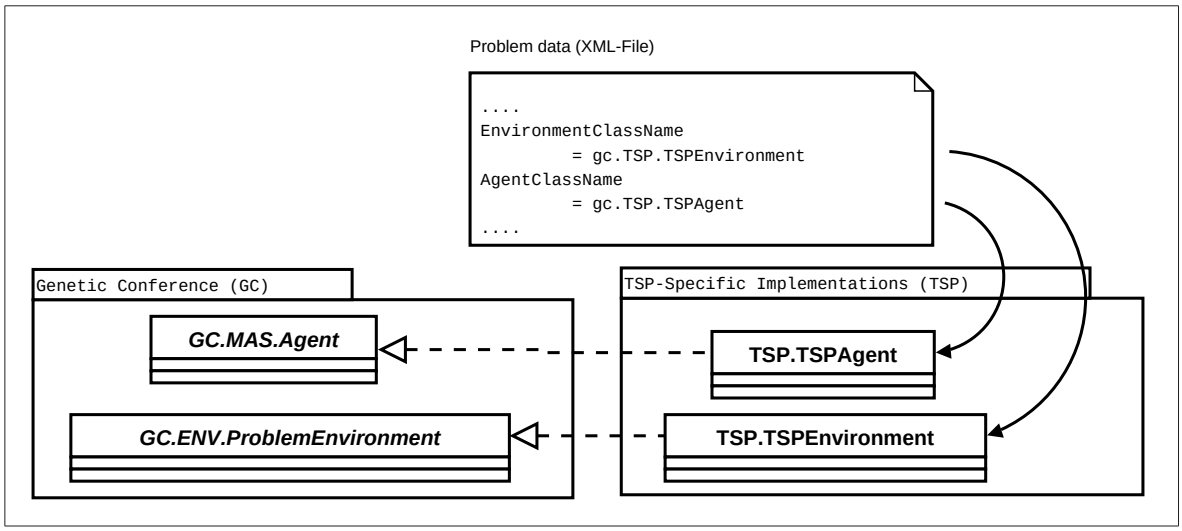


Abbildung 6.2: *Spezifikation nachzuladender Klassen über XML (Beispiel TSP)*

6.2.3 Der Aufbau eines Lösungsvorschlags *(Proposal)*

Ein Vorschlag (*Proposal*) vertritt die Rolle eines Individuums im Standard-GA. Hier werden alle Informationen gesammelt, die einen Lösungsvorschlag eines Problemexemplars charakterisieren. Bei der Genetischen Konferenz werden jeweils die Entscheidungsvariablen *eines* Objekts, dessen Konfiguration(en) Gegenstand der Optimierung sind, von den Agenten verwaltet. Die Variablen werden dazu in *Sätze* eingeteilt, so dass jeder Agent die Verwaltung genau einer Entscheidungsvariablen aus jedem Satz übernimmt (vgl. Abschnitt 5.4.1). In einem Lösungsvorschlag muss daher für *jeden der Agenten* und *jeden Satz von Entscheidungsvariablen* eine bestimmte Belegung gespeichert werden. Für die Belegungen werden *Objekte beliebigen Typs* zugelassen.

Zur Speicherung hält ein Vorschlag eine Zuordnungstabelle mit Schlüssel-Wert-Paaren – dies wurde bei der Java-Implementierung durch Erweiterung der Klasse `java.util.Hashtable` erreicht (vgl. dazu auch Abbildung B.1). Schlüssel sind die Agenten-Objekte der *MAS*-Komponente, die Werte sind Referenzen auf weitere, untergeordnete Zuordnungstabellen. In diesen untergeordneten Zuordnungstabellen werden Zeichenketten als Bezeichner für die verschiedenen Sätze von Entscheidungsvariablen als Schlüssel verwendet, die Werte sind die jeweiligen Belegungen dieser Variablen. Auf diese Weise ist für große Flexibilität gesorgt:

- Es können mehrere „Sätze“ von Entscheidungsvariablen verwendet werden. Ein Satz entspricht der Zuordnung eines beliebigen Datums zu jedem Agenten.

- Der Typ der Entscheidungsvariablen ist beliebig. Beschränkungen auf binäre oder reelle Kodierungen der Entscheidungsvariablen, wie sie beispielsweise beim kanonischen GA oder bei Evolutionsstrategien vorgesehen sind, entfallen. Stattdessen kann jeweils direkt der Variablentyp verwendet werden, der der „natürlichen“ Variablen im realen Problem am ehesten entspricht, oder es kann die Kodierung einzubeziehender problemspezifischer Heuristiken übernommen werden, wie dies Davis [Dav91] empfiehlt (vgl. Kapitel 4.8).

Beispiel TSP:
Ein Lösungsvorschlag für ein TSP mit n Städten besteht aus einer Zuordnungstabelle mit n Einträgen. Jedem der n Agenten wird eine untergeordnete Zuordnungstabelle für die Belegungen der Entscheidungsvariablen zugeordnet. In diesem Fall wird jedoch nur *ein* Satz von Entscheidungsvariablen benötigt – nämlich die „Position“ für jede durch den Agenten repräsentierte Stadt in der aktuellen Rundreise. Daher besteht die untergeordnete Zuordnungstabelle nur aus einem einzigen Schlüssel-Wert-Paar, bei denen jeweils unter dem Schlüssel „Position“ als Wert die Position der Stadt in der Rundreise eingetragen ist. Der Typ der Variablen „Position“ kann in diesem Fall ganzzahlig gewählt werden.

6.2.4 Ablauf der Optimierung: Modellierung mit nebenläufigen Prozessen

Die vier grundsätzlichen Ablaufschritte der Genetischen Konferenz (Initialisierung, Bewertung, Selektion und Variation) wurden bereits in Abschnitt 5.2 eingeführt. In der formalen Definition der Genetischen Konferenz wurde der Ablauf des Optimierungsverfahrens der Einfachheit halber jedoch zunächst als *sequenzieller* Algorithmus angegeben.

Als Grundlage für performante Implementierungen bietet sich aber eine Modellierung an, die es erlaubt, rechenintensive Operationen – soweit möglich – in unabhängigen, *parallelen* Prozessen ablaufen zu lassen. Zeitaufwändige Berechnungen ergeben sich insbesondere bei der *Variation* (in der formalen Definition durch die Methode *alterProp*) und der *partiellen Bewertung* (*calcSat* und *calcRatios*) von Vorschlägen. Diese Operationen können problemlos nebenläufig erfolgen.

Die partielle Bewertung (bzw. die Berechnung der Evaluations-Zwischenergebnisse) muss von jedem Agenten vorgenommen werden, bevor die aus den Einzelwerten aggregierte Gesamt-Güte eines Vorschlags bestimmt werden kann. Die *Reihenfolge* der Bewertungen ist jedoch beliebig, eine Verteilung der Berechnungen auf mehrere nebenläufige Prozesse ist auf einfache Weise möglich. In jeder Generation werden außerdem neue Vorschläge erzeugt. Die notwendigen Berechnungen sind jedoch ebenso unabhängig voneinander, daher kann die Erzeugung der Vorschläge auch parallel erfolgen. Die Variation und partielle Bewertung von Vorschlägen wird daher auf nebenläufige Prozesse verteilt, die jeweils einem einzelnen Agenten zugeordnet sind. Diese Form der Modellierung vereinfacht spätere Implementierungen, die Mehrprozessorsysteme performanzsteigernd auslasten und unterstreicht zudem noch in gewünschter Weise die Autonomie der Agenten. Allerdings treten bei einer Modellierung in nebenläufigen Prozessen stellenweise zusätzliche Koordinations- und Synchronisationaufgaben auf: Um zu verhindern, dass die maximal zulässige Anzahl neu eingebrachter Vorschläge überschritten wird, muss für eine Koordination zwischen der Erzeugung neuer Vorschläge und dem Generationsübergang (Ersetzen alter Vorschläge durch neue) gesorgt werden. Zur Durchführung des Generationsübergangs müssen zuvor alle Vorschläge bewertet worden sein. Dazu muss regelmäßig überprüft werden, ob ein Vorschlag bereits von allen Agenten partiell bewertet wurde und ein aggregierter Gesamt-Gütewert berechnet werden kann.

Diese Koordinationsaufgaben werden durch den *Moderator* aus der Kontrollkomponente als zentrale Steuerungsinstanz übernommen. Von ihm werden *Aufträge* an die Agenten zur *Initialisierung*, *Variation* und *lokalen Bewertung* von Vorschlägen verteilt. Diese Aufträge werden in individuelle Auftrags-Warteschlangen der Agenten eingestellt. Die Warteschlangen werden von den Agenten in nebenläufigen Prozessen abgearbeitet. Abbildung 6.3 zeigt das Zusammenspiel von Agent und Moderator bei der Auftragsvergabe und der Rückgabe der Ergebnisse.

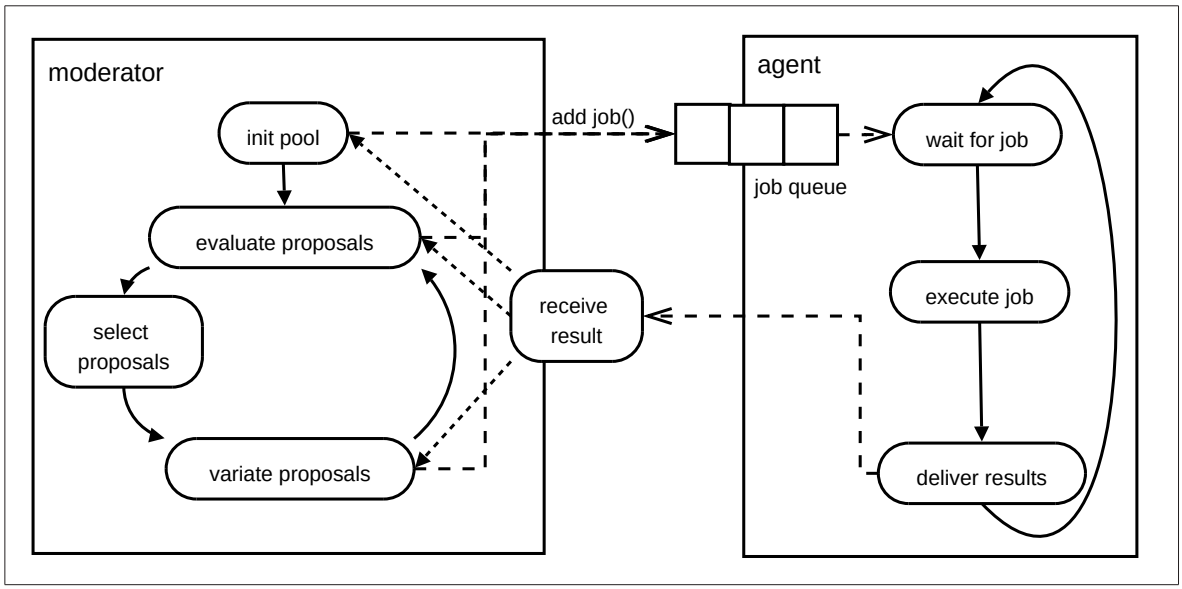


Abbildung 6.3: *Zusammenspiel von Diskussionsleiter und Agent bei der Vergabe von Aufträgen und der Rückgabe der Ergebnisse. Durchgezogene Pfeile stellen zeitliche Abfolgen dar, gestrichelte Pfeile deuten Übertragungen von Information an.*

Initialisierung von Vorschlägen

Abbildung 6.4 zeigt ein Aktivitätsdiagramm für die Initialisierung eines neuen Vorschlags zu Beginn einer Konferenz. Bevor die erste Generation von bewerteten Vorschlägen bereitsteht vergibt der Diskussionsleiter Aufträge zur Initialisierung von Vorschlägen an die Agenten in der *MAS*-Komponente, die *parallel* abgearbeitet werden. In der Abbildung ist der Übersichtlichkeit halber nur die Vergabe und Bearbeitung *eines* Auftrags dargestellt.

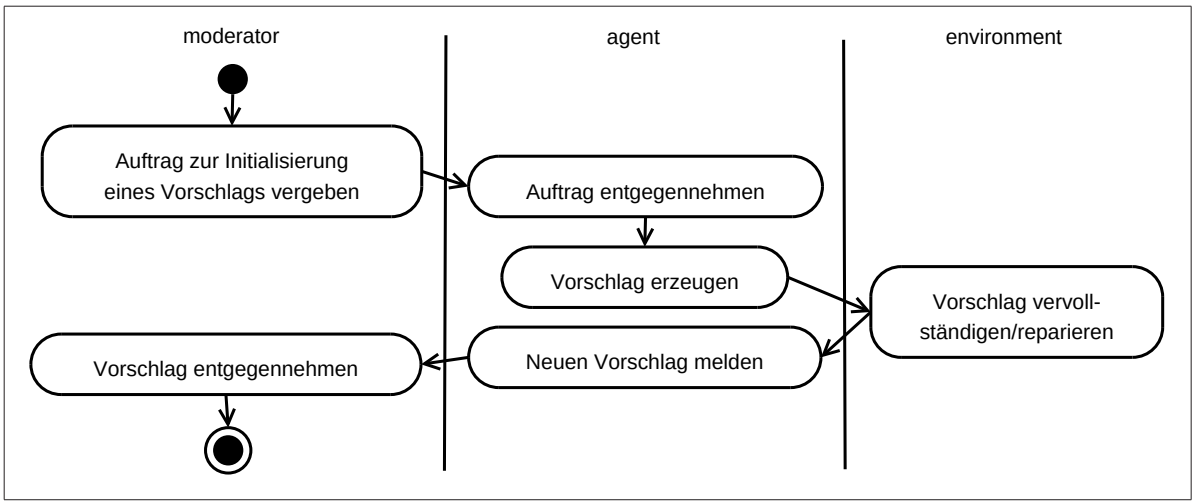


Abbildung 6.4: *Aktivitätsdiagramm für die Initialisierung eines neuen Vorschlags*

Der Auftrag wird in eine *Auftrags-Warteschlange* eingestellt, die vom Agenten abgearbeitet wird. Der beauftragte Agent erzeugt zunächst den Vorschlag – beispielsweise durch Anwendung einer problemspezifischen Konstruktionheuristik. Optional kann der Vorschlag von Methoden aus Klassen der *ENV*-Komponente im Anschluss vervollständigt oder repariert werden (damit beispielsweise sichergestellt ist, dass ein gültiger Vorschlag entstanden ist). Zuletzt meldet der Agent dem Diskussionsleiter die abgeschlossene Initialisierung. Der Diskussionsleiter kann dann mit der Vergabe von Aufträgen zur Bewertung des neuen Vorschlags fortfahren.

Bewertung von Vorschlägen

Der Diskussionsleiter verteilt nach Erhalt eines neuen Vorschlags an *alle* Agenten Aufträge zu seiner partiellen Bewertung. Die Agenten nehmen die partielle Bewertung *parallel* vor und melden die berechneten Zwischenergebnisse an den Diskussionsleiter zurück. Aufbauend auf den Zwischenergebnissen berechnen die Agenten auch ihre individuellen Güte-Koeffizienten sowie den Zufriedenheitswert und speichern die Ergebnisse in lokalen Tabellen (vgl. 5.4.3). Sobald alle partiellen Bewertungen vorliegen, aggregiert der Diskussionsleiter diese zu einem Gesamtgüte-Wert.

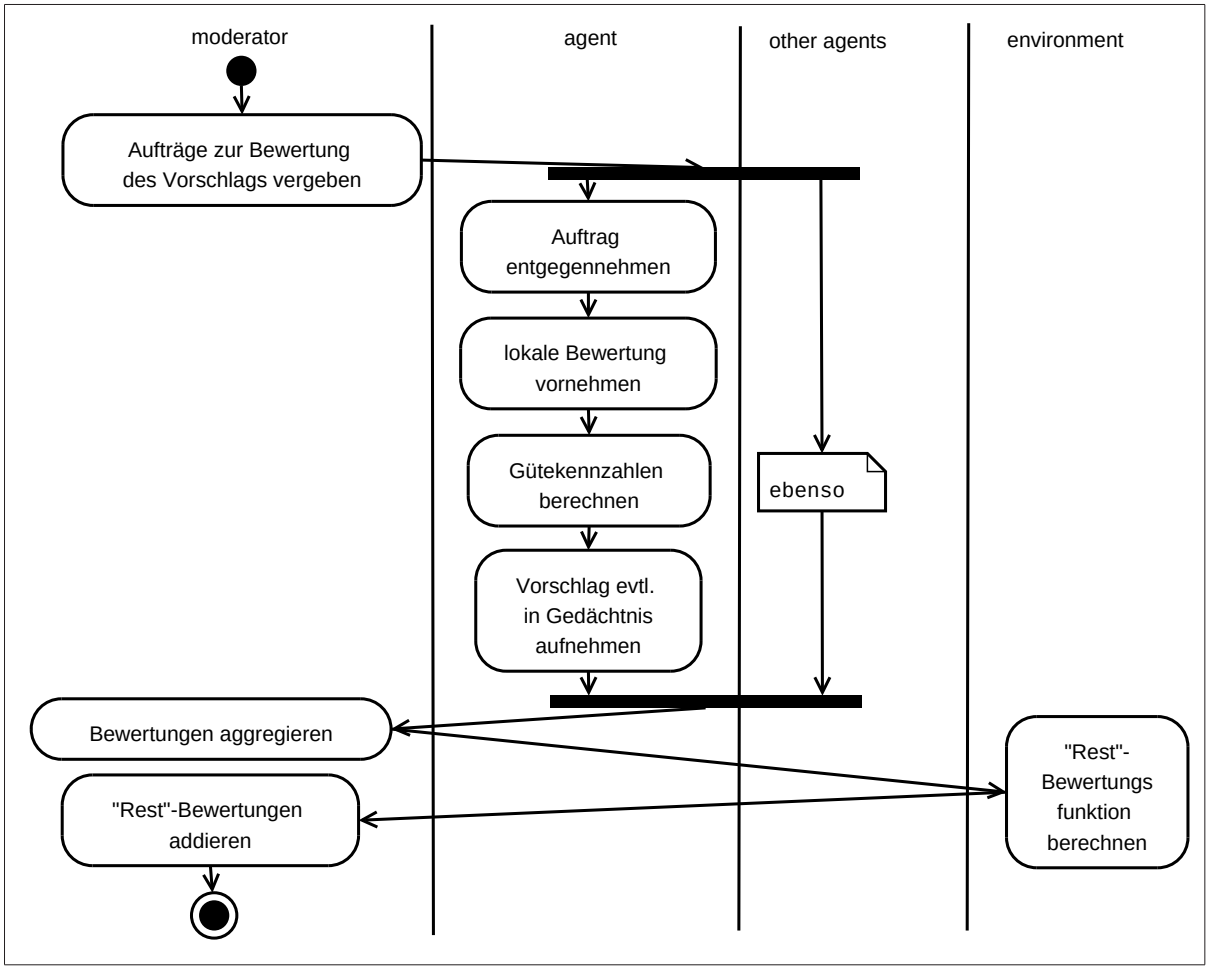


Abbildung 6.5: *Aktivitätsdiagramm für die Evaluierung eines Vorschlags*

Um auch die Bearbeitung von Problemen zu ermöglichen, bei denen eine Zerlegung der Zielfunktion in Teilfunktionen nicht vollständig möglich ist, wird im Anschluss eine „Rest"-Bewertungsfunktion einer problemspezifisch implementierten Klasse der ENV-Komponente aufgerufen, deren Ergebnis zusätzlich zum Gesamtgüte-Wert addiert wird.[5] Zuletzt übergibt der Moderator den Vorschlag zusammen mit dem Gesamtgüte-Wert an die *GA*-Komponente zur Eintragung in den Vorschlagspool. Abbildung 6.5 zeigt den Ablauf der Evaluierung eines Lösungsvorschlags.

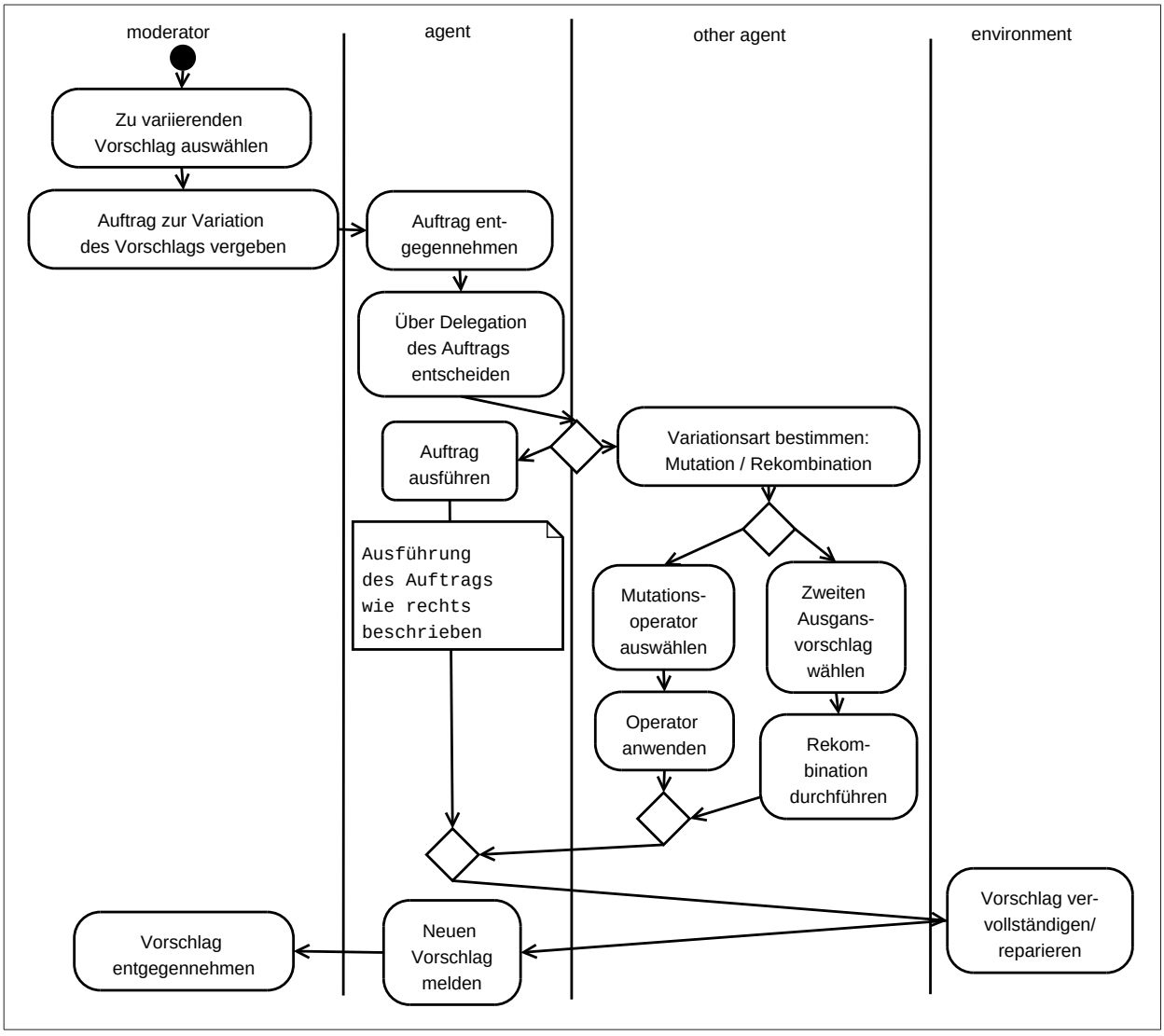


Abbildung 6.6: *Aktivitätsdiagramm für die Variation eines Vorschlags*

[5]Die Berechnung der Restfunktion für die Aggregation der partiellen Bewertungen zu einem Gesamtgüte-Wert wurde in die Umgebungs-Komponente ausgelagert, um die Methoden des Moderator-Objekts problemunabhängig zu halten.

Variation von Vorschlägen
Zur Variation eines Lösungsvorschlags wählt der Diskussionsleiter einen Ausgangsvorschlag und übergibt diesen an einen beliebigen Agenten. Dieser Agent entscheidet nun anhand des gespeicherten Zufriedenheitswertes, den er zuvor für den Vorschlag errechnet hat, ob er den Vorschlag selbst variieren oder den Variationsauftrag an einen anderen Agenten delegieren soll, der größeres Verbesserungspotential meldet. Der Agent, der die Variation schließlich durchführt, kopiert den Ausgangsvorschlag zunächst und entscheidet dann, ob eine Mutation oder eine Rekombination durchgeführt werden soll. Für eine Mutation wählt er anhand der berechneten Güte-Koeffizienten, die zuvor für den Ausgangsvorschlag berechnet wurden, einen passenden Operator aus und wendet diesen an. Für die Durchführung einer Rekombination muss zunächst ein zweiter Vorschlag ausgewählt werden. Dazu kann auf die entsprechende Schnittstelle der *GA*-Komponente zugegriffen werden. Zuletzt lässt der ursprünglich beauftragte Agent den neuen Vorschlag von der *ENV*-Komponente vervollständigen bzw. reparieren und meldet den Vorschlag an den Diskussionsleiter. Abbildung 6.6 zeigt ein Aktivitätsdiagramm für den Ablauf der Variation eines Lösungsvorschlags.

6.3 Weitere Programmpakete und die Einbettung in ein Gesamtsystem

Das im vorausgegangenen Abschnitt behandelte Paket *GC* beinhaltet die zentralen Komponenten des entwickelten Gesamtsystems zur Bearbeitung Kombinatorischer Optimierungsprobleme. Ein zweites Paket stellt die Benutzerschnittstelle (UI) dar, zu der auch die Klasse ConferenceMonitor gehört. Die Benutzerschnittstelle besteht aus den drei Komponenten MAIN, GUI und MONITOR, die in den folgenden Abschnitten besprochen werden. Ein weiteres „kleines Paket“ ergibt sich aus den vom Benutzer in problemspezifischer Weise zu implementierenden Klassen – der Name dieses Pakets wird sich an der jeweiligen Bezeichnung des zu bearbeitenden Problems orientieren. Neben den bereits erwähnten Klassen *Agent* und *ProblemEnvironment* muss die abstrakte Klasse *ConferenceMonitor* erweitert werden. Konkret muss hier eine Methode zur textuellen oder graphischen Darstellung eines Lösungsvorschlags implementiert werden. Eine detailliertere Beschreibung folgt in Abschnitt 6.3.3. Abbildung 6.7 zeigt die Pakete und ihre wichtigsten Komponenten im Überblick.

6.3.1 Die MAIN-Komponente des UI-Pakets

Über die Klassen der Komponente MAIN wird das Gesamtsystem im *Batchbetrieb* oder *mit Graphischer Benutzeroberfläche* gestartet.

Im Batchbetrieb wird ein Optimierungslauf direkt von den Klassen der MAIN-Komponente gestartet. Dazu muss der Name einer XML-Datei mit der Beschreibung des zu bearbeitenden Problemexemplars in der Kommandozeile übergeben werden.

Im GUI-Modus wird von Klassen der MAIN-Komponente kein Optimierungslauf gestartet, stattdessen wird eine Graphische Benutzeroberfläche initialisiert. Über diese Oberfläche können XML-Dateien mit Problembeschreibungen geladen werden. Die Graphische Benutzeroberfläche stellt außerdem unterschiedliche Methoden zur Darstellung des Verlaufs der Optimierung sowie der

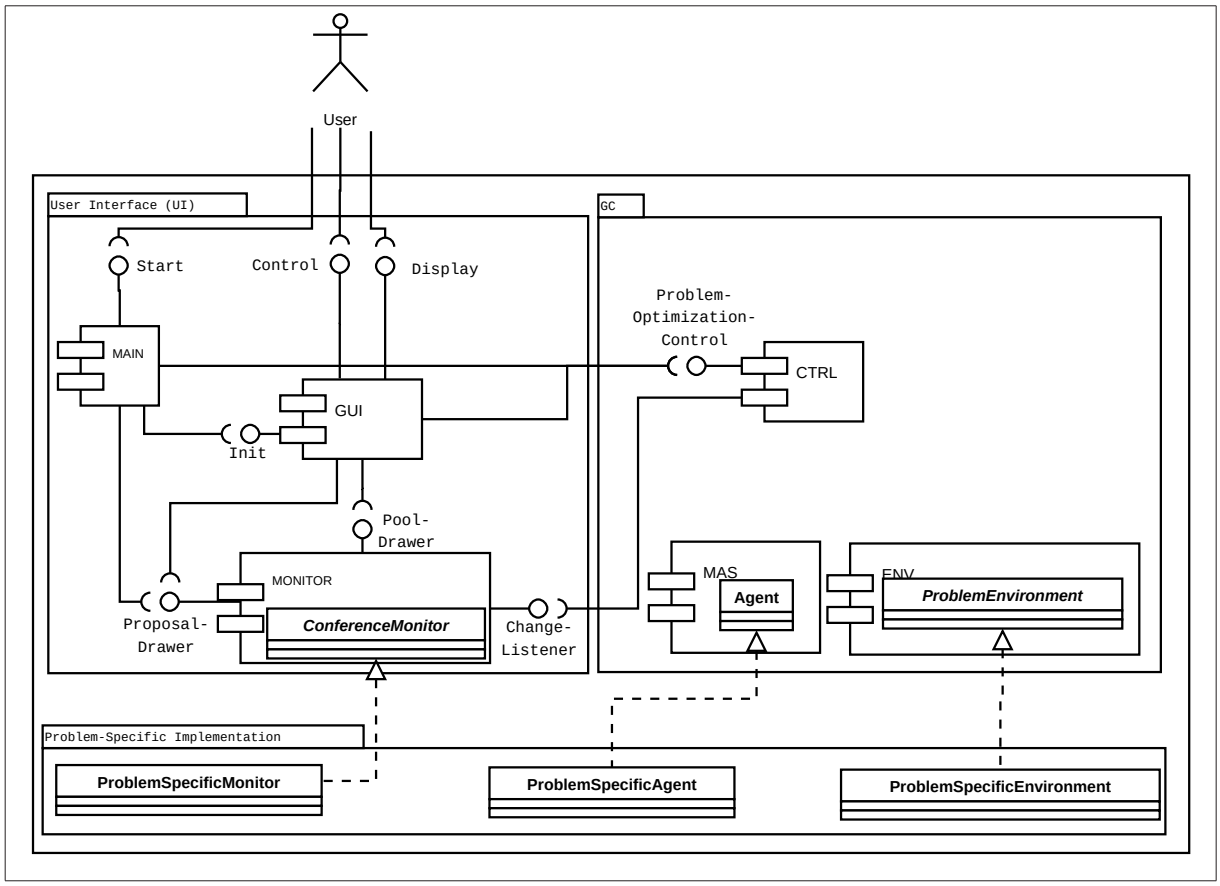


Abbildung 6.7: *Pakete und wichtigste Komponenten des Gesamtsystems*

Vorschläge im Vorschlagspool bereit und bietet dem Benutzer Möglichkeiten, den Ablauf der Optimierung zu steuern (z.B. Abbrechen und Pausieren der Optimierungslaufs). Mehrere (auch unterschiedliche) Problemexemplare können parallel geladen und Optimierungsläufen unterzogen werden.

6.3.2 Die MONITOR-Komponente des UI-Pakets

Je nachdem, in welchem Modus das Gesamtsystem gestartet wurde, greifen entweder Klassen der MAIN- oder der GUI-Komponente auf die Schnittstelle *ProposalDrawer* der MONITOR-Komponente zu, die eine Darstellung des momentan besten Vorschlags im Vorschlagspool bereitstellt. Diese Schnittstelle wird von der Konferenzmonitor-Klasse *(ConferenceMonitor)* implementiert, die jedoch abstrakt ist. Die entsprechende Darstellungs-Methode, die als Parameter einen darzustellenden Vorschlag sowie eine Darstellungsumgebung übergeben bekommt, muss vom Anwender überschrieben werden.
Wird diese Methode von der GUI-Komponente aufgerufen, so wird typischerweise eine Graphik-Umgebung mit übergeben, in die der Konferenzmonitor den Vorschlag „zeichnen“ soll. Beim Start ohne Graphische Benutzeroberfläche ruft die MAIN-Komponente dagegen die Methode

mit einem Ausgabestrom (z.B. Datei oder Standardausgabe) als Parameter auf.

Weitere Schnittstellen der MONITOR-Komponente, die ebenfalls von *ConferenceMonitor* implementiert werden, sind *PoolDrawer* und *PoolChangeListener*. Mittels letzterer informiert die Kontrollkomponente der Genetischen Konferenz darüber, dass ein Generationenübergang stattgefunden hat und eventuell neue Vorschläge zur Verfügung stehen, die angezeigt bzw. ausgegeben werden können. Die Fitness des jeweils besten Vorschlags einer Generation speichert der Konferenzmonitor in einem speziellen Array. *PoolDrawer* stellt die Methoden *drawPool* und *drawProgress* für die GUI-Komponete bereit. *drawPool* zeichnet Informationen über die Zusammensetzung des Vorschlagspool in eine übergebene Grafikumgebung. *drawProgress* zeichnet – ebenfalls in eine übergebene Graphikumgebung – eine Kurve, die die Güte des jeweils besten Vorschlags über die Generationen hinweg darstellt – dazu wird auf das erwähnte Array zurückgegriffen. Abbildung 6.8 zeigt die Interfaces und die Klasse der *MONITOR*-Komponente im Überblick.

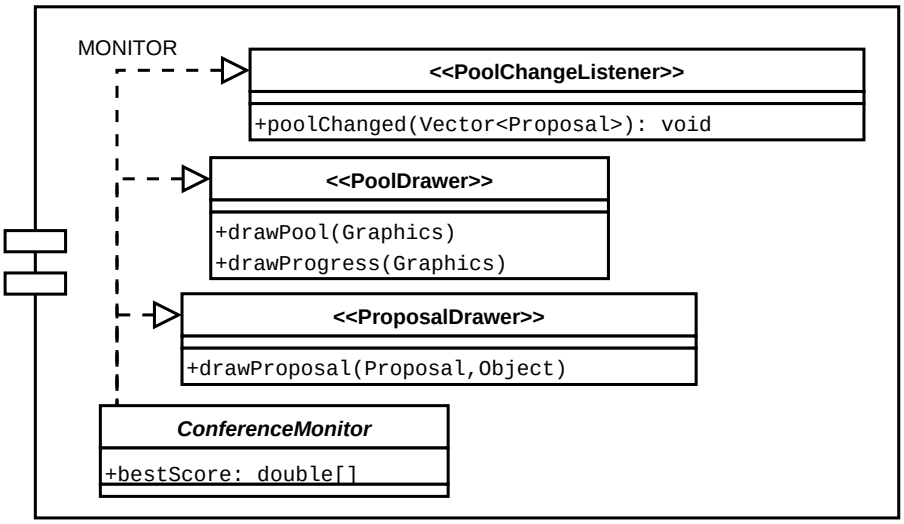


Abbildung 6.8: *Die Monitor-Komponente*

6.3.3 Die GUI-Komponente des UI-Pakets

Die graphische Benutzeroberfläche wurde mit dem *Java-Swing* Klassenpaket realisiert. Sie besteht beim Programmstart aus einem Hauptfenster mit Menüleiste und einem internen Fenster. Das interne Fenster zeigt die Güte der jeweils besten Vorschläge für unterschiedliche, parallel ablaufende Optimierungsvorgänge über die Generationen hinweg als Kurve – für diese Darstellung wird die besprochene Schnittstelle der MONITOR-Komponente angesprochen.

Statt einer detaillierten Beschreibung der *Klassen* der GUI-Komponente werden hier einige *Ansichten* und *Funktionen* beschrieben, die die GUI dem Anwender bietet. Über den Menüeintrag *File* → *Open* können XML-Files (vgl. Abschnitt 6.3.4) für konkrete Problembeispiele geladen werden. Wurde eine Datei ausgewählt, so wird ein internes Fenster geöffnet. Dieses Fenster verfügt über vier *Ansichten* (vgl. Abbildung 6.9), die vom *ConferenceMonitor*-Objekt erzeugt und über

die *ProposalDrawer*-Schnittstelle der MONITOR-Komponente bereitgestellt werden:

Lösungsvorschlags-Ansicht: In dieser Ansicht wird jeweils ein Lösungsvorschlag aus dem Pool graphisch dargestellt. Die Darstellung erfolgt problemspezifisch, d.h. die entsprechende Methode muss überschrieben werden.

Pool-Ansicht: Diese Ansicht bietet Informationen wie Güte, Alter, letzter modifizierender Agent sowie angewendeter Modifikationsplan für alle Lösungsvorschläge, die momentan im Pool enthalten sind.

Verlaufs-Ansicht: Diese Ansicht zeigt Kurven für den jeweils besten und den schlechtesten Lösungsvorschlag über die Generationen hinweg. Die Analyse der Entwicklung des Abstandes der beiden Kurven über die Generationen hinweg kann bei der Beurteilung des Konvergenzverhaltens des Algorithmus helfen.

SP-Matrix-Ansicht: Diese Darstellung zeigt die Situation-Plan Matrix mit den derzeitigen Einträgen (Wahrscheinlichkeiten). Die Ansicht ist insbesondere dann interessant, wenn sich die Wahrscheinlichkeiten über die Zeit hinweg ändern, weil der Lernmodus gewählt wurde.

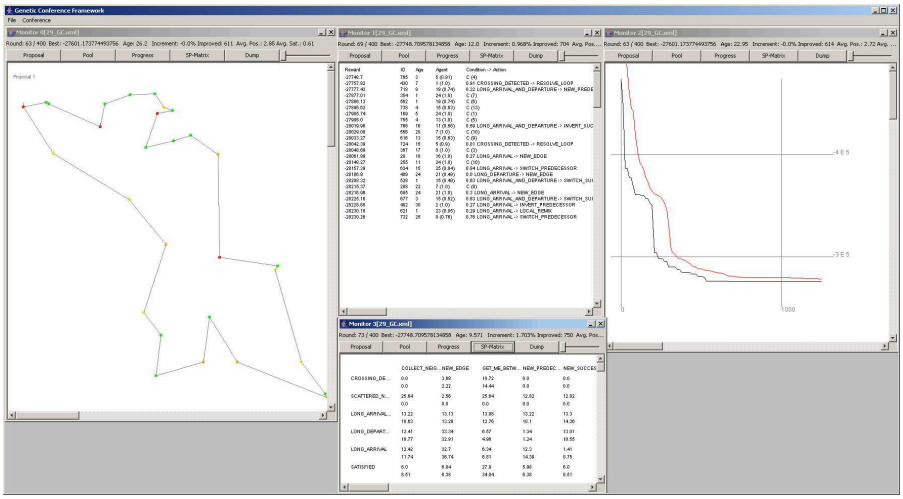


Abbildung 6.9: *Von einem ConferenceMonitor bereitgestellte Sichten: Lösungsvorschlags-Ansicht (links), Pool-Ansicht (mitte oben), Verlaufs-Ansicht (rechts) und SP-Matrix-Ansicht (mitte unten)*

Über weitere Menü-Einträge kann ein Optimierungslauf für alle momentan ausgewählten Problembeispiele gestartet und beendet werden. Außerdem ist ein Export der Ergebnisse eines Optimierungslaufs möglich.

6.3.4 Problemspezifikation und Belegung von Optimierungsparametern in XML-Files

In Abschnitt 6.2.2 wurde bereits erwähnt, dass die zur Bearbeitung eines konkreten Problemexemplars notwendige Problembeschreibung über eine XML-Datei spezifiziert werden muss.

Die XML-Datei enthält einerseits einige grundsätzliche Parameter für die Durchführung von Optimierungsläufen. Die wichtigsten seien hier zusammengestellt:

***MutationRate*:** Analog zum klassischen GA muss die Wahrscheinlichkeit für die Durchführung einer Mutation spezifiziert werden. Anders als beim klassischen GA erfolgt eine Mutation jedoch ohne vorherige Rekombination von Vorschlägen. Stattdessen wird über die Mutationsrate das Verhältnis von Rekombinationen und Mutationen festgelegt.

***MaxGeneration*:** Als Abbruchbedingung kann die maximale Generationenzahl bis zum Abbruch der Optimierung spezifiziert werden.

***DeleteSameProposals*:** Das gleichzeitige Auftreten identischer oder sehr ähnlicher Vorschläge im Lösungspool kann verboten werden, um die Vielfalt der Vorschläge zu erhöhen. Hier kann angegeben werden, wie viele Agenten zwei Vorschläge als unterschiedlich einschätzen müssen, damit beide Vorschläge erhalten bleiben (vgl. 5.4.4).

***PoolSize*:** Die Größe des Vorschlagspool (Vorschläge der aktuellen Generation) kann als Parameter spezifiziert werden.

***Elite*:** Die jeweils besten Vorschläge einer Generation können direkt in die nächste Generation übernommen werden, ohne mit neuen Vorschlägen konkurrieren zu müssen. Die Anzahl zu übernehmender Vorschläge wird mit diesem Parameter festgelegt.

***MemoryCrossoverRate*:** Mit diesem Parameter kann die Wahrscheinlichkeit angegeben werden, mit der bei der Durchführung einer Rekombination ein Vorschlag aus dem Gedächtnis eines Agenten herangezogen werden soll.

***CoalitionCrossoverRate*:** Analog wird mit diesem Parameter die Wahrscheinlichkeit spezifiziert, mit der bei der Durchführung einer Rekombination ein Vorschlag aus einer Koalition eines Agenten stammen soll.

Neben den Problemparametern sind andererseits noch weitere Spezifikationen zur Durchführung eines Optimierungslaufs notwendig:

Namen problemspezifisch implementierter Klassen Wie unter 6.2.2 beschrieben, müssen die Namen der problemspezifischen Implementierungen der abstrakten Klassen *Agent*, *ProblemEnvironment* und *ConferenceMonitor* angegeben werden.

Speicherort der SP-Matrix Der Pfad einer Datei, die die Situation-Plan-Matrix enthält, muss spezifiziert werden.

Lernrate Hier kann ein Faktor für die Stärke der Auswirkung des Feedbacks eines Agenten auf die SP-Matrix angegeben werden.

Die Umsetzung der Konzepte *Gedächtnis*, *Koalitionen*, *Auswahl der Verbesserungsstelle anhand lokalen Verbesserungspotentials* und *Situations-spezifische Auswahl des Variationsoperators* kann zu Vergleichszwecken außerdem über entsprechende Einträge in der XML-Datei „abgeschaltet“ werden. Die unterschiedlichen Modi für die Testläufe der Anpassung der Genetischen Konferenz für die Anwendungsbeispiele und für metrische TSP (Kapitel 7) entstehen durch An- und Abschalten dieser Optionen.

6.4 Die Java-Implementierung

Das Framework wurde in der Programmiersprache *Java* implementiert. *Java* bot sich zur Umsetzung des Modells aus mehreren Gründen an:

- Als *Objektorientierte Sprache* läßt *Java* eine einfache Umsetzung des beschriebenen objektorientierten Modells zu.
- Java bietet mit dem „*Dynamic Class Loading*“ die Möglichkeit, Klassen erst zur Laufzeit des Programms einzubinden. Dadurch wird die Einbindung problemspezifischer Klassen in das prinzipiell problemunabhängige Framework auf elegante Weise möglich (vgl. 6.2.2).
- Die Verwaltung mehrerer voneinander unabhängiger Prozesse *(Threads)*, die für die Implementierung der Agenten benötigt wird, ist in *Java* unproblematisch.
- Mit dem Swing-Paket stellt *Java* eine Bibliothek für die einfache Entwicklung ansprechender Benutzeroberflächen bereit und für eine spätere Erweiterung der Implementierung zur Ausführung auf verteilten Systemen steht das *RMI*-Paket für den Aufruf entfernter Methoden (*Remote Method Invocation*) zur Verfügung.
- Die möglicherweise zu erwartende schnellere Ausführungszeit von Programmkode, der in einer in anderen Sprachen entwickeltet wurde (z.B. C, C++ usw.) ging zunächst nicht in Implementierungsentscheidungen ein, hier sollte primär die Umsetzbarkeit des Modells an sich gezeigt werden. Die Implementierung ist auch insofern als Prototyp anzusehen.

Object-Recycling

Während des Ablaufs einer Optimierung werden sehr viele neue Vorschläge erzeugt. Die Konstruktion neuer Objekte ist einerseits zeitintensiv. Andererseits wird für jedes neu zu konstruierende Objekt Hauptspeicherplatz beansprucht. Ohne sorgfältige Verwaltung des Hauptspeicherplatzes besteht hier die Gefahr von Speicherüberläufen. Bei *Java* wird die Hauptspeicherverwaltung von der *Java virtual machine* übernommen, ein spezieller *Garbage Collector* befreit den Hauptspeicher von nicht mehr referenzierten Objekten.

Bei rechenintensiven Anwendungen mit vielen Konstruktoraufrufen kommt der Garbage Collector leider oft nicht mehr mit der Freisetzung von Hauptspeicher-Ressourcen nach. Unter anderem aus diesem Grund wird oft die Technik des so genannten Object-Recycling angewendet: Statt der Erzeugung neuer Objekte werden die Attribute nicht mehr benötigter Objekte gelöscht und neu gesetzt. So muss oft nur eine sehr geringe Zahl von Objekten tatsächlich *konstruiert* werden, bereits vorhandene Objekte kommen *immer wieder neu* zum Einsatz.

Bei der Implementierung des Framework wird Object-Recycling für die Vorschlags-Objekte verwendet. Eine spezielle (statische) Klasse *ProposalFactory* hält Referenzen auf alle Vorschläge, die nicht mehr im Vorschlagspool, den Gedächtnissen der Agenten oder in Koalitionen referenziert werden. Wenn ein Agent einen neuen Vorschlag in die Konferenz einbringen möchte ruft er eine spezielle Methode der ProposalFactory auf. Diese Methode liefert – sofern vorhanden – eine Referenz auf ein „altes“ Vorschlags-Objekt zurück (bei dem natürlich zuvor alle Inhalte auf den Ausgangszustand gesetzt werden). Nur wenn keine alten Vorschlags-Objekte zur Verfügung stehen, wird ein neues Objekt konstruiert.

Umfang der Implementierung

Der Quellcode des prototypisch implementierten Framework umfasst ca. 3000 Kodezeilen in 13 Dateien. Ergebnis einer Kompilierung ist Bytecode im Umfang von ca. 100 KB. Die problemspezifischen Erweiterungen für das TSP und die drei Anwendungsfälle (Kapitel 7) bestehen jeweils aus 3 bis 5 Klassen mit insgesamt jeweils ca. 1800 Kodezeilen (ca. 40 KB kompilierter Bytecode).

Kapitel 7

Problemspezifische Anpassung und Anwendung

Dieses Kapitel beschreibt die Anpassung und Anwendung des entwickelten Framework auf die in Kapitel 3 eingeführten Anwendungsbeispiele.

Im ersten Abschnitt wird ein Vorgehensmodell für problemspezifische Implementierungen des Framework auf konkrete Optimierungsprobleme beschrieben (Abschnitt 7.1). Anhand einer exemplarischen Anpassung für metrische TSP werden die zur problemspezifischen Implementierung notwendigen Schritte erläutert.

Die darauf folgenden Abschnitte 7.2 bis 7.4 enthalten eine Beschreibung der Adaption der Genetischen Konferenz anhand dieses Vorgehensmodells auf die Anwendungsbeispiele.

7.1 Vorgehensmodell am Beispiel einer Anpassung für metrische TSP

Drei Klassen des Framework wurden abstrakt gehalten und müssen problemspezifisch implementiert werden. Daraus resultieren die wesentlichen Schritte bei der Anpassung: Die Implementierung der Agenten und die Spezifikation ihres Verhaltens (1), die Implementierung der Umgebung (2) und die Implementierung eines Konferenzmonitors (3). Diese Schritte werden im Folgenden erläutert und an einer exemplarischen Anpassung für metrische TSP verdeutlicht.

Schritt 1: Implementierung der Agenten und Spezifikation ihres Verhaltens

1.1: Die Identifikation der Agenten:
Bei kombinatorischen Optimierungsproblemen sucht man nach der bestmöglichen Anordnung, Zuordnung, Gruppierung oder Auswahl einer Menge diskreter Objekte. Die Entscheidungsvariablen der Zielfunktion lassen sich bei einer intuitiven Lösungsrepräsentation und Kodierung[1] diesen Objekten zuordnen. Jeder Agent soll die Konfiguration eines Objekts verwalten. Der erste Schritt bei der Adaption des Framework auf das Optimierungsproblem besteht also in der Analyse des Problems und der Fitness-Funktion mit dem Ziel, die Objekte zu identifizieren, die jeweils von einem Agenten repräsentiert werden sollen.

Es soll später möglich sein, sinnvolle Güte-Koeffizienten für die Belegungen der Entscheidungsvariablen zu definieren, die jeweils einem Agenten zugeordnet sind. Dazu muss die Identifikation der Agenten natürlich derart erfolgen, dass die Berechnung solcher Güte-Koeffizienten auf Grundlage der Belegung der einem Agenten zugeordneten Variablen und in Abhängigkeit *möglichst weniger* zusätzlicher Entscheidungsvariablen überhaupt möglich ist (vgl. Abschnitt 5.4.1 und 5.4.3).

Beispiel Anpassung an metrische TSP:
Bei einer Anpassung der Genetischen Konferenz zur Bearbeitung von TSP bietet es sich an, die Agenten jeweils eine *Stadt* repräsentieren zu lassen. Es wird die in Abschnitt 2.3 vorgestellte Kodierung der Lösungsvorschläge gewählt. Jedem Agenten wird so genau eine Entscheidungsvariable zugewiesen, da jeder Stadt s_1 bis s_n für die vollständige Spezifikation einer Rundreise nur eine Positionsnummer $pos(s_i)$ zugeordnet werden muss. Es wird also nur ein Satz von Entscheidungsvariablen benötigt, d.h. eine Variable pro Agent. Damit gültige Rundreisen entstehen, darf jede Positionsnummer natürlich nur einmal vergeben werden.

1.2: Die Speicherung von Zwischenergebnissen der lokalen Bewertung (*calcIntermediateEvalResults*)
Bei partiell bewertbaren KOP kann die Zielfunktion meist zum großen Teil als Summe von Teilfunktionen aufgefasst werden, die jeweils einem Einzelobjekt zugeordnet werden können. Wenn bei der Evaluation dieser Teilfunktionen Zwischenergebnisse anfallen, die auch für die Berechnung von Güte-Koeffizienten verwendet werden können, sollten diese Zwischenergebnisse nur einmal berechnet und dann zwischengespeichert werden[2]. Dafür steht die Funktion *calcIntermediateE-*

[1] vgl. Abschnitt 3.1.3

[2] Detailliertere Erläuterungen und Beispiele zur lokalen Evaluierung von Lösungsvorschlägen finden sich in Abschnitt 5.4.3

valResults bereit. Die Funktion muss ein beliebig langes Feld (*Array*) von Gleitkommawerten zurückliefern.

Beispiel Anpassung an metrische TSP:
Offensichtlich kann die Zielfunktion eines TSP sehr einfach in Teilfunktionen zerlegt werden, indem jede Teilfunktion die Entfernung der „Anbindung“ einer einzelnen Stadt misst. Sei $dist(s_i, s_j)$ die Entfernungsfunktion für die Messung der Distanz zwischen zwei Städten s_i und s_j. Für jede Stadt s_i des Problembeispiels und einen Lösungsvorschlag p, bei dem die Stadt s_i zwischen den Städten $pred(s_i)$ und $succ(s_i)$ besucht wird, berechnet sich dann die partielle Evaluierungsfunktion y_i als

$$y_i = \frac{1}{2}(dist(pred(s_i), s_i) + dist(succ(s_i), s_i))$$

Der Faktor 1/2 wird hinzugenommen, da bei Aufsummierung der partiellen Evaluierungsfunktionen jede Strecke in der Rundreise doppelt gezählt wird. Da die Entfernungen zu Vorgänger- und Nachfolgerstadt auch für die Berechnung der Güte-Koeffizienten verwendet werden können, wird die Methode *calcIntermediateEvalResults* so implementiert, dass sie ein Feld *(Array)* bestehend aus diesen beiden Entfernungen zurückliefert.
Überschrieben wird außerdem die partielle Evaluierungsfunktion *getLocalEvaluationValue*, um den erwähnten Faktor 1/2 aufzunehmen.

1.3: Die Berechnung von Güte-Koeffizienten (*calculateRatios*)
Die Evaluations-Zwischenergebnisse werden automatisch zur Berechnung der Teilfunktionen herangezogen und in spezielle Tabellen abgelegt, damit bei der Berechnung der Güte-Koeffizienten in der Funktion *calculateRatios* auf sie zurück gegriffen werden kann. Die Ermittlung der Situation und damit auch die Auswahl von Variationsoperatoren erfolgt nach Maßgabe des Anwenders in Abhängigkeit der Koeffizienten. Außerdem wird die lokale Zufriedenheit eines Agenten mit einem Lösungsvorschlag auf Basis der Koeffizienten berechnet: Die Zufriedenheit, die auch zur Abschätzung des lokalen Verbesserungspotentials herangezogen wird, ergibt sich zunächst automatisch durch das Produkt der Koeffizienten.[3] Die Güte-Koeffizienten müssen daher zwischen null und eins skaliert sein.

Beispiel Anpassung an metrische TSP:
Die Methode *calculateRatios* kann beispielsweise so implementiert werden, dass sie für einen Lösungsvorschlag eines konkreten Problembeispiels drei Güte-Koeffizienten zurückliefert: Einen Koeffizient für den Anreiseweg, d.h. die Entfernung zur Vorgängerstadt im Verhältnis zur Entfernung der nächstliegenden Städte, einen Koeffizient für den Abreiseweg (Bedeutung analog) und einen Koeffizient für die Länge des Umwegs, der durch den Besuch der Stadt verursacht wird. Die ersten beiden Koeffizienten ergeben sich aus dem Verhältnis von der Entfernung der vom Agenten repräsentierten Stadt s_a zur Vorgängerstadt $pred(s_a)$ (bzw. Nachfolgerstadt $succ(s_a)$) und der Entfernung zu den jeweils nächstliegenden Städten des konkreten Problemexemplars. Der dritte Koeffizient errechnet sich aus dem Verhältnis von der Summe aus An- und Abreiseweg zur Entfernung zwischen Vorgänger- und Nachfolgerstadt.

[3] Für eine andere Art der Berechnung der lokalen Zufriedenheit aus den Güte-Koeffizienten muss die Methode *getAggregatedSatisfaction* überschrieben werden.

1.4: Die Initialisierung von Startlösungen (*createNewProposal*)
Der Anwender muss die Methode *createNewProposal* implementieren, mit der die Agenten neue Vorschläge initialisieren können. Hier können zufallsbasierte Verfahren oder problemspezifische Konstruktionsheuristiken zum Einsatz kommen. Um eine gewisse Variationsbreite im Vorschlagspool zu gewährleisten und so frühzeitige Konvergenz zu vermeiden, sollte darauf geachtet werden, dass die einzelnen Agentenexemplare möglichst unterschiedliche (beispielsweise jeweils „für sie“ besonders günstige) Lösungsvorschläge liefern.

Beispiel Anpassung an metrische TSP:
Für die Initialisierung von Startlösungen kommt beispielsweise eine *„NearestNeighbour“* - Heuristik in Frage: Ein Agent erzeugt einen neuen Vorschlag, indem er für „sich selbst“ eine beliebige Positionsnummer einträgt und für die beiden Agenten, die die am nähesten liegenden Städte repräsentieren, die „benachbarten“ Positionsnummern. Im Anschluss vergibt er die übrigen Positionen, indem er die Teilreise an den Enden jeweils durch eine möglichst nahe liegende, noch nicht besuchte Stadt erweitert. Sobald für jeden Agent eine Position eingetragen ist, ist die Rundreise fertig „konstruiert“ und kann in den Vorschlagspool eingebracht werden.

1.5: Die Integration lokaler Verbesserungsverfahren, Spezifikation von Situationen und Plänen *(SA-Matrix, getStatus, triggerPlan)*
Mit den Zufriedenheitsfunktionen der Agenten kann bereits ermittelt werden, *an welcher Stelle* ein Vorschlag lokales Verbesserungspotential bietet. Der Vorschlag wird automatisch einem Agenten zur Variation überlassen, der hohes Verbesserungspotential meldet. Mit der Methode *getStatus* legt der Anwender fest, in welcher Situation sich ein Agent bezüglich eines Lösungsvorschlags befindet. Dazu kann auf die zuvor berechneten und automatisch zwischengespeicherten Güte-Koeffizienten zurückgegriffen werden.

Beispiel Anpassung an metrische TSP:
Die Methode *getStatus* könnte so implementiert werden, dass sie je nach Ausprägung der Güte-Koeffizienten eine der in Tabelle 7.1 beschriebenen Situationen zurückliefert.

Anzahl und Bezeichnungen der Situationen (und der Pläne) sind vom Anwender frei wählbar. In einer Situations-Plan-Matrix *(SP-Matrix)* spezifiziert der Anwender die Wahrscheinlichkeiten für die Durchführung bestimmter Pläne in den jeweiligen Situationen. Anhand der Wahrscheinlichkeiten werden die auszuführenden Pläne dann stochastisch ausgewählt.

Für die Variation von Vorschlägen kann der Anwender beliebige problemspezifische Variationsoperatoren in das Optimierungsverfahren integrieren. Über die Funktion *triggerPlan* legt er fest, welche Operatoren bei der Durchführung welches Plans zur Anwendung kommen sollen.

Beispiel Anpassung an metrische TSP:
Für die lokale Verbesserung von Lösungsvorschlägen können beispielsweise die in Tabelle 7.2 beschriebenen Pläne herangezogen werden. Als Situations-Plan-Matrix *(SP-Matrix)* kann beispielsweise die Tabelle 7.3 verwendet werden. Mit den dort eingetragenen Wahrscheinlichkeitswerten konnten in Testläufen (vgl. Kapitel 8) besonders gute Resultate erzielt werden.

Situation	Beschreibung
Crossing-Detected	Die Rundreise weist eine Kreuzung auf. Die Strecke zwischen der vom Agent repräsentierten Stadt und der als nächstes zu besuchenden Stadt kreuzt sich mit einem anderen Teil der Rundreise. Die Rundreise kann in diesem Fall immer verbessert werden, indem diese „Kreuzung“ aufgelöst wird, da die zu behandelnden Problembeispiele metrisch sind. Zur Erkennung einer Kreuzung kann jedoch keine der Güte-Koeffizienten herangezogen werden.
LDist-PredSucc	Die repräsentierte Stadt liegt in der Rundreise zwischen zwei Städten, die *beide* im Vergleich zu anderen Städten weit entfernt sind.
LDistSucc bzw. LDist-Pred	Die Stadt, die als nächstes besucht wird (bzw. zuvor besucht wurde), ist im Vergleich zu anderen Städten weit entfernt, während die andere „Nachbarstadt“ in der Rundreise relativ nahe ist.
LongDetour	Der Umweg durch den Besuch der repräsentierten Stadt ist vergleichsweise lang.
Satisfied	Der Agent ist mit dem Lösungsvorschlag (lokal) zufrieden.

Tabelle 7.1: *Situationen eines TSP-Agenten*

Plan	Beschreibung
GoBetween-Neighbours	Die Stadt wird zwischen zwei in der Nähe liegende, momentan in der Rundreise nacheinander besuchte Städte eingefügt.
NewPredecessor, NewSuccessor	Eine Stadt, die in der Nähe liegt, wird als neue Vorgängerstadt bzw. Nachfolgerstadt in der Rundreise eingetragen. Dazu wird die Positionsnummer dieser Stadt neu gesetzt, die Positionsnummern der übrigen Städte werden entsprechend verschoben.
Resolve-Loop	Eine erkannte Schleife wird aufgelöst.
NewEdge	Eine Kante zwischen der vom Agenten verwalteten Stadt und einer weiteren Stadt wird eingefügt. Die Städte, die in der Rundreise zwischen der verwalteten und der neuen Stadt lagen, werden sukzessive zwischen jeweils möglichst nahe liegende andere Städtepaare eingefügt.
RandomMove	Der Agent setzt die Position der repräsentierten Stadt in der Rundreise auf eine beliebige neue Postion, die anderen Positionsnummern werden entsprechend verschoben.
Random-Exchange	Der Agent tauscht die Positionsnummer „seiner“ Stadt mit einer beliebigen anderen aus.

Tabelle 7.2: *Pläne eines TSP-Agenten*

1.6: Die Rekombination von Vorschlägen (*recombineProposals*)
Für die Variation von Vorschlägen durch Rekombination wird die Methode *recombineProposals* verwendet, die ebenfalls vom Anwender implementiert werden muss. Als Parameter wird ein Vorschlag übergeben, bei dem der Agent eher großes lokales Verbesserungspotential berechnet hat sowie ein Vorschlag, mit dem er lokal zufrieden ist. Ziel ist, durch Kombination der Belegungen der beiden Vorschläge einen neuen Vorschlag zu erzeugen, bei dem das lokale Verbesserungspotential möglichst ausgeschöpft wird.

	Go-Between-Neighbours	New-Predecessor	New-Successor	Resolve-Loop	New-Edge	Random-Move	Random-Exchange
CrossingDetected	0	0	0	0.9	0	0.05	0.05
LDistPredSucc	0.8	0.05	0.05	0	0	0.05	0.05
LDistSucc	0.05	0	0.55	0	0.3	0.05	0.05
LDistPred	0.05	0.55	0	0	0.3	0.05	0.05
LongDetour	0.5	0.15	0.15	0	0.1	0.05	0.05
Satisfied	0.3	0.3	0.3	0	0	0.05	0.05

Tabelle 7.3: *Situation-Plan-Matrix für das TSP*

Beispiel Anpassung an metrische TSP:
Der erste zu rekombinierende Vorschlag entspricht einer Rundreise, die insgesamt relativ kurz ist, bei der jedoch in der „Umgebung“ der durch den Agenten repräsentierten Stadt relativ lange Strecken zurückgelegt werden. Beim zweiten Vorschlag ist diese Stadt günstiger angebunden, d.h. die Entfernungen zu direktem Vorgänger und Nachfolger sind gering.
Stammt der zweite Vorschlag, mit dem der Agent lokal zufrieden war, aus einer Koalition, so werden zunächst die Belegungen aller Koalitionsmitglieder entsprechend dieses Vorschlags übernommen. Aufgrund der Auswahl der Koalitionsmitglieder muss so eine zusammenhängende Teilreise entstehen. Handelt es sich beim zweiten Vorschlag nicht um die „Basis“ einer Koalition, so wird zufällig eine Teilreise extrahiert, an der der Agent beteiligt ist. Die Teilreise wird dann sukzessive an den Enden erweitert, indem noch nicht enthaltene Städte entsprechend der Reihenfolge in der Rundreise angehängt werden, die durch den ersten zur Rekombination übergebenen Vorschlag kodiert ist.[a]

[a] Dieses Rekombinationsschema entspricht dem von Davis [Dav85] vorgeschlagenen *Order Crossover (OX)*.

1.7: Kandidatenauswahl zur Bildung einer Koalition (*getPossibleCoalitionMembers*)
Um festzulegen, welche Agenten eine Koalition eingehen können, muss der Anwender die Methode *getPossibleCoalitionMembers* überschreiben, die jeweils für einen Agenten eine Menge von Kandidaten für die Bildung einer Koalition zurück liefern soll.

Bei Problemen mit Raumbezug bietet es sich an, die Distanzen zwischen den Objekten, die die Agenten repräsentieren, bei der Auswahl einzubeziehen. So kann die gemeinsame Vererbung der Belegungen stark interagierender Entscheidungsvariablen begünstigt werden.

Beispiel Anpassung an metrische TSP:
Für die Bildung einer Koalition baut die problemspezifisch implementierte Methode *getPossibleCoalitionMembers* auf folgende Weise eine Kandidatenmenge auf:
Zunächst werden die durch den Lösungsvorschlag spezifizierten direkten Nachbarn in der Rundreise in die Kandidatenmenge aufgenommen, wenn Ihre lokale Zufriedenheit über einem Schwellenwert liegt. Der Schwellenwert kann beispielsweise durch eine zufällig zwischen 0 und 1 gewählte Zufallszahl erzeugt werden. Sukzessive werden so weitere Agenten in die Kandidatenmenge aufgenommen, wenn sie Nachbarn von ausgewählten Kandidaten sind und ihre Zufriedenheit ebenfalls über einem (neu erzeugten) Schwellenwert liegt.

Schritt 2: Die Implementierung der Umgebung

2.1: Vervollständigung und Reparatur von Vorschlägen (*completeAndRepairProposal*)
Die Methode *completeAndRepairProposal* kann implementiert werden, wenn nach der Initialisierung oder Variation eines Vorschlags Operationen zu dessen Vervollständigung oder Reparatur durchgeführt werden sollen.

Beispiel Anpassung an metrische TSP:
Für die Optimierung von Rundreisen wird die Vervollständigungs- und Reparaturfunktion *completeAndRepairProposal* zwar nicht benötigt, da die Operatoren nur gültige Rundreisen zurückliefern. Möglich wäre jedoch beispielsweise die Implementierung einer lokalen Suchheuristik (z.B. einer k-change-Heuristik), um Lösungsvorschläge im Sinne eines *Memetischen Algorithmus* (vgl. Abschnitt 4.8.2) in lokale Optima bezüglich einer bestimmten Nachbarschaft umzuwandeln.

2.2: Implementierung der Restfunktion für die Bewertung von Lösungsvorschlägen (*remainderEvaluation*)
Die Umgebung stellt außerdem eine Schnittstelle für eine zusätzliche partielle Evaluierungsfunktion bereit, die genutzt werden kann, wenn die Zielfunktion des Optimierungsproblems nicht komplett in Teilfunktionen dekomponierbar war. In diesem Fall muss die Methode *remainderEvaluation* implementiert werden.

Beispiel Anpassung an metrische TSP:
Eine Restfunktion zur Bewertung von Lösungsvorschlägen wird hier ebenfalls nicht benötigt, da die Länge einer Rundreise komplett aus den Zwischenergebnissen der einzelnen Agenten bestimmt werden kann. Die Methode *remainderEvaluation* kann daher so implementiert werden, dass sie immer den Wert Null zurückliefert.

Schritt 3: Implementierung des Konferenzmonitors

Zuletzt sollte der Anwender einen problemspezifischen Konferenzmonitor implementieren, um später den Ablauf von Optimierungsläufen überwachen und Ergebnisse der Optimierung ausgeben zu können.[4]

3.1: Graphische Darstellung von Lösungsvorschlägen (*drawProposal*)
Zur graphischen Darstellung von Lösungsvorschlägen muss die Methode *drawProposal* implementiert werden, die einen Lösungsvorschlag in einer übergebenen Graphik-Umgebung (ein Objekt vom Typ *java.awt.Graphics*) darstellt. Für die Darstellung des Lösungsvorschlags steht ein Fenster mit 1280x1024 Pixeln und weißem Hintergrund zur Verfügung.
Auf die graphische Darstellung von Lösungsvorschlägen kann auch verzichtet werden, wenn das Framework nur ohne graphische Benutzeroberfläche gestartet werden soll.

[4]Da die beiden Methoden *dumpProposal* und *drawProposal* vom System im Verlauf der Optimierung in regelmässigen Abständen immer wieder aufgerufen werden, sollte darauf geachtet werden, dass deren Ausführung nicht zu zeitaufwändig ist.

Beispiel Anpassung an metrische TSP:
Mit der Methode *drawProposal* wird eine Rundreise graphisch dargestellt. Die Städte werden anhand der Koordinaten in das Zeichenfenster eingepasst und als Punkte gezeichnet. Die Punkte werden zusätzlich je nach Zufriedenheitsgrad des entsprechenden Agenten unterschiedlich eingefärbt. Abbildung 7.1 zeigt die graphische Darstellung von Rundreisen im Konferenzmonitor.

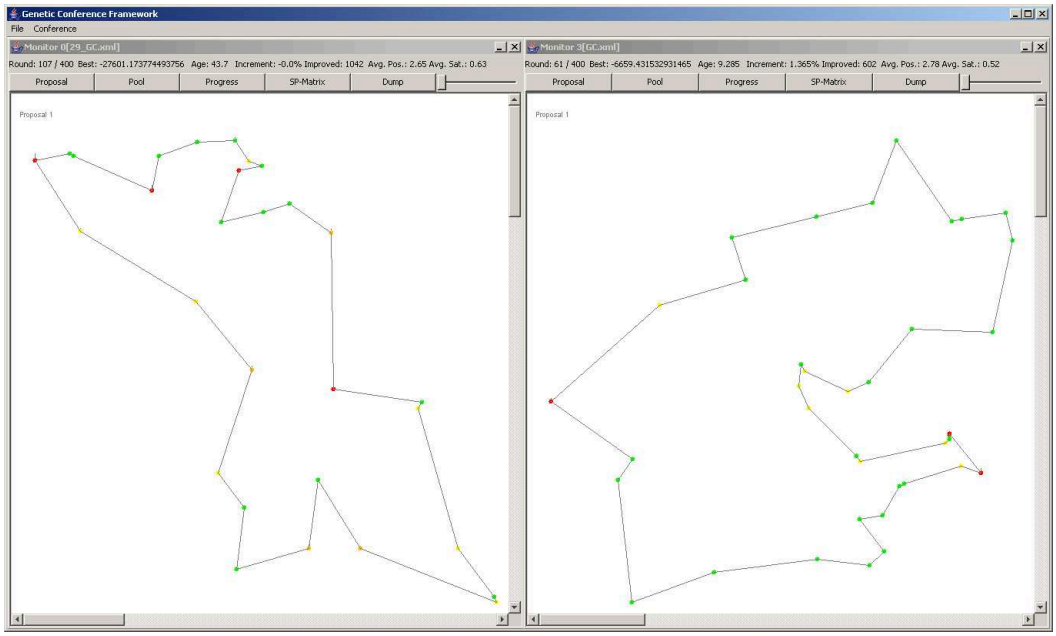


Abbildung 7.1: *Konferenzmonitor mit Darstellung von Lösungsvorschlägen für das TSP*

3.2: Textuelle Ausgabe von Lösungsvorschlägen: Implementierung der Methode *dumpProposal*

Die Methode *dumpProposal* soll die Beschreibung eines Lösungsvorschlags als Zeichenkette zurückliefern. Sie kann während des Optimiervorgangs für die textuelle Ausgabe eines Lösungsvorschlags genutzt werden, wenn die Genetische Konferenz beispielsweise ohne graphische Benutzeroberfläche gestartet wurde. Wird in der XML-Konfigurationsdatei eine Zieldatei für Zwischenergebnisse spezifiziert, so wird die Methode nach jeder Generation für den jeweils besten Lösungsvorschlag aufgerufen und der zurückgelieferte Text in diese Zieldatei geschrieben.

Beispiel Anpassung an metrische TSP:
Zur textuellen Ausgabe von Lösungsvorschlägen liefert die Methode *dumpProposal* eine Zeichenkette bestehend aus den Indizes der in der Rundreise aufeinander folgenden Städte (mit Leerzeichen getrennt) zurück.

7.2 Anwendungsbeispiel I: Anpassung auf das Problem der Standortplanung

Als erster Anwendungsfall für die Anpassung und Anwendung des Framework wird das Problem der Standortplanung (vgl. 3.2) herangezogen: Ziel ist die Transport-, Fix- und Strafkosten-minimierende Planung von Umschlagpunkten – im Folgenden mit *UP* abgekürzt – einer zweistufigen Distributionslogistik. Dabei *können* potenzielle Standorte vorgegeben sein und neben den Koordinaten *kann* auch die Anzahl zu planender Standorte Gegenstand der Optimierung sein.

Zunächst werden einige Vorüberlegungen zur Anpassung des Framework angestellt und die Parameter und Entscheidungsvariablen unterschiedlicher Problemexemplare identifiziert, die mit dem angepassten Framework bearbeitet werden können (7.2.1). Es folgt die Spezifikation der Zielfunktion zu bearbeitender Problemexemplare (7.2.2). Dann wird die Anpassung des Framework anhand des Vorgehensmodells beschrieben (7.2.3). Im Anschluss werden die Ergebnisse des angepassten Framework im Einsatz beschrieben (7.2.4) und mögliche Erweiterungen aufgezeigt (7.2.5). Das Verfahren wird mit anderen bekannten Optimierungsverfahren für Standortplanungsprobleme verglichen (7.2.6). Zuletzt findet sich ein Performanzvergleich mit Simulationen eines MAS und eines GA (7.2.7).

7.2.1 Vorüberlegungen zur Anpassung des Framework

Kodierung der Entscheidungsvariablen, räumliche Autokorrelation und Epistasie
Bei Standortplanungsproblemen bieten sich je nach Problemform binäre oder reellwertige Kodierungen der Entscheidungsvariablen an.

Handelt es sich um diskrete Probleme mit fest vorgegebenen potenziellen Standorten, so kann eine binäre Kodierung verwendet werden: Jedem potenziellen Standort pot_i wird eine Binärvariable b_i zugeordnet, die anzeigt, ob am entsprechenden Standort ein UP geplant werden soll ($b_i = 1$) oder nicht ($b_i = 0$). Die Variation einzelner Binärvariablen ($b_i = 1 - b_i$) entspricht der Hinzunahme oder der Wegnahme eines UP. Mit dieser Kodierung können auch Probleme mit variabler Zahl zu planender UP bearbeitet werden. Bei kleinen Änderungen eines Lösungsvorschlags sind auch nur relativ geringe Änderungen der Lösungsgüte zu erwarten, sofern die Anzahl der zu planenden UP hoch und die Unterschiede der Fixkosten gering sind. Es ist also von einer relativ hohen räumlichen Autokorrelation der Fitnesslandschaft auszugehen (vgl. A.5.1). Die Entscheidungsvariablen haben außerdem festen Ortsbezug, und dieser Ortsbezug bestimmt die Höhe des wechselseitigen Einflusses der Variablen auf die Zielfunktion: Gerade bei einer größeren Zahl potenzieller Standorte und zu planender UP sind die Auswirkungen der Belegungen einzelner Entscheidungsvariablen hauptsächlich von den Belegungen „benachbarter“ Entscheidungsvariablen abhängig, d.h. denen, die nahe gelegene andere potenzielle Standorte repräsentieren. Eine binäre Kodierung würde sich daher für die Bearbeitung mit der Genetischen Konferenz anbieten, die Agenten würden dann jeweils einen der *potenziellen* Standorte repräsentieren.

Zwei Gründe sprechen jedoch gegen eine solche Modellierung bei der Anpassung des Framework: Einerseits wäre die Anzahl der Agenten möglicherweise sehr hoch. Diese Tatsache könnte sich negativ auf die Performanz des Verfahrens auswirken – ein eher technisches Problem. Andererseits sollen mit dem angepassten Framework auch Problemexemplare *ohne* vorgegebene potenzielle Standorte bearbeitet werden können.

Es muss daher auf *reellwertige* Entscheidungsvariablen zurückgegriffen werden, die jeweils die Koordinaten der UP festlegen, wobei die Anzahl zu planender UP dabei variabel bleiben kann. Ein Ansatz, bei dem die Zahl der Agenten (die dann die UP repräsentieren) variabel bleibt, ist mit der Genetischen Konferenz jedoch technisch schwer realisierbar. Stattdessen wird zunächst von einer *maximal möglichen* Zahl von UP ausgegangen und ein weiterer *Satz* von binären Entscheidungsvariablen verwendet, die für jeden UP zusätzlich entscheidet, ob er auch tatsächlich in die Planung aufgenommen werden soll.

Auch bei dieser gemischt reell-binären Kodierung ist von relativ hoher räumlicher Autokorrelation der Fitnesslandschaft auszugehen, sofern sich die Distanz der Planungsorte eines Standorts bei zwei Lösungen auch in der Distanz der entsprechenden Punkte in der Fitnesslandschaft ausdrückt: Geringfügige Verschiebungen eines UP haben nur geringen Einfluss auf die Lösungsgüte. Der wechselseitige Einfluss der Variablen kann scheinbar wie bei der binären Kodierung räumlich differenziert werden: Die Güte einer Standortentscheidung ergibt sich vor allem aus den Standorten *umliegender* UP, da diese die Zuordnung potenzieller Kunden des UP beeinflussen. Die Lage weiter entfernter UP ist dagegen in diesem Zusammenhang weniger relevant. Da jedoch die räumliche Ausprägung (=Koordinaten) gerade die *Belegung* der Entscheidungsvariablen darstellt, kann eine solche Differenzierung nur für *konkrete Lösungsvorschläge*, nicht aber *für die Variablen an sich* – die nun ohne konkrete Belegung keinerlei festen Ortsbezug haben – vorgenommen werden. Es kann also nicht mehr *übergreifend für alle Lösungsvorschläge* entschieden werden, welche Entscheidungsvariablen aufgrund ihrer „räumlichen Nähe“ vermutlich starke (bzw. geringe) wechselseitige Einflüsse auf die Zielfunktion haben und daher bei der Rekombination gemeinsam (bzw. getrennt voneinander) vererbt werden sollten. Diese Entscheidung kann stattdessen nur für *konkrete Lösungsvorschläge jeweils neu* getroffen werden. Dies muss bei der Implementierung der Methoden berücksichtigt werden, die an der Durchführung von Rekombinationen beteiligt sind.

Das Allokationsproblem und die Performanz des Optimierungsverfahrens
Bei der Anwendung von Heuristiken für die Lösung von Standortplanungsproblemen spielt die *Rechenzeit für die Zuordnung der Kunden zu den UP* (Allokation) für einen Lösungsvorschlag eine entscheidende Rolle: Die Ermittlung des jeweils nächstliegenden UP für jeden Kunden (und jedes Werk) kann – insbesondere bei größeren Problemexemplaren und bei kapazitierten Problemen – viel Zeit in Anspruch nehmen.[5] Bei heuristischen Lösungsansätzen müssen jedoch meist viele solcher Zuordnungsprobleme gelöst werden. Hier wird ein Ansatz verfolgt, bei dem das Allokationsproblem so weit wie möglich vereinfacht wird:

Zunächst werden Kunden immer so zu den UP zugeordnet, dass die Lieferkosten (Primär- plus Sekundärkosten) minimal werden. Die Kapazitierung von UP wird nur durch Strafkosten und nicht durch alternative Zuordnungen einbezogen (vgl. Abschnitt 3.2.3).

Des Weiteren kann die Zuordnung der Kunden für einen neuen Vorschlag beschleunigt werden, wenn der Vorschlag auf einem Vorgängervorschlag basiert, für den die Zuordnung bekannt ist: Unterscheiden sich Lösungsvorschläge von ihren Vorgängern nur in wenigen Entscheidungsvariablen (d.h. Standorten von UP), müssen in der Regel nur wenige Kunden neu zugeordnet werden. Bei einem Problemexemplar mit m Kunden und n UP müssen pro Werk theoretisch $O(n \cdot m)$ Distanzvergleiche stattfinden, um alle Kunden zuzuordnen. Wird jedoch ausgehend von einem Vorschlag *mit bekannter Zuordnung* nur *ein* UP verschoben, dem im ursprünglichen Vorschlag

[5] Im Grunde genommen handelt es sich bei der Allokation um ein eigenständiges Optimierungsproblem.

beispielsweise k Kunden zugeordnet waren, kann dieser Aufwand auf $O(k \cdot n + (m-k))$ reduziert werden, da für die $(m-k)$ ursprünglich nicht dem UP zugeordneten Kunden pro Werk jeweils nur *ein* Vergleich (mit der Distanz zur neuen Position des UP) stattfinden muss.[6]

Parameter und Entscheidungsvariablen
Die Parameter für Problemexemplare des in Abschnitt 3.2 beschriebenen Problems der Standortplanung bei der Firma WIGeoGIS lassen sich den *Werken*, den *Kunden* und den *Umschlagpunkten* zuordnen. Die Namen der verwendeten Parameter sowie einiger Hilfsfunktionen beginnen mit Kleinbuchstaben. Entscheidungsvariablen sowie Komponenten der Zielfunktion werden mit GROSSBUCHSTABEN notiert.

- Die *Anzahl* der Werke (*factories*) sei mit dem Parameter *fMax*, die Anzahl der Kunden (*clients*) mit *cMax* und die *Nachfrage* (*demand*) eines Kunden i am Werk j mit d_{ij} gegeben.
- Die Koordinaten jedes Werks i seien mit fx_i und fy_i festgelegt. Zur Optimierung bei gegebenen potenziellen Standorten für die UP wären zwar im Prinzip nur die *Entfernungen* der Werke zu diesen potenziellen Standorten notwendig, für eine Visualisierung der Distributionslogistik bietet sich jedoch an, direkt auf die Koordinaten zurückzugreifen. Außerdem werden die Koordinaten der Werke zur *„on demand“*-Berechnung ihrer Distanzen zu UP benötigt, wenn deren potenzielle Standorte nicht fest vorgegeben sind: Die Ermittlung der Distanz erfolgt dann zum Zeitpunkt der Evaluation eines Lösungsvorschlags durch eine Schätzung, die auf der Euklidschen Distanz des Werks zum UP *(„Luftlinieentfernung“)* und einem zusätzlichen Faktor *linToStr („Umwegfaktor“)* mit $linToStr \geq 1$ basiert (vgl. den in Abschnitt 3.2.3 erwähnten *inflation factor*).
- Für jeden Kunden i muss wiederum die jeweilige *Lage* als Koordinatenpaar cx_i, cy_i für die Visualisierung und die Berechnung von Distanzen gegeben sein.
- Des Weiteren kann eine *maximale Lieferdistanz maxD* vom Standort bzw. UP zum Kunden festgelegt werden. Kann ein Kunde aufgrund dieser Beschränkung nicht zugeordnet werden, so fallen pro nachgefragter Tonne Strafkosten in Höhe von *penaltyD* an.
- Die *minimale* und die *maximale Zahl* zu planender Umschlagpunkte (*hubs*) sei im Vorfeld durch die Parameter *hMin* bzw. *hMax* festgelegt. So können sowohl Probleme mit fester ($hMin = hMax$) als auch mit variabler Zahl von Standorten ($hMin < hMax$) behandelt werden.
- Die *maximale Anzahl maxC* und *Nachfrage in Tonnen maxT* zu einem Standort zugeordneter Kunden kann begrenzt werden. Bei Überschreitung solcher Kapazitätsbeschränkungen werden zusätzliche Strafkosten *penaltyC* verteilt.
- Der Optimierung unterliegen die *Koordinaten* (X_i, Y_i) der Umschlagpunkte. Weitere Entscheidungsvariablen sind *Binärvariablen* B_i, mit denen festgelegt wird, ob Umschlagpunkt i tatsächlich geplant wird ($B_i = 1$) oder ob er aus der Planung genommen wird ($B_i = 0$). Als

[6] Die Anzahl der notwendigen Vergleiche kann – sofern Luftlinien-Distanzen verwendet werden – noch weiter eingeschränkt werden, wenn die Kunden zusätzlich entsprechend ihrer Koordinaten auf disjunkte „Kacheln“ (Quadrate gleicher Seitenlänge) verteilt werden. Entspricht die Seitenlänge der maximalen Abdeckungsdistanz, so können alle Distanzvergleiche zwischen Kunden und Umschlagpunkten eingespart werden, die nicht auf gleichen oder benachbarten Kacheln liegen.

Nebenbedingung muss gelten: $hMin \leq \sum_i B_{i=1}^{hMax}$. *Bestehende Standorte* der Umschlagpunkte einer realen Ausgangskonfiguration können definiert sein und auch *fixiert* werden. In letzteren Fall unterliegen die Koordinaten X_i und Y_i für einen Umschlagpunkt i nicht der Optimierung und B_i ist mit 1 fixiert.

- Die *Distanz für den Primärtransport* von einem Werk j zu einem Umschlagpunkt i mit den Koordinaten $(X_i, Y_i) = (x, y)$ sei mit $distP_{j,(x,y)}$ bezeichnet. Sind potenzielle Standorte im Vorfeld festgelegt, so können diese Distanzen durch Spezifikation einer entsprechenden Distanzmatrix festgelegt werden, andernfalls müssen sie wie bereits angedeutet *„on demand“* berechnet werden durch: $distP_{j,(x,y)} = linToStr\sqrt{(fx_j - x)^2 + (fy_j - y)^2}$
- Ähnliches gilt für den *Sekundärtransport* von einem Umschlagpunkt i mit den Koordinaten $(x, y) = (X_i, Y_i)$ zu einem Kunden j. Hier sei die entsprechende Distanz mit $distS_{(x,y),j}$ bezeichnet und ist entweder als Parameter gegeben oder muss analog den Primärdistanzen *„on demand“* berechnet werden.
- Der *Transportkostensatz* pro Tonnenkilometer für den Primärverkehr sei $tcPrim$, der für Sekundärverkehr $tcSec$.
- Die *Fixkosten* eines Umschlagpunkt i, der an den Koordinaten $(X_i, Y_i) = (x, y)$ geplant wird, seien mit $fix_{(x,y)}$ bezeichnet. Fixkosten fallen nur an, wenn der Standort i tatsächlich geplant wird ($B_i = 1$). In diesem Fall können sie konstant oder – beispielsweise bei vorgegebenen potenziellen Standorten – auch in Abhängigkeit der Koordinaten vorliegen.

7.2.2 Modellierung der Zielfunktion

Zu minimieren ist eine Funktion $f(X, Y, B)$, die die Transportkosten $TC(X, Y, B)$, die Fixkosten $FC(X, Y, B)$ sowie die Strafkosten $PC(X, Y, B)$ aggregiert:

$$f(X, Y, B) = \alpha \cdot TC(X, Y, B) + \beta \cdot FC(X, Y, B) + \gamma \cdot PC(X, Y, B)$$

Entscheidungsvariablen sind die Koordinatensätze X mit $X = (X_1, ..., X_{hMax})$ und Y mit $Y = (Y_1, ..., Y_{hMax})$ sowie der Satz von Binärvariablen B mit $B = (B_1, ..., B_{hMax})$, die über die tatsächliche Planung des UP entscheiden. α, β und γ sind Gewichtungsparameter.

Für die Berechnung der Kosten eines Lösungsvorschlags müssen zunächst die Lieferwege festgelegt werden, d.h für jede Nachfrage d_{ik} eines Kunden k am Werk i muss entschieden werden über welchen UP die Lieferung stattfinden soll. Daher bietet sich die Verwendung der Hilfsfunktionen $a_{ijk}(X, Y, B)$ für die Zuordnung (*assignment*) der Kunden zu den Umschlagpunkten an. Die Funktionen zeigen an, ob die Lieferung vom Werk i zum Kunden k über den Standort j (mit den Koordinaten (X_j, Y_j)) erfolgt ($a_{ijk} = 1$) oder nicht ($a_{ijk} = 0$):[7]

$$a_{ijk}(X, Y, B) = \begin{cases} 1 \text{ falls } B_j = 1 \text{ und } distS_{(X_j,Y_j),k} < maxD \\ \qquad \text{und } \forall o \text{ mit } o \neq j \text{ und } B_o = 1 : \\ \qquad\qquad distP_{i,(X_j,Y_j)} \cdot tcPrim + distS_{(X_j,Y_j),k} \cdot tcSec < \\ \qquad\qquad distP_{i,(X_o,Y_o)} \cdot tcPrim + distS_{(X_o,Y_o),k} \cdot tcSec \\ 0 \text{ sonst.} \end{cases}$$

[7] O.b.d.A wird hier und im Folgenden angenommen, dass sich jeweils *ein* günstigster Lieferweg finden lässt.

Die Transportkosten $TC(X,Y,B)$ lassen sich dann auf Grundlage dieser Zuordnung und in Abhängigkeit der Entfernungen der Lieferungen von den Werken über die Standorte zu den Kunden sowie der Höhe der Nachfrage n_{ik} eines Kunden k von einem Werk i bestimmen:

$$TC(X,Y,B) = \sum_{i=1}^{fMax} \sum_{j=1}^{hMax} \sum_{k=1}^{cMax} a_{ijk}(X,Y,B) \cdot d_{ik} \left(distP_{i,(X_j,Y_j))} \cdot tcPrim + distS_{(X_j,Y_j),k} \cdot tcSec \right)$$

Die Fixkosten $FC(X,Y,B)$ sind abhängig von der Lage der UP und der Binärvariablen B:

$$FC(X,Y,B) = \sum_{j=1}^{hMax} B_i \cdot fix_{(X_i,Y_i)}$$

Strafkosten entstehen bei Überschreitung der maximalen Kapazität von Umschlagpunkten (1) sowie bei bei Nichtzuordnung von Kunden (2).

$$PC(X,Y,B) = PCC(X,Y,B) + PCN(X,Y,B)$$

Für (1) verwenden wir wiederum eine Hilfsfunktion $excessC_j$, die für einen Standort j entscheidet, ob die Kapazitätsbeschränkungen $maxC$ und/oder $maxT$ überschritten sind:

$$excessC_j(X,Y,B) = \begin{cases} 1 \text{ falls } \sum_{i=0}^{fMax} \sum_{k=1}^{cMax} a_{ijk}(X,Y,B) \cdot d_{ik} > maxT \\ 1 \text{ falls } \sum_{i=0}^{fMax} \sum_{k=1}^{cMax} a_{ijk}(X,Y,B) > maxC \\ 0 \text{ sonst.} \end{cases}$$

$$PCC(X,Y,B) = \sum_{j=1}^{uMax} excessC_j \cdot penaltyC$$

Für (2) entscheidet die Hilfsfunktion $notDelivered_{ik}(X,Y,B)$, ob die Nachfrage des Kunden k vom Werk i befriedigt werden kann.

$$notDelivered_{ik}(X,Y,B) = \begin{cases} \left(1 - \sum_{j=1}^{hMax} a_{ijk}(X,Y,B)\right) & \text{falls } d_{ik} > 0 \\ 0 & \text{sonst} \end{cases}$$

$$PCN(X,Y,B) = \sum_{i=1}^{fMax} \sum_{k=1}^{cMax} notDelivered_{ik}(X,Y,B) \cdot d_{ik} \cdot penaltyN$$

7.2.3 Anpassung des Framework nach dem Vorgehensmodell

Schritt 1: Implementierung der Agenten und Spezifikation ihres Verhaltens

1.1: Die Identifikation der Agenten

Bei den Objekten, deren Konfiguration Gegenstand der Optimierung ist, handelt es sich um die *UP*. Es wird zwei Sätze von Entscheidungsvariablen geben und zwar zunächst die Koordinatenpaare für die UP und darüber hinaus einen Satz Boolscher Variablen, die über die tatsächliche

Errichtung eines Standorts entscheiden. Ziel eines solchen Agenten ist es dann, eine möglichst günstige Lage für „seinen“ UP zu finden – oder anfallende Fixkosten zu vermeiden, indem der UP gar nicht angelegt wird. Potentielle Standorte können vorgegeben sein oder die Koordinaten des UP sind (innerhalb eines bestimmten Bereichs) frei wählbar.[8]

Die Zielfunktion f kann fast vollständig durch Aggregation partieller, von den Agenten auszuwertender Teilfunktionen berechnet werden:

$$f(X,Y,B) = \sum_{j=1}^{hMax} (TF_j(X_j,Y_j,B_j)) + PCN(X,Y,B)$$

Eine Teilfunktion $TF_j(X_j,Y_j,B_j)$ berechnet die Fixkosten FC_j des Standorts, die Transportkosten TC_j für alle Waren, die über den UP laufen und die anfallenden Strafkosten PCN_j:

$$TF_j(X_j,Y_j,B_j) = \alpha \cdot TC_j(X_j,Y_j,B_j) + \beta \cdot FC_j(X_j,Y_j,B_j) + \gamma \cdot PCC_j(X_j,Y_j,B_j)$$

$$TC_j(X_j,Y_j,B_j) = \sum_{i=1}^{fMax} \sum_{k=1}^{cMax} a_{ijk}(X,Y,B) \cdot d_{ik} \left(distP_{i,(X_j,Y_j))} \cdot tcPrim + distS_{(X_j,Y_j),k} \cdot tcSec \right)$$

$$FC_j(X_j,Y_j,B_j) = B_j \cdot fix_{(X_j,Y_j)}$$

$$PCC_j(X_j,Y_j,B_j) = excessC_j(X,Y,B) \cdot penaltyC$$

Nicht einzelnen Agenten zugeordnet werden können dagegen Strafkosten $PCN(X,Y,B)$ für nicht zugeordnete Kunden. Diese Kosten müssen daher im Anschluss an die partiellen Bewertungen der Agenten durch die Methode *remainderEvaluation* des Umgebungs-Objekts berechnet werden.

1.2: Die Speicherung von Zwischenergebnissen der lokalen Bewertung
Für die spätere Berechnung der Güte-Koeffizienten sind die bei Lieferungen über einen UP anfallenden *Transportkosten* weniger interessant als die *Anzahl* der dem UP zugeordneten Kunden. Da die Verteilung der Kunden über das Untersuchungsgebiet als einigermaßen gleichmäßig angenommen werden kann (vgl. 3.2.3), kann die Anzahl der zugeordneten Kunden – in Relation zur Gesamtzahl der Kunden – als ein lokales Gütekriterium herangezogen werden.[9] Die Zwischenergebnisse für die partielle Bewertung und die Berechnung der Güte-Koeffizienten können der Tabelle 7.4 entnommen werden. Die Evaluation eines Lösungsvorschlags basiert auf Aggregation der Zwischenergebnisse Z_1, Z_2 und Z_3 für alle Agenten sowie der Auswertung der Restfunktion. Um die Zwischenergebnisse mit den Faktoren α, β und γ zu gewichten wird die Aggregationsfunktion *getLocalEvaluationValue* entsprechend überschrieben.

1.3: Die Berechnung von Güte-Koeffizienten
Die Agenten „kennen“ die durchschnittlichen Fixkosten aller potenziellen Standorte sowie die Gesamtanzahl aller Kunden. Aufgrund dieser Informationen können Sie nach der Berechnung der

[8] Um ein allgemein anwendbares Vorgehen zu ermöglichen, suchen die Agenten zunächst *immer* nach einem *beliebigen* neuen Standort für „ihren“ UP. Sind potenzielle Standorte vorgegeben, wird im Anschluss nach den jeweils *nächstliegenden* „erlaubten“ Standorten gesucht. Zur Beschleunigung dieser Suche werden die potenziellen Standorte entsprechend ihrer Koordinaten in einer Kachel-artigen Datenstruktur gehalten, die auch beim Allokationsproblem Performanzgewinne verspricht (vgl. Fußnote 6). Diese Art der Modellierung hat den Vorteil, dass sie für die Optimierung sowohl mit als auch ohne vorgegebene potenzielle Standorte verwendet werden kann. Dieser Vorteil wird allerdings durch leichte Performanzeinbußen für die Suche nach „erlaubten“ Standorten erkauft. Die potenziellen Standorte wurden der Übersichtlichkeit halber nicht als Parameter aufgeführt.

[9] Existieren mehrere Werke, muss jeder Kunde mehrfach zugeordnet werden.

Zwischen-ergebnis	**Beschreibung**	**Berechnung für den Agent des UP j**
z_1	Transportkosten	$z_1^j = TC_j(X_j, Y_j, B_j)$
z_2	Fixkosten	$z_2^j = FC_j(X_j, Y_j, B_j)$
z_3	Strafkosten	$z_3^j = PCC_j(X_j, Y_j, B_j)$
z_4	Zugeordnete Kunden	$z_4^j = \sum_{i=1}^{fMax} \sum_{k=1}^{cMax} a_{ijk}(X, Y, B)$

Tabelle 7.4: *Zwischenergebnisse der UP-Agenten*

Evaluations-Zwischenergebnisse das lokale Verbesserungspotential eines Vorschlags prognostizieren. Liegen beispielsweise die Fixkosten des momentanen Standorts weit über dem Durchschnitt, oder weicht die Anzahl pro Werk zugeordneter Kunden stark vom Quotienten aus der Gesamtzahl der Kunden und der Anzahl der UP ab, so ist anzunehmen, dass durch alternative Wahl des eigenen Standorts Verringerungen der Gesamtkosten erreicht werden können. Außerdem kann jeder Agent die Entfernung seines UP zum jeweils nächsten UP messen. Ist diese Entfernung sehr gering (beispielsweise deutlich geringer als die maximale Abdeckungsdistanz), ist ebenfalls Verbesserungspotential zu erwarten, da durch gleichmäßigere Verteilung der UP vermutlich eine bessere Abdeckung der Kunden möglich wäre. Die Methode *calculateRatios* wird daher so implementiert, dass sie vier Güte-Koeffizienten zurück liefert, die in Tabelle 7.5 beschrieben sind.

Koeffizient	**Beschreibung**
Fixkosten-Koeffizient	Die momentan am UP j entstehenden Fixkosten z_2^j wird in Relation zu den durchschnittlichen Fixkosten gebracht.
Strafkosten-Koeffizient	Der Koeffizient basiert auf dem Zwischenergebnis z_3^j und liefert den Wert 0 oder den Wert 1 zurück, je nachdem ob Strafkosten anfallen oder nicht.
Allokations-Koeffizient	Bei der Berechnung des Allokations-Koeffizenten wird implizit angenommen, dass bei einem guten Lösungsvorschlag die Kunden gleichmäßig auf die UP verteilt sind (vgl. 3.2.3). Er berechnet sich auf Grundlage des Verhältnisses zugeordneter Kunden z_4^j zur Gesamtzahl zuzuordnender Kunden (pro Werk).
Distanz-Koeffizient:	Mit diesem Koeffizienten wird die Distanz zum nächsten anderen UP beurteilt. Ist diese Distanz groß genug, so wird der Wert 1 zurückgeliefert – der Schwellenwert ist vom Anwender einstellbar, beispielsweise bietet sich die doppelte maximale Abdeckungsdistanz als Wert an. Unterhalb des Schwellenwerts liegt der Rückgabewert proportional zur Distanz zwischen 0 und 1.

Tabelle 7.5: *Güte-Koeffizienten der UP-Agenten*

Um Experimente mit unterschiedlichen Gewichtungen dieser Koeffizienten bei der Aggregation zu einem lokalen Zufriedenheits-Wert zu ermöglichen, wurde die Methode *getAggregatedSatisfaction* entsprechend überschrieben, die Gewichtungen können vom Anwender spezifiziert werden.

1.4: Die Initialisierung von Startlösungen

Eine Startlösung wird generiert, indem die Koordinaten aller UP zufällig festgelegt bzw. aus der Liste potenzieller Standorte ausgewählt werden. Alternativ kann als Startlösung eine derzeitige, reale Konfiguration verwendet werden.

1.5: Die Integration lokaler Verbesserungsverfahren, Spezifikation von Situationen und Plänen
Die Methode *getStatus* wurde so implementiert, dass sie Anhand der Güte-Koeffizienten eine der Situationen auswählt, die in Tabelle 7.6 aufgeführt sind. Die Pläne für die Verbesserung eines Vorschlags sind in Tabelle 7.7 aufgelistet. Ebenso wie die Definition der Güte-Koeffizienten ist die bei der problemspezifischen Anpassung getroffene Auswahl der Situationen und Pläne natürlich nur *eine* Möglichkeit – sicherlich können andere Situationen und Pläne spezifiziert werden, mit denen das Framework mindestens ebenso erfolgreich arbeitet.

Status	**Beschreibung**
LocationTooExpensive	Der momentan belegte Standort ist im Vergleich zu anderen potenziellen Standorten zu teuer.
OtherTooClose	Ein anderer UP befindet sich in direkter Nachbarschaft, d.h. beispielsweise näher als die maximale Abdeckungsdistanz.
TooManyClients	Dem UP sind überdurchschnittlich viele Kunden zugeordnet.
NotEnoughClients	Dem UP sind unterdurchschnittlich wenig Kunden zugeordnet.
Satisfied	Der Agent ist mit dem Vorschlag zufrieden.

Tabelle 7.6: *Situationen der UP-Agenten*

Plan	**Beschreibung**	**Anwendbar in Situation(en)**
JumpTo-FreeRegion	Der UP wird an einen ganz neuen Standort gesetzt, der möglichst große Distanz zu anderen UP aufweist.	in allen Situationen
PushOff-Neighbour	Der UP und der nächste UP in der Umgebung rücken auseinander.	*NotEnoughClients*, *OtherTooClose*
Pull-Neighbour	Der UP und der nächste UP in der Umgebung rücken näher zusammen.	*TooManyClients*
MoveLocal	Der UP wird geringfügig verschoben.	*LocationTooExpensive*
PushOff-Neighbours	Alle anderen UP innerhalb einer bestimmten Distanz werden vom UP wegbewegt	*NotEnoughClients*, *OtherTooClose*
Pull-Neighbours	Alle anderen UP innerhalb einer bestimmten Distanz werden zum UP hinbewegt.	*TooManyClients*

Tabelle 7.7: *Pläne der UP-Agenten*

1.6: Die Rekombination von Vorschlägen
Bei einer Rekombination wird eine Lösung, bei der ein Agent noch Verbesserungspotential erkannt hat, kombiniert mit einer Lösung, mit der er lokal zufrieden ist. Dazu wird zunächst der erste Vorschlag kopiert und die Standorte aller UP werden übernommen. Dann werden die Standorte bestimmter UP mit den Belegungen des zweiten Vorschlags überschrieben. Bei einer „normalen“ Rekombination handelt es sich dabei um die UP innerhalb einer gewissen Distanz zum Standort des durch den Agenten repräsentierten UP. Für die Distanz wird ein relativ kleiner, zufälliger Wert herangezogen. Handelt es sich beim zweiten Vorschlag dagegen um den Basisvorschlag einer Koalition, werden die Standorte der UP aller an der Koalition beteiligten Agenten entsprechend des zweiten Vorschlags übernommen.

Mit dieser Art der Rekombination wird versucht, entsprechend der *Building-Block-Hypothese* zusammengehörende Teillösungen gemeinsam zu vererben. Als *Building Block* wird hier eine Reihe von „benachbarten“ UP angesehen, die einen – möglichst zusammenhängenden – Bereich des gesamten Kundengebiets abdecken. Solche „geographischen“ *Building Blocks* werden durch die Anwendung der beschriebenen Form der Rekombination möglichst erhalten.[10]

1.7: Die Auswahl von Kandidaten für die Bildung einer Koalition
Durch die Methode *getPossibleCoalitionMembers* werden mögliche Mitglieder einer Koalition auf Grundlage der Distanz der repräsentierten UP ausgewählt. Agenten, die UP verwalten, die in einem bestimmten Vorschlag „benachbart“ sind, können eine Koalition eingehen, sofern sie mit den Lokalisierungen „ihrer“ Standorte überdurchschnittlich zufrieden sind. Durch die Möglichkeit der beschriebenen „Koalitions-Rekombination“ können diese Belegungen gemeinsam in einen Nachkommensvorschlag übertragen werden. Eine solche Koalition entspricht dann in gewisser Weise der Lösung eines kleinräumigeren Standortplanungsproblems, d.h. eines Problems, das sich nur auf ein eingeschränktes Untersuchungsgebiet bezieht.

Schritt 2: Die Implementierung der Umgebung

2.1: Implementierung der Vervollständigungs- und Reparaturfunktion
Die Zuordnung der Kunden zu den UP wird nicht durch die Agenten selbst, sondern vor der Evaluation zentral durch die Methode *completeAndRepairProposal* durchgeführt. Im Vorschlag ist zu jedem Agenten nicht nur eine Referenz auf die aktuellen Koordinaten des UP sondern auch eine Referenz auf die Menge aller zugeordneten Kunden eingetragen. Da bei neuen Vorschlägen meist nur die Positionen weniger UP verändert werden, können die meisten Zuordnungen des Vorgänger-Vorschlags übernommen werden. Es müssen nur diejenigen Kunden überprüft werden, die UP zugeordnet waren, deren Standorte verändert wurden. Außerdem muss für alle Kunden die Distanz zu „ihrem“ bisherigen UP mit der Distanz zu allen UP mit veränderten Standorten verglichen werden, um die Zuordnung eventuell zu aktualisieren (vgl. 7.2.1).

2.2: Implementierung der Restfunktion für die Bewertung von Lösungsvorschlägen
Die Methode *remainderEvaluation* berechnet die Strafkosten für nicht abgedeckte Kunden.

Schritt 3: Implementierung des Konferenzmonitors

3.1: Graphische Darstellung von Lösungsvorschlägen
Da die Koordinaten aller Kunden, aller Werke sowie die Standorte für UP eines Lösungsvorschlags bekannt sind, kann die Darstellung des Lösungsvorschlags durch die Methode *drawProposal* ohne zusätzliche Angaben erfolgen. Jeder Kunde wird als kleiner Punkt, jeder Umschlagspunkt als großer Punkt dargestellt. Unterschiedliche Farben der Kundenpunkte sowie Verbindungslinien machen die Zuordnung zu den verschiedenen UP deutlich. Da für jeden Vorschlag auch der „Vorgängervorschlag“ bekannt ist, können die Veränderungen gegenüber dem Vorgänger zusätzlich visualisiert werden. Dazu werden die vorherigen Standorte der UP zusätzlich durch grau ausgefüllte Punkte gekennzeichnet. Abbildung 7.2 zeigt die Benutzeroberfläche des auf die Standortplanung angepassten Framework. Man erkennt die (nummerierten) UP und die zugeordneten Kunden für den aktuell besten Vorschlag eines Optimierungslaufs.

[10] Dieses Vorgehen ist allerdings nur dann erfolgversprechend, wenn die Diversizität unter den Lösungsvorschlägen nicht zu groß ist, damit die Vorschläge „rekombinierbar“ bleiben. Dazu können beispielsweise die „Aktionsradien“ der Agenten eingeschränkt, d.h. Standortveränderungen nur innerhalb eines bestimmten Radius erlaubt werden.

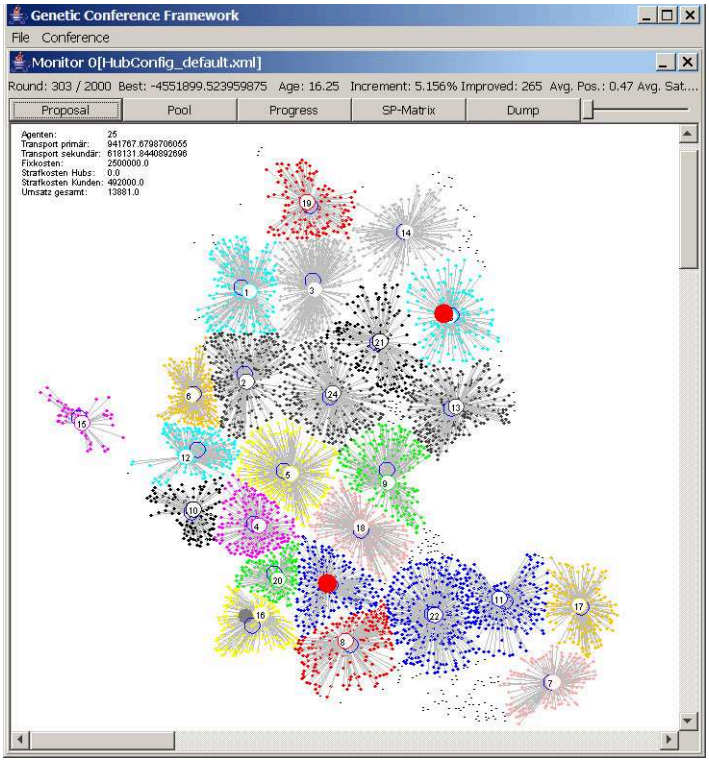


Abbildung 7.2: *Benutzeroberfläche der Anpassung des Framework für die Standortplanung. Geplant wird ein Distributionsnetz für Deutschland, Österreich und Belgien*

3.2: Textuelle Ausgabe von Lösungsvorschlägen
Mit der Methode *dumpProposal* werden Informationen wie die Koordinaten der UP sowie die zugeordneten Kunden ausgegeben.

7.2.4 Distributionsnetzplanung für die BSH mit dem angepassten Framework

Eine Weiterentwicklung des auf die beschriebene Weise angepassten Framework wird von der Firma WIGeoGIS unter anderem in einem Beratungsprojekt für die die Bosch-Siemens-Hausgeräte GmbH (BSH) eingesetzt. Die BSH plant Umstrukturierungen ihrer Distributionsnetze. Das primär bearbeitete Auslieferungsgebiet erstreckt sich über Deutschland, Österreich und Belgien (vgl. Abbildung 7.2). In diesem Gebiet werden jährlich ca. 290.000 Tonnen Ware von drei unterschiedlichen Werken über etwa 23 UP an ca. 33.000 Kunden ausgeliefert – es handelt sich um ein zweistufiges Distributionsnetz. Die Kosten für Unterhalt und Betrieb des Netzes (Fix- und Transportkosten) liegen im zweistelligen Millionenbereich. Ein Ziel des Beratungsprojekts war es, mit der Durchführung von Optimierungsläufen zu ermitteln, ob diese Kosten durch Hinzunahme, Auflassen oder alternative Lokalisation von Umschlagpunkten gesenkt werden können.

Die beschriebene Anpassung des Framework stellte den Prototyp für ein entsprechendes Modul zur Standort-Optimierung dar. Dieses Modul wurde von WIGeoGIS in ein GIS-System integriert. Über das GIS-System können einerseits die Eingangsdaten der Optimierung auf komfortable Weise angepasst werden können. Um beispielsweise die Komplexität der Konfiguration zu

senken (und damit die Geschwindigkeit des Optimierungsverfahrens zu erhöhen), bietet es sich an, nahe beieinander liegende Kunden zunächst zu Clustern zusammen zu fassen. Beachtet man dabei die Warennachfrage der Kunden und die Kapazität der Auslieferungsfahrzeuge, so lassen sich Cluster bilden, deren Kunden gemeinsam mit *einer Tour* abgedeckt werden können.[11] Da die Entfernungen von den UP zum Zentrum des Clusters dabei im Allgemeinen deutlich höher ausfallen im Vergleich zu den Entfernungen der Kunden eines Clusters untereinander, können die Längen der Auslieferungstouren relativ gut durch Pendeltouren von den UP zu den Kundenclustern angenähert werden. Ergebnisse der Optimierung können direkt an das GIS-System übertragen, dort angezeigt und eventuell manuell nachbearbeitet werden.

Für das beschriebene Auslieferungsgebiet wurden Optimierungsläufe mit ca. 4800 Kundenclustern durchgeführt. Die Ergebnisse sind vielversprechend: Auf einem üblichen 2GHz PC-System wurden bereits nach wenigen Minuten Lösungen gefunden, die (basierend auf den Modellannahmen) gegenüber der derzeitigen Konfiguration mehr als 10% der Kosten einsparen. Nach weiteren Feinabstimmungen hat sich die Software bei der BSH als anerkanntes Werkzeug für die Optimierung bestehender und Planung neuer Distributionsnetze im Unternehmen etabliert und wird inzwischen auch erfolgreich in anderen Projekten eingesetzt. Des Weiteren plant die WIGeoGIS, den Algorithmus in Zusammenarbeit mit dem Autor für die Bearbeitung anderer Planungsszenarien weiterzuentwickeln und als Erweiterung für GIS-Software zu vertreiben.

7.2.5 Geplante Erweiterungen

Geplante Erweiterungen der Anpassung des Framework bestehen darin, neben der Planung von Umschlagpunkten für Distributionsnetze auch die Planung von *Filialstandorten* und von *Vertriebsgebieten* zu ermöglichen. Für diese Erweiterungen müssen Anpassungen an den Bewertungsfunktionen vorgenommen werden, außerdem müssen zusätzliche (bzw. alternative) Situationen und Pläne für die Agenten vorgesehen werden. Die Erweiterungen werden voraussichtlich in Zusammenarbeit mit der Firma WIGeoGIS erfolgen.

Bei der *Filialstandortplanung* werden keine Distributionsnetze für die *Auslieferungen* von Waren zu Kunden geplant. Stattdessen sollen in einem Gebiet potenzieller Kunden die lukrativsten Gebiete für zu planende *Filialen* (z.B. Einzelhandelsbetriebe) gefunden werden. Anders als bei der Distributionsnetzplanung wird also keine Vollabdeckung der Kunden angestrebt. Wiederum spielt die Entfernung der Kunden zu den (Filial-)Standorten eine entscheidende Rolle. Diese werden jedoch nun nicht mehr minimiert, um die Transportkosten zu begrenzen, sondern um einen möglichst hohen Anteil des Kundenpotentials in einem Untersuchungsgebiet abzuschöpfen. Die Kunden werden nicht beliefert, sondern erwerben Waren selbständig in den Filialen. Dabei wird die Annahme zugrunde gelegt, dass der zu erwartende Umsatz (und Gewinn) an einem potenziellen Kunden mit zunehmender Entfernung abnimmt – da der Kunde die Ware möglicherweise gar nicht oder in anderen Filialen erwirbt. Berücksichtigt werden können hier eventuell auch die Filialstandorte von Konkurrenzbetrieben. Das Kundenpotential muss daher unter Berücksichtigung der Entfernungen unter mehreren Filialen aufgeteilt werden (vgl. 3.2.2).

Bei der *Vertriebsgebietsplanung* ist das Ziel, ein Gebiet mit Vertriebsgebieten komplett abzudecken. Die Vertriebsgebiete werden anhand eines Zielpotenzials gebildet, das für jedes Gebiet

[11] Alternativ können auch Cluster gebildet werden, deren Kunden jeweils mit genau zwei, drei oder mehr Touren beliefert werden können.

möglichst ähnlich sein sollte. Als Potenzial kann dabei zum Beispiel der zu erwartende Umsatz oder die Anzahl der Kunden dienen. Die Anzahl der Gebiete ergibt sich durch den Quotienten aus Gesamtpotential und Zielpotenzial. Die Vertriebsgebiete sollten möglichst kompakt sein, eine maximale Abdeckungsdistanz kann daher angegeben werden.

7.2.6 Ein Vergleich mit anderen Lösungsverfahren

Im Folgenden werden einige bekannte Lösungsverfahren für Standortplanungsprobleme beschrieben und Ähnlichkeiten und Unterscheide zur beschriebenen Anpassung des Framework aufgezeigt. Dabei soll die Anpassung des Framework auf Standortplanungsprobleme im Folgenden als GC_S bezeichnet werden. Da mit den Verfahren nicht nur UP geplant werden, soll allgemein von *Einrichtungen* gesprochen werde.

Add-und Drop-Verfahren, Standort-Austausch-Verfahren:
Zwei bekannte *Konstruktionsheuristiken* für diskrete Multi-Standortplanungs-Probleme mit vorgegebener Zahl zu planender Einrichtungen sind das *Add-* und das *Drop-Verfahren*. Das Add-Verfahren startet mit einer leeren Liste von Standorten, an denen Einrichtungen geplant werden sollen. Iterativ wird in jedem Schritt derjenige Standort zur Liste hinzugefügt, dessen Hinzunahme am wenigsten zusätzliche Kosten verursacht bzw. die Gesamtkosten am weitesten absinken lässt. Entsprechend startet man beim Drop-Verfahren mit einer Liste, in der alle potenziellen Standorte aufgenommen sind und entfernt iterativ den Standort, bei dem dadurch die größtmögliche Verbesserung (bzw. geringst-mögliche Verschlechterung) der Zielfunktion erreicht wird. Die Verfahren enden, wenn die geforderte Anzahl von Einrichtungen erreicht ist [DD96, S.60ff].

Ein *Greedy-Verbesserungsverfahren* stellt das *Standort-Austausch-Verfahren* dar. Ausgehend von einem Lösungsvorschlag mit einer vorgegebenen Zahl von Einrichtungen wird iterativ der Standort einer (momentan) zu planenden Einrichtung mit einen potenziellen Standort getauscht, an dem derzeit keine Einrichtungen geplant ist. Die beiden Standorte werden dabei so ausgewählt, dass sich die maximal mögliche Verbesserung der Zielfunktion ergibt.

Ebenso wie das GC_S-Verfahren handelt es sich beim Standort-Austausch-Verfahren um eine Verbesserungsheuristik, während Add-und Drop-Verfahren Konstruktionsheuristiken darstellen. Im Unterschied zum GC_S-Verfahren können mit allen Verfahren nur diskrete Probleme (mit vorgegebenen potenziellen Standorten) bearbeitet werden und die Zahl zu planender Standorte muss gegeben sein. Aufgrund der rechenintensiven Evaluation von Lösungsvorschlägen bieten sich die Verfahren nicht an, wenn die Anzahl potenzieller Standorte sehr hoch ist. Es wird kein zusätzliches Problemwissen – insbesondere über die Möglichkeiten der partiellen Bewertung und lokalen Verbesserung von Vorschlägen – ausgenutzt, wie dies beim GC_S-Verfahren der Fall ist.

Alternierende Lokations-Allokations-Heuristik
Die *alternierende Lokations-Allokations-Heuristik* von Cooper [Coo64] macht sich die inhärente Zweiteilung von Standort-Einzugsbereichs-Problemen zunutze: Nach Generierung einer zufälligen Startkonfiguration für die Einrichtungen folgt jeweils alternierend eine Allokationsphase, in der die Kunden den momentan jeweils nächsten Einrichtungen zugeordnet werden und eine Lokationsphase, in der für jedes der entstandenen Vertriebs- bzw. Auslieferungsgebiete ein einfaches Medianproblem gelöst wird, und so neue „Koordinaten“ für die Einrichtungen ermittelt werden.

Bei der alternierenden Lokations-Allokations-Heuristik handelt es sich – wie beim GC_S-Verfahren – um eine Heuristik, mit der auch kontinuierliche Probleme bearbeitet werden können. Während

die Heuristik von Cooper dabei zunächst auf einstufige Distributionsnetze zugeschnitten ist[12], können mit dem GC_S-Verfahren auch zweistufige Probleme bearbeitet werden. Im Unterschied zum GC_S-Verfahren wird das Untersuchungsgebiet immer voll abgedeckt, es werden weder Kapazitätsbeschränkungen für die Einrichtungen noch maximale Abdeckungsdistanzen einbezogen. Außerdem muss die Zahl der zu planenden Einrichtungen vorgegeben sein. Als „Wissen“ wird die Eigenschaft der Problemexemplare verwendet, dass sich die Einrichtungen bei einer optimalen Lösung jeweils am Medianzentrum der zugeordneten Kunden („Vertriebsgebiet“) befinden. Leider ist die Bestimmung des Medianzentrums nur näherungsweise möglich und relativ rechenintensiv.

Optimierung mit Hill-Climbing bzw. -Descending-Verfahren
Bei der Bearbeitung von *kontinuierlichen* Problemen mit Hill-Climbing (bzw. Descending)-Methoden wird ein bestehender Vorschlag iterativ in Richtung des Gradienten der Zielfunktion verändert. Bei der Bearbeitung von Standortplanungsproblemen können *Näherungen der partiellen Ableitungen* für die Bestimmung des Gradienten verwendet werden. Hauptnachteil der Methode ist der große Zeitaufwand für die Bewertung der Lösungsvorschläge insbesondere bei größeren Problembeispielen. Für jede Bewertung ist zunächst eine Zuordnung (Allokation) aller Kunden zu den Einrichtungen erforderlich. Bei der Methode des steilsten Abstiegs wird eine Bewertung (und damit eine jeweils auch eine Allokation) für *jede partielle Ableitung* notwendig, um die Richtung des Gradienten zu bestimmen.

Simulated Annealing (SA)
Um die Anzahl der Evaluierungen zu begrenzen, können auch Metaheuristiken wie Simulated Annealing zur Lösung von Standortplanungs-Problemen eingesetzt werden. Als neue Lösungskandidaten werden jeweils zufällige Lösungsvorschläge aus der Nachbarschaft eines momentanen Vorschlags herangezogen, d.h. Vorschläge, bei dem sich die Koordinaten der Einrichtungen nur wenig unterscheiden – die Berechnung von Gradienten entfällt.

Im Laufe des Beratungsprojekts der Firma WIgeoGIS wurde ein Studierender der Geographie der LMU München im Zuge einer Diplomarbeit mit der Evaluation von Lösungsverfahren für Standortplanungsprobleme beauftragt. Bei Tests konnten mit dem GC_S-Verfahren signifikant bessere Ergebnisse erzielt werden, als mit einer SA-Implementierung. Außerdem war das GC_S-Verfahren deutlich schneller (vgl. [Buc05]). Die Vorteile des GC_S-Verfahrens sind hier sicherlich einerseits in der Rekombination von Lösungen an sich zu sehen, die SA nicht vorsieht. Außerdem werden die Einrichtungen beim SA-Verfahren in zufällige Richtungen verschoben, während Variationsoperatoren beim GC_S-Verfahren jeweils entsprechend der spezifischen Situation ausgewählt werden.

Branch and Bound
Blunck [Blu05] beschreibt die Optimierung von Hubstandorten in Transportnetzen mit einem Branch-and-Bound-Verfahren. Statt den Kundenstandorten sind Depotstandorte gegeben und im Unterschied zu den hier betrachteten Distributionsnetzen handelt es sich um Netze, bei denen Güter zwischen beliebigen Depots ausgetauscht werden können. Güter können des Weiteren auf ihrem Weg zwischen zwei Knoten mehrere Hubs durchlaufen, die Hubs werden dabei als Sortierstationen genutzt. Solche Netze treten beispielsweise bei Kurier-, Express und Postdienstleistern auf. Zur Bestimmung einer Startlösung setzt er das Add- und Drop-Verfahren ein. Die Berechnung der unteren Schranken erfolgt durch ein Näherungsverfahren.

[12] Allerdings wäre auch eine Adaption der Heuristik für zweistufige Netze möglich, bei der die Zuordnung der Kunden und die Positionierung der Einrichtungen entsprechend angepasst werden.

Mit dem Ansatz können Problemexemplare mit bis zu 100 Depotstandorten bearbeitet werden. Die potenziellen Standorte sowie die Anzahl zu planender Hubs ist im Vorfeld gegeben.

7.2.7 Performanzvergleich der Anpassung mit Simulationen der Basisverfahren MAS und GA

Um die Performanz der Genetischen Konferenz im Vergleich zu den Basisverfahren GA und MAS zu messen, wurden jeweils 10 Optimierungsläufe für das BSH-Distributionsnetzes mit dem integrierten Verfahren sowie mit zwei Varianten durchgeführt, die einen GA und ein Greedy-MAS getrennt simulieren. Bei ersterer Variante finden keine Delegation von Lösungsvorschlägen und keine Situations-spezifische Operatorauswahl statt. Bei letzterer Variante wird nur an *einem* Lösungsvorschlag gearbeitet, Rekombinationen können daher nicht durchgeführt werden.

Abbildung 7.3 zeigt die gemittelten Fitness-Werte der jeweils besten gefundenen Lösungsvorschläge – die Optimierung wurde jeweils mit der derzeitigen Konfiguration des Distributionsnetzes als Ausgangsvorschlag gestartet. Während die MAS-Variante zunächst noch schneller Verbesserungen findet als das verknüpfte Verfahren, stagniert sie später oft in schlechteren lokalen Optima. Das verknüpfte Verfahren liefert ab ca. 400 ausgewerteten Vorschlägen bessere Ergebnisse. Auch wenn die Werte von GC- und MAS-Variante dabei relativ ähnlich scheinen, kann die Signifikanz der Abweichungen der Mittelwerte nach 2000 ausgewerteten Vorschlägen noch bei einer Irrtumswahrscheinlichkeit $\alpha = 1,5\%$ nachgewiesen werden (Student'scher t-Test). Die GA-Variante zeigt langsamere, relativ stetige Verbesserungen und fällt relativ weit zurück.

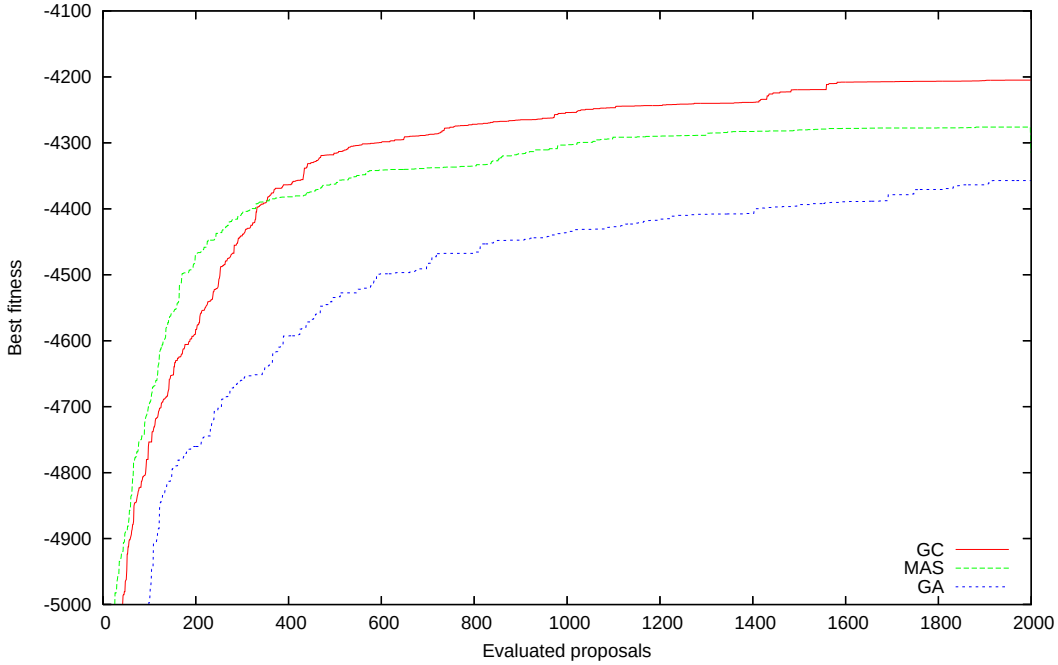


Abbildung 7.3: *Performanzvergleich der Genetischen Konferenz (GC) mit Simulationen eines Genetischen Algorithmus (GA) und eines Greedy-MAS (MAS) bei der Standortplanung*

7.3 Anwendungsbeispiel II: Anpassung zur Optimierung Integraler Taktfahrpläne

Als zweiter Anwendungsfall wird die Anpassung des Framework zur Optimierung Integraler Taktfahrpläne (ITF) beschrieben.

7.3.1 Vorüberlegungen zur Anpassung des Framework

Die Qualität eines Fahrplans
Ähnlich wie bei der Standortplanung werden bei der Optimierung von ITF gleichzeitig mehrere Ziele verfolgt: Die Reisezeiten für die Kunden sollen möglichst gering sein, ebenso wie die Kosten für Infrastrukturmaßnahmen sowie für den Betrieb des Fahrplans. Die Qualität des Fahrplans wird also nicht (nur) aus der Sicht des Kunden, sondern auch betriebswirtschaftlich beurteilt: Kürzere Reisezeiten können oft nur durch Infrastrukturmaßnahmen und höhere Betriebskosten erreicht werden. Die Kenngrößen, die für diesen Ansatz zur Berechnung der Qualität eines Fahrplans herangezogen werden, entsprechen im Wesentlichen denen des Programmsystems *OptiTakt*, diese werden in [HGH05] detailliert erläutert. Im dort beschriebenen Modell ergibt sich die Güte Q (*Quality*) eines Fahrplans aus den im Netz auftretenden Verlustminuten PLT (*Period Loss Time*) pro Taktperiode, den Kosten für notwendige Infrastrukturmaßnahmen IC (*Infrastructure Costs*) und den Kosten für den Betrieb des Fahrplans TM (*Total Mileage*):

$$Q = -\alpha \times PLT - \beta \times IC - \gamma \times TM$$

Die Kenngrößen PLT, IC und TM gehen mit den Gewichtungsparametern α, β und γ in die Berechnung der Qualität ein. Sie werden in Abschnitt 7.3.3 näher erläutert.

Taktfrequenzen / Shuttlemodell
Wie bereits in Abschnitt 3.3.3 beschrieben soll die Anpassung des Framework sich auf die Grobplanung von ITF basierend auf einem Shuttle-Modell beschränken. Modellhaft „pendeln“ also die Fahrzeuge jeweils zwischen zwei „benachbarten“ Haltepunkten. Gegenstand der Optimierung sind die Abfahrtszeiten der Shuttle, mögliche Infrastrukturmaßnahmen für deren Beschleunigung sowie ihre Bedienfrequenz. Ausgehend von einem „Grundtakt“ *cycle* des Fahrplans sind für die Shuttle dabei nur regelmäßige, ganzzahlige Bedienfrequenzen möglich.[13] Auch wenn diese Modellierung zunächst unflexibel erscheinen mag, da Strecken nur in Intervallen gemäß der Taktrate befahren werden können, bildet sie die Grundlage für die Bildung eines „echten“ ITF mit Linien, deren Abfahrtszeiten auf allen Strecken bezüglich des Grundtakts aufeinander abgestimmt sind.

Knotenzeiten
Ziel der Optimierung ist die Identifikation sinnvoller ITF-Knoten. Die Knotenzeiten gehen jedoch nicht als Entscheidungsvariablen in die Zielfunktion ein, sie sollen sich stattdessen „selbständig“ ergeben durch die Abfahrtszeiten der an einem Haltepunkt anliegenden Shuttle. Dabei sollen nicht nur Null- und Halbknoten gebildet werden, sondern auch Richtungsknoten. Für jeden Bahnhof wird eine Knotenzeit gesetzt, wenn mehr als 50% der Abfahrtszeiten den Haltepunkt bedienender Shuttle diese einhalten. Fahren beispielsweise mehr als die Hälfte der Shuttle kurz nach einer vollen Stunde ab, wird die Knotenzeit des Haltepunkts auf 0 gesetzt.

[13] Beispielsweise können bei die Shuttle einem Grundtakt von einer vollen Stunde alle 60 min. (Bedienfrequenz $F = 1$), 30 min ($F = 2$), 20 min. ($F = 3$), 15 min. ($F = 4$), 12 min. ($F = 5$), 10 min. ($F = 6$) usw. verkehren.

7.3.2 Parameter und Entscheidungsvariablen

Wie bei der Standortplanung werden Parameter und Hilfsfunktionen beginnend mit Kleinbuchstaben und Entscheidungsvariablen sowie Komponenten der Zielfunktion in GROSSBUCHSTABEN notiert. Gewichtungsparameter tragen griechische Buchstaben.

Parameter

- Als globaler Parameter des Fahrplans sei zunächst der Grundtakt *cycle* in Minuten gegeben. Die Taktperiode bei der Bearbeitung von Fahrplänen im SPNV könnte beispielsweise $cycle = 60$ oder $cycle = 120$ betragen und gibt das Intervall in Minuten an, innerhalb der jedes in den ITF eingebundene Shuttle *mindestens einmal* verkehrt.
- Des Weiteren sei die Anzahl der Haltepunkte (stations) mit $stMax$ und die Anzahl der Shuttle mit $shMax$ gegeben.
- Die minimale Umsteigezeit (*changeover time*) an einem Haltepunkt h betrage $minCT_h$. Damit ist die Zeit gemeint, die ein Fahrgast nach der Ankunft eines Verkehrsmittels am Haltepunkt benötigt, um ein anderes, abfahrendes Verkehrsmittel zu erreichen.
- Zur graphischen Darstellung werden für jeden Haltepunkt h außerdem die Koordinaten x_h und y_h benötigt.
- Die momentane Fahrzeit (*driving time*) eines Shuttle s vom Start- zum Zielhaltepunkt betrage dt_s. Die tatsächliche Fahrzeit kann eventuell durch Beschleunigungsmaßnahmen modifiziert werden. Neben Investitionen in das Verkehrsnetz sind oft auch Beschleunigungen durch den Einsatz alternativen Rollmaterials erreichbar (im Schienenverkehr beispielsweise durch Neigetechnik). Hier wird jedoch davon ausgegangen, dass bereits das schnellstmögliche Rollmaterial eingesetzt wird, dt_s ist also die *schnellstmögliche* Fahrzeit für den Shuttle s.
- Die Distanz der Strecke, auf der ein Shuttle s verkehrt, sei mit $dist_s$ gegeben.
- Die Anzahl der Personen, die vom Haltepunkt h_1 zum Haltepunkt h_2 reisen, betrage w_{h_1,h_2}.

Entscheidungsvariablen

- Für jedes Shuttle s ist die Frequenz F_s eine Entscheidungsvariable. Ein Shuttle verkehrt innerhalb der Taktperiode *cycle* in regelmäßigen Abständen genau F_s ($F_s \in \mathbb{N}$) mal.
- Weitere Entscheidungsvariable ist die erste Abfahrtszeit (*departure*) D_s eines Shuttle s am Starthaltepunkt innerhalb der Taktperiode *cycle*. Für $F_s > 1$ ergeben sich $F_s - 1$ weitere Abfahrtszeiten am Starthaltepunkt zu den Zeiten $D_s + k \cdot \frac{cycle}{F_s}$ mit $k = 1, ..., (F_s - 1)$. Die Abfahrtszeit(en) am Zielhaltepunkt sowie die Ankunftszeiten lassen sich durch die Fahrzeit und die Symmetrieeigenschaften des Fahrplans bestimmen.
- Zuletzt kann die Fahrzeit eines Shuttle s beeinflusst werden durch die Entscheidungsvariable S_s. S_s gibt den Anteil der ursprünglichen Fahrzeit an, der nach einer Infrastrukturmaßnahme (Streckenausbau usw.) eingespart werden kann (*speedup*). Die Fahrzeit nach einer Infrastrukturmaßnahme $dtNew_s$ bemisst sich damit durch:

$$dtNew_s = (1 - S_s)\, dt_s$$

7.3.3 Modellierung der Zielfunktion

Entsprechend dem in den Vorüberlegungen beschriebenen Modell (vgl. 7.3.1) wird eine zu minimierende Zielfunktion f (f = -Q) des Problems definiert mit:

$$f(F, D, S) = \alpha \times PLT(F, D, S) + \beta \times IC(S) + \gamma \times TM(F)$$

Verlustzeiten (*PLT*)

Mit der *Period Loss Time (PLT)* wird die über die Taktperiode (mit den Fahrgastzahlen gewichtete) gemittelte Differenz zwischen tatsächlicher Reisezeit (*actual travel time*) att_{ij}^{t} und theoretisch bestmöglicher Reisezeit (*possible travel time*) ptt_{h_1,h_2} zwischen allen Haltepunkten h_1 und h_2 und für jede Minute t der Taktperiode[14] gemessen. Dabei soll zur tatsächlichen Reisezeit auch die Wartezeit des Fahrgasts am Starthaltepunkt zählen.

$$PLT(F, D, S) = \sum_{h_1=1}^{stMax} \sum_{h_2=1}^{stMax} \frac{w_{h_1,h_2}}{cycle} \sum_{t=0}^{cycle-1} \left(att_{h_1,h_2}^{t}(F, D, S) - ptt_{h_1,h_2} \right)$$

Die theoretisch bestmögliche Reisezeit ptt_{h_1,h_2} zwischen den Haltepunkten h_1 und h_2 kann unabhängig von den Entscheidungsvariablen bereits im Vorfeld der Optimierung allein auf Grundlage der Fahrzeiten *dt* der Shuttle bestimmt werden. Die Berechnung kann über eine Pfadsuche in einem Graphen mit den Haltepunkten als Knoten und den Shuttle als Kanten erfolgen. Dagegen können die tatsächlichen Reisezeiten $att_{h_1,h_2}^{t}(F, D, S)$ nur für eine gegebene „Startminute t“ und für konkrete Werte der Entscheidungsvariablen F, D und S ermittelt werden.

Ist die Auswertung von Vorschlägen langsam, so können bei fester Gesamt-Rechenzeit weniger Vorschläge ausgewertet werden. Das schränkt die Anzahl der Generationsübergänge und/oder die Populationsgröße ein und kann daher schlechtere Resultate zur Folge haben. Daher sollte die Rechenzeit für die Bestimmung der tatsächlichen Reisezeiten begrenzt werden. Für die Begrenzung der Rechenzeit kommen die Einschränkung der Häufigkeit der Ausführung des Suchalgorithmus durch Beschränkung auf bestimmte Ankunftszeiten an den Startpunkten der Reise (1), die Begrenzung der Suchtiefe des Algorithmus (2) und die Beschleunigung des Suchalgorithmus an sich (3) in Betracht.

1. Beschränkung auf bestimmte Ankunftszeiten
Die Suche nach kürzesten Verbindungen muss theoretisch für jede Minute der Taktzeit individuell erfolgen. Über einen speziellen Parameter kann die Berechnung jedoch auf *größere Intervalle* (z.B. alle 15 Minuten) beschränkt werden.

2. Die Begrenzung der Suchtiefe
Bekannt ist die Anzahl der Reisenden für jedes Paar von Haltepunkten. In vielen Fällen legt ein Großteil der an einem Ausgangspunkt startenden Reisenden nur geringe Distanzen im Netz zurück. Um die Suchtiefe zu begrenzen, kann der Aufbau des Suchbaums abgebrochen werden, wenn die kürzesten Verbindungen zu den jeweiligen Reisezielen für einen bestimmten, vom Anwender spezifizierbaren *Anteil* der Reisenden bereits gefunden wurden. So wird vermieden, dass ein Großteil des Rechenaufwands für einen sehr kleinen Anteil der Reisenden betrieben wird.

[14] Um den Nutzen höherer Bedienfrequenzen in die Zielfunktion einfließen zu lassen, reicht es nicht aus, die kürzestmögliche Verbindung zwischen Start- und Zielhalt zu betrachten. Stattdessen wird eine gleichwahrscheinliche Ankunftszeit von Reisenden am Starthaltepunkt über die Taktperiode hinweg unterstellt (*Modell der zufälligen Ankunft am Starthaltepunkt*).

3. Die Beschleunigung des Suchalgorithmus an sich
Sind wie im vorliegenden Fall *geographische Informationen zur Lage der Knoten* vorhanden, so kann versucht werden, den verwendeten kürzeste-Wege-Algorithmus mittels heuristischer Methoden zu beschleunigen, indem zielgerichtet, d.h. vorrangig „in der richtigen Richtung“ gesucht wird (vgl. [WW03], [WW07]). Beispielsweise kann der Algorithmus von Dijkstra [Dij59] für die Ermittlung des kürzesten Wegs zwischen zwei Knoten durch eine zusätzliche Schätzfunktion für den Luftlinienabstand der Knoten erweitert werden. Ein solches Vorgehen bietet sich jedoch nur dann an, wenn die kürzeste Verbindung zu *einem* Ziel gesucht wird. Im vorliegenden Fall sind jedoch ohnehin die kürzesten Verbindungen zu *allen* anderen Haltepunkten gesucht, so dass die mittels der Heuristik möglicherweise „eingesparten“ Expansionen des Suchbaums (zum größten Teil) ohnehin für die kürzesten Wege zu den anderen Zielen benötigt werden. Ob und wie ein solcher zielbasierter (heuristischer) Ansatz auch für die Suche nach *mehreren* Zielen verwendet werden kann, müsste noch geklärt werden [KSSSW07].

Wagner et al. [KSSSW07] schlagen für die Suche nach *mehreren* Zielen einen hierarchischen Ansatz vor. Dabei wird zunächst eine so genannte *highway hierarchy* aufgebaut, die aus unterschiedlichen Ebenen besteht. Während die unterste Ebene dem Ausgangsgraphen entspricht, werden auf den höheren Ebenen jeweils nur bestimmte „wichtige“ Kanten und Knoten der darunterliegenden Ebene aufgenommen. Die Idee basiert darauf, von Start- und Zielknoten aus eine beidseitige Suche zu den jeweils nächstliegenden Knoten auf den verschiedenen Ebenen durchzuführen. Die verbleibende Distanz zwischen jeweiligen „Eingangsknoten“ der selben Ebene können beispielsweise vorausberechnet werden. Durch die Verwendung so genannter *buckets*, in denen bestimmte Pfadteile gespeichert werden, wird außerdem das mehrfache Abschreiten derselben Pfade vermieden.

Im vorliegenden Fall besteht das zusätzliche Problem, dass vorher berechnete Reisezeiten (und die entsprechenden Pfade) von Zwischenknoten zu einem Zielknoten nur bedingt wiederverwendet werden können, da sich für *unterschiedliche Ankunftszeiten* am Zwischen-Haltepunkt im Allgemeinen *verschiedene Reisezeiten und Pfade* ergeben.

Hier soll ein anderer Ansatz verfolgt werden, der ebenfalls auf einer Hierarchie unter den Haltepunkten beruht. Zunächst wird eine vollständige Suche nach allen kürzesten Reisezeiten (und den entsprechenden Pfaden) für alle Ausgangshaltepunkte ab einem bestimmten, vom Anwender einstellbaren Knotengrad durchgeführt (Knoten der Hierarchieebene 1). Für die übrigen Knoten (Hierarchieebene 2) wird der Suchbaum nicht über Knoten der Hierarchieebene 1 hinaus expandiert. Pfade zu den dann noch nicht erreichten Haltepunkten ergeben sich durch Ermittlung eines kürzesten bzw. schnellsten Pfads vom Ausgangsknoten über einen Knoten der Hierarchieebene 1 und des entsprechenden Restpfads – die Restpfade wurden ja vorher bereits ermittelt. Dabei ergeben sich möglicherweise geringe Abweichungen zu den tatsächlichen Reisezeiten, wenn die Restpfade nicht für alle, sondern nur für bestimmte Abfahrtsminuten vorausberechnet wurden (siehe 1.). Die Begrenzung der Suchtiefe (2) kommt nur für die Haltepunkte der Hierarchieebene 2 als Ausgangspunkte zur Anwendung.

Infrastukturkosten (*IC*)

Für die Bestimmung der Qualität des Fahrplans werden außerdem die Kenngrößen *IC* und *TM* berechnet. Mit der Kenngröße *IC* werden die Kosten der notwendigen Infrastrukturmaßnahmen erfasst. Die Höhe dieser Kosten ist im hier verwendeten Modell abhängig von der relativen Beschleunigung und der Distanz der Strecke. Beschleunigungsmaßnahmen durch den Einsatz al-

ternativen Rollmaterials (deren Kosten nicht von Distanz der zu befahrenden Strecke abhängig sind) werden hier nicht betrachtet (vgl. 7.3.2). Die Infrastrukturkosten werden modellhaft berechnet als:

$$IC(S) = \sum_{s=1}^{shMax} dist_s \cdot \left(\left(\frac{1}{1 - S_s} \right)^{\lambda} - 1 \right) \left[= \sum_{s=1}^{shMax} dist_s \cdot \left(\left(\frac{dt_s}{dtNew_s} \right)^{\lambda} - 1 \right) \right]$$

Dabei wird ein zusätzlicher Parameter λ verwendet, über den die Intensität der Kostensteigerungen für resultierende Fahrzeitgewinne eingestellt werden kann. Abbildung 7.4 zeigt die Infrastrukturkosten pro Entfernungseinheit in Abhängigkeit vom Grad der Beschleunigung für unterschiedliche λ-Werte.

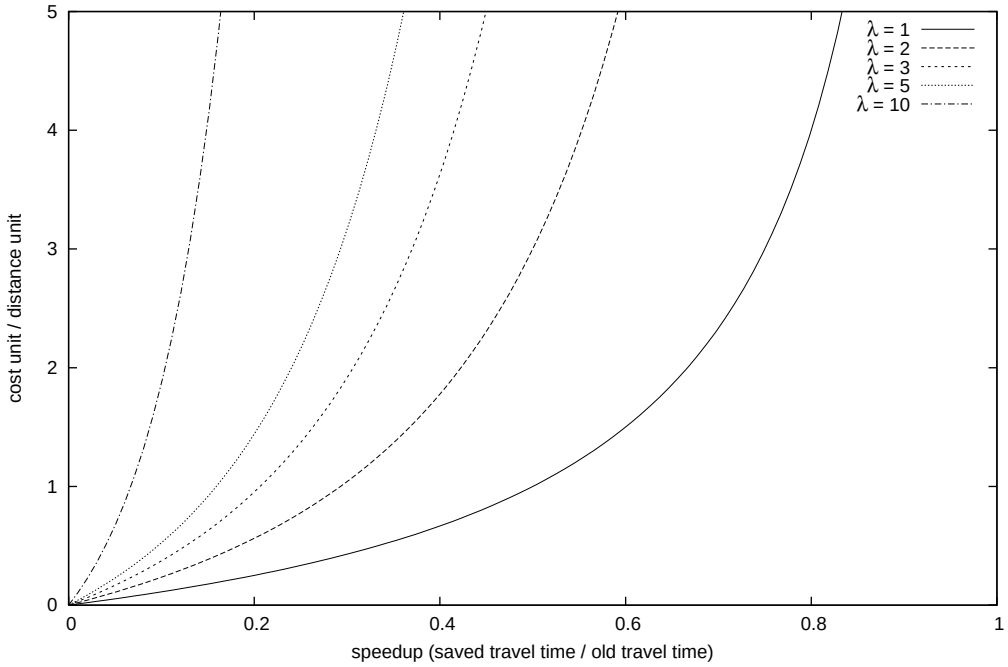


Abbildung 7.4: *Infrastrukturkosten je Streckeneinheit in Abhängigkeit vom Grad der Beschleunigung und dem Parameter λ*

Zugkilometer (TM)

Mit der Kenngröße TM wird die Anzahl von Zugkilometern berechnet, die nach Fahrplan in einer Taktperiode gefahren werden müssen. Die Entscheidungsvariable F_s bezeichnet die Frequenz des Shuttle s, die zurückzulegende Distanz ist $dist_s$.

$$TM(F_s) = \sum_{s=1}^{shMax} dist_s \cdot F_s$$

Die so errechneten Infrastrukturkosten IC und Zugkilometer TM gehen mit den Gewichtungsparametern β und γ in die Zielfunktion ein.

7.3.4 Anpassung des Framework nach dem Vorgehensmodell

Schritt 1: Implementierung der Agenten und Spezifikation ihres Verhaltens

1.1: Die Identifikation der Agenten:
Bei der Optimierung von ITF werden sowohl die *Haltepunkte* als auch die auf den Strecken des Verkehrsnetzes verkehrenden *Shuttle* durch Agenten repräsentiert.[15] Diese Modellierung bietet sich an, da sich die Zielfunktion so auf einfache Weise in partielle Evaluierungsfunktionen (Teilfunktionen) zerlegen lässt, die den einzelnen Agenten zugeordnet werden können:

$$f(F,D,S) = \sum_{h=1}^{stMax} HTF_h(F,D,S) + \sum_{s=1}^{shMax} STF_s(F,D,S)$$

Teilfunktionen der Shuttle:
Für die Shuttle ergeben sich die Teilfunktionen STF_s, mit denen die Infrastrukturkosten IC und die Bedienkosten TM berechnet werden:

$$STF_s(S_s,F_s) = \beta \cdot dist_s \cdot \left(\left(\frac{1}{1-S_s} \right)^{\lambda} - 1 \right) + \gamma \cdot dist_s \cdot F_s$$

Teilfunktionen der Haltepunkte:
Bezüglich der Haltepunkt-Agenten wäre eine Zerlegung der PLT in Teilfunktionen HTF_h^1 möglich, mit denen jeder Haltepunkt-Agent h die Verlustminuten berechnet, die für Reisende im Verkehrsnetz entstehen, deren Reise am entsprechenden Haltepunkt i *beginnt*:

$$HTF_h^1 = \sum_{h_2=1}^{stMax} \frac{w_{h,h_2}}{cycle} \sum_{t=0}^{cycle-1} \left(att_{h,h_2}^t(F,D,S) - ptt_{h,h_2} \right)$$

Nachteil einer solchen Berechnung ist, dass die Haltepunkt-Agenten jeweils Verlustminuten aggregieren, die zum großen Teil an ganz anderen „Stellen“, d.h. an anderen Haltepunkten entstehen. Die Zwischenergebnisse solcher partieller Evaluierungen bieten sich daher nicht als Basis für die Berechnung lokaler Güte-Koeffizienten an.
Stattdessen kann ein Haltepunkt-Agent mit seiner Teilfunktion auch die Verlustminuten aggregieren, die im *von ihm repräsentierten Haltepunkt* entstehen. Die Funktion $loss(t,h,h_1,h_2)$ zähle für einen Lösungsvorschlag die bei einer zum Zeitpunkt t am Haltepunkt h_1 begonnenen Reise zum Haltepunkt h_2 auftretenden Verlustminuten am Haltepunkt h. Dann lässt sich die entsprechende Teilfunktion HTF_h^2 schreiben als:

$$HTF_h^2 = \sum_{h_1=1}^{stMax} \sum_{h_2=1}^{stMax} \frac{w_{h_1,h_2}}{cycle} \sum_{t=0}^{cycle-1} loss(t,h,h_1,h_2)$$

Leider gilt $\sum_{h=1}^{stMax} HTF_h^2 \neq \sum_{h=1}^{stMax} HTF_h^1$: Die Summe der an den Haltepunkten auftretenden Verlustminuten entspricht nicht der Differenz zwischen den tatsächlichen und den bestmöglichen

[15] Das Framework sieht dazu die Spezifikation von zwei unterschiedlichen Agenten-Klassen in der XML-Datei vor.

Reisezeiten. Grund ist, dass kürzeste Reiseverbindungen möglicherweise über „Umwege“ führen, wenn auf der „streckenmäßig“ kürzesten Verbindung schlechte Anschlüsse vorherrschen. Auch wenn sich die an den Haltepunkten entstehenden (mit den Fahrgästen gewichteten) Verlustminuten als Zwischenergebnis für die Berechnung eines *Güte-Koeffizienten* gut eignen, müssen als *partielle Evaluationsfunktionen* die Teilfunktionen HTF^1 zusätzlich herangezogen werden.

1.2: Die Speicherung von Zwischenergebnissen der lokalen Bewertung
Für eine schnelle Ausführung der partiellen Bewertung berechnet jeder Haltepunkt-Agent h die theoretisch kürzest-mögliche Verbindung $ptt_{h,\bar{h}}$ zu allen anderen Haltepunkten $\bar{h}$ bereits zu Beginn der Optimierung im Voraus. Die Berechnung der tatsächlichen Fahrzeiten $att^t_{h,\bar{h}}$ für einen gegebenen Vorschlag und die unterschiedlichen Zeitpunkte t übernimmt die Methode *completeAndRepairProposal* des Umgebungs-Objekts. Die Resultate werden beim Vorschlag – quasi als Belegung einer weiteren Entscheidungsvariablen – eingetragen. Im Zuge dieser Berechnung trägt die Methode außerdem für jeden Haltepunkt h alle auftretenden Umsteige-Verluste, d.h. das Ergebnis der beschriebenen Funktion HTF^2_h, sowie die Anzahl aller am Haltepunkt umsteigenden Personen ein. Analog werden für jedes Paar benachbarter Shuttle[16] die auftretenden Umsteige-Verluste sowie die Anzahl der Umsteiger aggregiert und im Vorschlag eingetragen. Ebenfalls gespeichert wird die Anzahl der Fahrgäste jedes Shuttle.

Von einem Haltepunkt-Agent werden die in Tabelle 7.8 zusammengefassten Zwischenergebnisse berechnet. Die Teilfunktion *getLocalEvaluationValue* wird für Haltepunkt-Agenten so überschrieben, dass sie nur das Produkt des Zwischenergebnisses ZST_3 und des Gewichtungsfaktors α zurückliefert – damit ergeben die aufsummierten Rückgabewerte aller Haltepunkt-Agenten die mit dem Faktor gewichtete Kenngröße *PLT*. Die beiden anderen Zwischenergebnisse werden nur für die Berechnung der Güte-Koeffizienten verwendet. Die Shuttle-Agenten berechnen die in Tabelle 7.9 zusammengefassten Zwischenergebnisse.

Die Methode *getLocalEvaluationValue* eines Shuttle-Agent liefert die jeweiligen Anteile an den Kenngrößen *IC* und *TM* zurück. Dazu werden die Zwischenergebnisse ZSH_3 und ZSH_4 gewichtet mit den Faktoren β und γ aufsummiert.

Zwischenergebnis	**Beschreibung**	**Berechnung für den Agent des Haltepunkt j**
ZST_1	Umsteigeverluste	ZST_1^j ist bereits im Vorschlag eingetragen
ZST_2	Umsteigeranzahl	ZST_2^j ist bereits im Vorschlag eingetragen
ZST_3	Zeitverluste für am Haltepunkt beginnende Reisen	Aggregierte Differenzen zwischen bestmöglicher und tatsächlicher Reisezeit aller auftretenden Reisen zu anderen Haltepunkten
ZST_4	Knotenzeit des Haltepunkt	Sobald über 50% aller anliegenden Shuttle sich an eine gemeinsame Knotenzeit halten, wird diese als Zwischenergebnis eingetragen.
ZST_5	Zwischenergebnis für Einhaltung der Knotenzeit	Anteil anliegender Shuttle, der die Knotenzeit einhält.

Tabelle 7.8: *Zwischenergebnisse ZST der Haltepunkt-Agenten*

[16] Zwei Shuttle gelten dabei als benachbart, wenn sie einen Haltepunkt gemeinsam haben.

Zwischen-ergebnis	**Beschreibung**	**Berechnung für den Agent des Shuttle j**
$\text{ZSH}_{1.1}...\text{ZSH}_{1.n}$	Umsteigeverluste zu den n benachbarten Shuttle	$\text{ZSH}^j_{1.1}...\text{ZSH}^j_{1.n}$ sind bereits im Vorschlag eingetragen
$\text{ZSH}_{2.1}...\text{ZSH}_{2.n}$	Umsteigeranzahl zu den n benachbarten Shuttle	$\text{ZSH}^j_{2.1}...\text{ZSH}^j_{2.n}$ sind bereits im Vorschlag eingetragen
ZSH_3	Infrastrukturkosten	$\text{ZSH}^j_3 = dist_j \cdot \left(\left(\frac{1}{1-S_j}\right)^\lambda - 1\right)$
ZSH_4	Zugkilometer	$\text{ZSH}^j_4 = dist_j \cdot F_j$
ZSH_5	Anzahl Fahrgäste	ZSH^j_5 ist bereits im Vorschlag eingetragen

Tabelle 7.9: *Zwischenergebnisse ZSH der Shuttle-Agenten*

1.3: Die Berechnung von Güte-Koeffizienten
Ein Haltepunkt-Agent berechnet mittels der Methode *calculateRatios* die in Tabelle 7.10, und ein Shuttle-Agent die in Tabelle 7.11 zusammengestellten Koeffizienten.

Koeffizient	**Beschreibung**	**ZE.**
Reisezeit-verlust	Durchschnittliche Abweichung der momentanen Fahrzeit von der theoretisch bestmöglichen Fahrzeit für die Fahrt zu allen anderen Haltepunkten	ZST_3
Knotenzeit	Güte der momentanen Knotenzeit. Null- und Halbknoten werden hier besser bewertet als Richtungsknoten, ohne Standard-Begegnungszeit anliegender Shuttle werden die niedrigsten Werte werden vergeben.	ZST_4
Knotenzeit-Einhaltung	Das Zwischenergebnis ZST_5 wird als Koeffizient direkt übernommen.	ZST_5
Umsteige-verlust	Koeffizient für den durchschnittlichen Umsteigeverlust am Haltepunkt.	ZST_1, ZST_2

Tabelle 7.10: *Güte-Koeffizienten der Haltepunkt-Agenten. In der Spalte „ZE.“ sind die zur Berechnung der Koeffizienten herangezogenen Zwischenergebnisse eingetragen.*

1.4: Die Initialisierung von Startlösungen:
Startlösungen generieren Haltepunkt- und Shuttle-Agenten, indem sie „für sich günstige “ und ansonsten beliebige Fahrpläne generieren. Haltepunkt-Agenten setzen beispielsweise die Abfahrtszeiten anliegender Shuttle alle so, dass sie selbst einen Null- oder Halbknoten darstellen, Shuttle-Agenten bilden „Liniensegmente“ mit anderen Shuttle-Agenten, indem sie deren Abfahrtszeiten aufeinander abstimmen. Die übrigen Abfahrtszeiten werden einfach zufällig vergeben.

1.5: Die Integration lokaler Verbesserungsverfahren, Spezifikation von Situationen und Plänen
Anhand der beschriebenen Koeffizienten bestimmen Haltepunkt- und Shuttle-Agenten, in welcher Situation sie sich befinden. Die Situationen sind in Tabelle 7.12 zusammengefasst. Pläne zur Variation von Vorschlägen können Tabelle 7.13 entnommen werden. Für die Pläne sind jeweils die Situationen angegeben, in denen sie primär ausgeführt werden. Entsprechende Wahrscheinlichkeitswerte finden sich dann in der SP-Matrix, die hier nicht explizit aufgeführt ist.

Koeffizient(en)	**Beschreibung**	**ZE.**
Nächster-Anschluss	Zwei Koeffizienten für den jeweiligen *minimal* möglichen Umsteigeverlust am Start und am Zielbahnhof (d.h. die Wartezeit bis zur Abfahrt eines beliebigen anderen Shuttle)	-
Umsteige-verlust	Zwei Koeffizienten für den durchschnittlichen tatsächlich Umsteigeverlust von Fahrgästen an Start- und Zielhalt	ZSH_1, ZSH_2
Knotenzeit-Einhaltung	Das Zwischenergebnis zur Einhaltung der Knotenzeit wird als Koeffizient direkt übernommen.	ZSH_5
Kosten/ Nutzen	Koeffizient aus den Zwischenergebnissen zu Infrastrukturkosten, Zugkilometer und Fahrgastzahl	ZSH_3, ZSH_4, ZSH_5

Tabelle 7.11: *Güte-Koeffizienten der Shuttle-Agenten. In der Spalte „ZE.“ sind die zur Berechnung der Koeffizienten herangezogenen Zwischenergebnisse eingetragen.*

Situation	**Beschreibung**
NetDelay (H)	Zwischen tatsächlichen und minimal möglichen Verbindungszeiten zu den übrigen Haltepunkten bestehen große Differenzen.
BadDDT (H)	Der Haltepunkt hat gar keine oder eine ungünstige Knotenzeit (Default Departure Time). Günstige Knotenzeiten liegen auf (Null-knoten) oder gegenüber der Symmetrieachse (Halbknoten).
TranferLoss (H)	Die durchschnittliche Umsteige-Wartezeit aller am Haltepunkt umsteigenden Fahrgäste ist groß.
Satisfied (H/S)	Der Haltepunkt- oder Shuttle-Agent ist zufrieden.
NoConnection1, No-Connection2 (S)	An Start- bzw. Zielhalt besteht kein Anschluss. Eine mögliche spätere Durchbindung des Shuttle mit anderen Shuttle zu einer Linie wird daher nicht ohne Wartezeiten am Haltepunkt erfolgen können.
BadConnection1, BadConnection2 (S)	An Start- bzw. Zielhalt treten lange durchschnittliche Umsteige-Wartezeiten für Fahrgäste dieses Shuttle auf.
DDTDeviation1, DDTDeviation2 (S)	Die Abfahrts- bzw. Ankunftszeit stimmt nicht mit der Knotenzeit (Default Departure Time) des entsprechenden Haltepunkts überein.
BadCostValueRatio (S)	In Relation zur Fahrgastzahl sind die Infrastrukturmaßnahmen zu teuer oder die Bedienfrequenz ist zu hoch.

Tabelle 7.12: *Situationen der ITF-Agenten. Mit (H) und (S) wird angedeutet, ob die Situation für (H)altepunkt- oder (S)huttle-Agenten in Frage kommt*

1.6: Die Rekombination von Vorschlägen

Die Rekombination von Vorschlägen basiert auf ähnlichen Überlegungen wie beim ersten Anwendungsbeispiel: Zusammengehörende Lösungsteile sollen möglichst gemeinsam „vererbt“ werden. Wie bei der Standortplanung können dabei entsprechende Teillösungen aus benachbarten Objekten gebildet werden. Allerdings haben die den Agenten zugeordneten Objekte nun festen Raumbezug, da sich die Koordinaten der Haltepunkte (und die Topologie des Netzes) nicht ändern. Die Koordinaten der Haltepunkte und Topologie des Netzes können zur Definition von Nachbarschaften herangezogen werden. Bei der Standard-Rekombination übernimmt ein Haltepunkt-Agent bzw. ein Shuttle-Agent zunächst die Konfigurationen aller (Shuttle-)Objekte aus dem ersten, insgesamt gut bewerteten Vorschlag – mit dem der Agent selbst jedoch nicht zufrieden war –

Plan	Beschreibung	z.B. in Situation
NewDDT	Ein Haltepunkt nimmt eine neue Knotenzeit an (und passt die Abfahrten aller anliegenden Shuttle an)	BadDDT (H)
AdaptDDT-ToNeighbour	Ein Haltepunkt richtet seine Knotenzeit nach der eines benachbarten Bahnhofs und setzt die Abfahrten anliegender Shuttle entsprechend	BadDDT (H)
Adapt-Departures-ToDDT	Ein Haltepunkt setzt die Abfahrtszeiten anliegender Shuttle gemäß seiner momentanen Knotenzeit.	TransferLoss (H)
BuildLine	Eine „Linie“ zwischen zwei Haltepunkten bestehend aus mehreren Shuttle aufeinander abgestimmter Ankunfts- und Abfahrtszeiten wird gebildet.	NetDelay (H)
Adapt-Departure (-Arrival)	Die Abfahrtszeit (Ankunftszeit) eines Shuttle wird anhand der Ankunftszeiten (Abfahrtszeiten) anderer Shuttle an einem Haltepunkt neu bestimmt	BadConnection1 (S)
Adapt-Investment	Die Infrastrukturmaßnahmen eines Shuttle werden intensiviert bzw. zurückgenommen.	BadCostValueRatio (S)
Adapt-Frequency	Die Bedienfrequenz eines Shuttle wird erhöht bzw. verringert.	BadCostValueRatio (S)

Tabelle 7.13: *Situationen der ITF-Agenten*

in einen neuen Vorschlag. Anschließend ersetzt er die Konfiguration aller Shuttle, deren Start- und Endhaltepunkte entweder eine bestimmte Entfernung nicht überschreiten oder aufgrund der Topologie als benachbart gelten können, durch die des zweiten Vorschlags (mit dem der Agent zufrieden war). Bei einer Rekombination auf Grundlage einer Koalition werden stattdessen alle Konfigurationen von Koalitionsmitgliedern aus dem zweiten Vorschlag übernommen.

1.7: Die Auswahl von Kandidaten für die Bildung einer Koalition
Koalitionen dürfen unter Haltepunkt- und Shuttle-Agenten gebildet werden, sofern diese *benachbart* sind. Zwei Haltepunkte gelten dabei als benachbart, wenn zwischen ihnen ein Shuttle verkehrt, zwei Shuttle gelten als benachbart, wenn sie einen gemeinsamen Haltepunkt bedienen, und ein Shuttle ist mit einem Haltepunkt benachbart, wenn das Shuttle den Haltepunkt bedient – hier wurde die Nachbarschaft also auf Grundlage der Topologie des Netzwerks definiert.

Schritt 2: Die Implementierung der Umgebung

2.1: Implementierung der Vervollständigungs- und Reparaturfunktion
Die Vervollständigungs- und Reparaturfunktion *completeAndRepairProposal* wird verwendet, um die Routen für die jeweils kürzesten Verbindungen zwischen allen Haltepunkten und damit die Auslastung der Shuttle und die Umsteigehäufigkeiten sowie auftretende Wartezeiten zwischen Paaren von Shuttle und an Haltepunkten für einen gegebenen Fahrplan zu bestimmen (vgl. Schritt 1.2). Die Routen werden dabei jeweils für mehrere „Ankunftszeiten“ am Ausgangs-Haltepunkt berechnet (*Modell der zufälligen Ankunft am Starthaltepunkt*).

2.2: Implementierung der Restfunktion für die Bewertung von Lösungsvorschlägen
Eine Restfunktion wird nicht benötigt, da die Zielfunktion vollständig durch Aggregation der Teilfunktionen evaluiert werden kann.

Schritt 3: Implementierung des Konferenzmonitors

3.1: Graphische Darstellung von Lösungsvorschlägen

Die Darstellung eines Lösungsvorschlags durch die Methode *drawProposal* erfolgt in einer erweiterten Netzgrafik. Angezeigt werden die Haltepunkte als Kreise und die Shuttle als Linien. Die Verteilung der Fahrgastströme wird durch die Dicke der Linien verdeutlicht, die Fahrzeit und mögliche Beschleunigungen eines Shuttle (in Minuten) sind in der Mitte der Linie angegeben. Die Abfahrtszeiten der Shuttle werden an den entsprechenden Enden angegeben, Ankunftszeiten ergeben sich durch die Symmetrieachse (hier die Minute 0). Die Umsteige-Relationen werden durch Splines in den Haltepunkten angezeigt. Die Dicke der Splines deutet wiederum die Anzahl der Fahrgäste an, die diese Umsteigebeziehung nutzen. Treten im Durchschnitt weniger als 5 Umsteige-Warteminuten auf, so wird der Spline grün eingefärbt. Gelbe Splines kennzeichnen Umsteigebeziehungen mit durchschnittlich 5 bis 10 Minuten Wartezeit. Ungünstigere Umsteigebeziehungen werden durch rote Splines hervorgehoben. Abbildung 7.5 zeigt einen Ausschnitt der Darstellung eines Lösungsvorschlags für ein ITF-Optimierungsproblem. Neben den Knotenzeiten und Abfahrtszeiten der Haltepunkte und Shuttle sind u.a. auch die momentanen Zufriedenheiten der entsprechenden Agenten angegeben.

3.2: Textuelle Ausgabe von Lösungsvorschlägen

Die textuelle Ausgabe eines Lösungsvorschlags beinhaltet die Belegung der Entscheidungsvariablen aller Shuttle sowie die resultierenden Knotenzeiten der Haltepunkte.

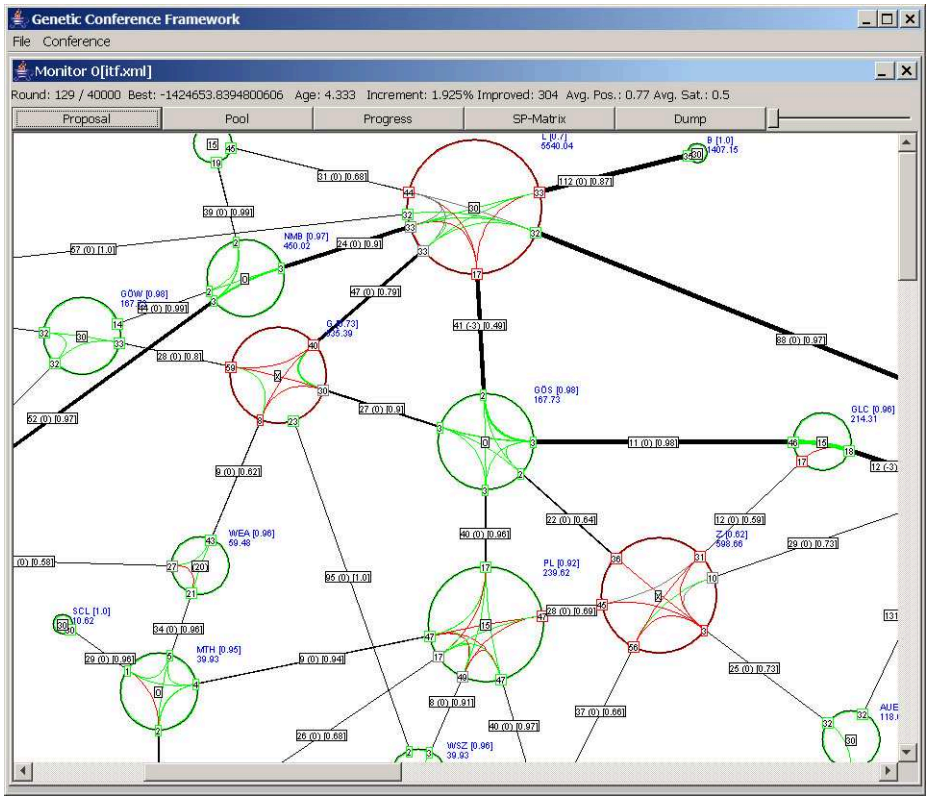


Abbildung 7.5: *Konferenzmonitor für Lösungsvorschläge des ITF-Optimierungsproblems*

7.3.5 Einsatz der Genetischen Konferenz bei der Optimierung von ITF als Modul des Expertensystems *OptiTakt*

Die beschriebene Anpassung des Framework zur Optimierung von ITF wurde durch entsprechende Implementierung der abstrakten Java-Klassen durchgeführt. Daneben wurde das auf dieser Implementierung beruhende Modul *OptiTaktGC* für das Softwaresystem *OptiTakt* entwickelt. *OptiTakt* stellt die unter 3.3.2 beschriebenen Funktionen zur Erstellung und Analyse der Fahrpläne „von Hand“ zur Verfügung, das Modul *OptiTaktGC* erweitert diese nun um die *automatisierte Grobplanung* von ITF. *OptiTaktGC* beruht zwar weitgehend auf der beschriebenen Anpassung des Framework, es ergeben sich jedoch zwei Unterschiede: Erstens werden die Funktionen zur Darstellung von Lösungsvorschlägen vom Basissystem übernommen, das Modul *OptiTaktGC* verwendet daher keinen eigenen Konferenz-Monitor mit graphischer Ausgabe. Zweitens kann der Anwender im Kernsystem Abfahrtszeiten bestimmter Shuttle fixieren und Beschleunigungen verbieten, also Entscheidungsvariablen im Vorfeld fest belegen. So können beispielsweise mit *OptiTaktGC* Abfahrtszeiten für den Regionalverkehr bei festgelegten Zeiten des Fernverkehrs optimiert werden. Eine solche mehrstufige Planung hat sich in der Praxis als sinnvoll erwiesen.

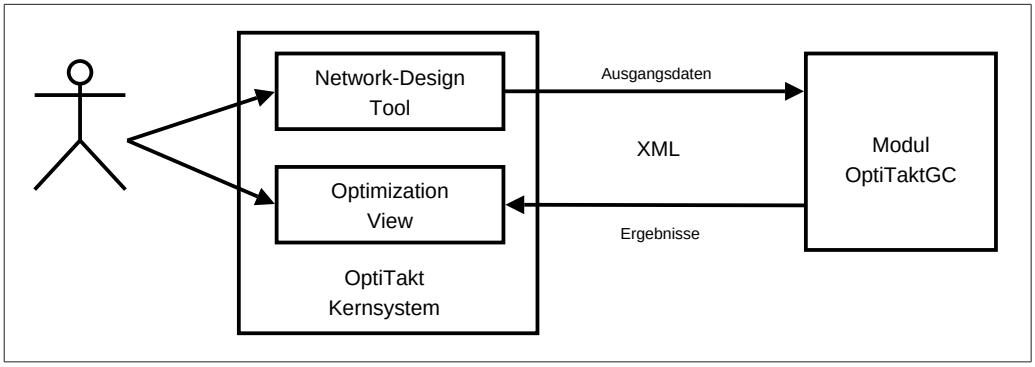


Abbildung 7.6: *Kommunikation zwischen OptiTaktGC-Modul und Kernsystem*

Die Kommunikation zwischen *OptiTaktGC* und dem *OptiTakt*-Kernsystem erfolgt über eine XML-Schnittstelle. Die notwendigen Ausgangsdaten für die Optimierung, d.h. die Konfiguration des Netzes, werden vom Anwender über die CAD-Funktionen des Basissystems generiert und können dort auch als XML-Dateien persistent gespeichert werden. Vor dem Start eines Optimierungslaufs werden diese Daten in einen XML-Datenstrom kodiert, der an das *OptiTaktGC*-Modul übertragen wird. Zum Start eines Optimierungslaufs werden zusätzlich benötigte Optimierparameter wie z.B. die maximale Generationenzahl übertragen. Im Verlauf der Optimierung liefert das *OptiTaktGC*-Modul in regelmäßigen Abständen Informationen über den Stand der Optimierung. Dabei wird unter anderem der momentan beste Fahrplan über die XML-Schnittstelle zurückgeliefert, damit dieser als eigenständige Sicht im Kernsystem angezeigt werden kann. Die graphische Darstellung erfolgt ähnlich wie im Konferenzmonitor des angepassten Framework.

Das System *OptiTakt* wurde in diversen Beratungsprojekten erfolgreich eingesetzt, eine Aufstellung der Projekte findet sich beispielsweise in [HGH05]. Mit der Erweiterung zur automatisierten Optimierung können realistische Netze mit ca. 60 Knoten problemlos bearbeitet werden.

7.3.6 Ein Vergleich mit anderen Ansätzen der Fahrplanoptimierung

Es existieren natürlich auch andere Ansätze zur Optimierung von Taktfahrplänen. Einige sollen an dieser Stelle erwähnt und die wesentlichen Unterschiede aufgezeigt werden.

Modellierung als Quasi-QAP: Domschke [Dom89] schlägt heuristische Lösungsmethoden sowie einen Branch-And-Bound-Algorithmus für das Problem der Fahrplangestaltung im öffentlichen (Personen-) Nahverkehr vor. Sein Modell sieht vordefinierte Linien und Taktzeiten vor, die Anzahlen der an den Verknüpfungspunkten umsteigenden Passagiere sind gegeben. Der Ansatz weist Ähnlichkeiten zu einem Quadratischen Zuordnungsproblem *(Quadratic Assignment Problem, QAP)* auf.

Im Unterschied dazu wird hier ein Ansatz verfolgt, bei dem die Durchbindung der Linien nicht feststeht (Shuttle-Modell), die Taktzeiten – wenn auch nur in Abstimmung mit einem Grundtakt – angepasst werden können und die Verteilung der Kundenströme erst anhand der jeweils schnellsten Reiserouten erfolgt.

Modellierung als PESP: Liebchen et al. [LM04, LPW04] und Peeters [Pee03] verwenden Ansätze, die auf einer Modellierung als Periodic Event Scheduling Problem (PESP) beruhen. Ankünfte und Abfahrten von Linien an Haltepunkten werden als periodische Ereignisse (*events*) modelliert. Für die (zeitlichen) Beziehungen unter den Ereignissen werden Bedingungen (*constraints*) formuliert, darüber werden z.B. Umsteigebeziehungen modelliert. Wiederum ist die Durchbindung der Linien im Vorfeld gegeben.

Der hier verfolgte Ansatz besteht dagegen in der Grobplanung (symmetrischer) Integraler Taktfahrpläne, d.h. der Charakterisierung sinnvoller Taktknoten und entsprechender Knotenzeiten sowie notwendiger Ausbaumaßnahmen. Das PESP-Modell dagegen eignet sich weniger für die Planung symmetrischer Fahrpläne [Lie04].

Optimierung mit Genetischen Algorithmen: Voget [Vog95] verwendet einen Genetischen Algorithmus, um zunächst die Abfahrtszeiten von Fahrplänen in Personenverkehr zur optimieren. In einem zweiten Schritt lässt er auch Veränderungen der Fahrzeiten zu und sucht nach der kostengünstigen Realisierung eines Integralen Taktfahrplans. Er kombiniert den GA mit einem Fuzzy-Regler, der bei der Selektion für eine möglichst gute Repräsentation der Pareto-Front sorgt.
Kolonko und Engelhard-Funke [KE02] beschreiben das Programm HiTT, das ebenfalls genetische Algorithmen zur Optimierung von Fahrplänen verwendet. Wiederum wird von einem gegebenen Linienplan ausgegangen. Neben Integralen Taktfahrplänen können auch andere Fahrpläne betrachtet werden. Im Vordergrund steht eine Kosten-Nutzen-Analyse der Investitionsmaßnahmen und der auftretenden Wartezeiten. Ebenfalls betrachtet wird die *Robustheit* des Fahrplans, d.h. seine Anfälligkeit für Verspätungen.

Sowohl der Ansatz von Voget, als auch das System HiTT weist durch den heuristischen Charakter und die Verwendung Genetischer Algorithmen Ähnlichkeiten zum hier verfolgten Ansatz auf. Allerdings wird von festen Durchbindungen der Linien ausgegangen. Unterschiede zum HiTT-System ergeben sich des Weiteren in der Komplexität: An der Umsetzung von HiTT waren einige Diplomanden und Doktoranden über einen längeren Zeitraum beschäftigt. Beim hier verfolgten Ansatz werden des Weiteren *ausschließlich* ITF betrachtet, im Fokus steht die *Grobplanung* der Taktknoten.

7.3.7 Performanzvergleich der Anpassung mit Simulationen der Basisverfahren MAS und GA

Zuletzt soll wiederum ein Performanzvergleich der Anpassung des Framework mit Simulationen eines Greedy-MAS und eines GA durchgeführt werden. Dazu wird als Problemexemplar ein kleines Verkehrsnetz im Raum Nordbayern, Hessen und Thüringen mit 25 Haltepunkten aus einem aktuellen Planungsprojekt herangezogen. Es wurden jeweils 10 Optimierungsläufe mit der Anpassung der Genetischen Konferenz und den beiden Varianten durchgeführt, die ein Greedy-MAS und einen GA simulieren. Abbildung 7.7 zeigt die gemittelten Fitness-Werte der jeweils besten gefundenen Lösungsvorschläge.

Wie bereits bei der Standortplanung findet die MAS-Variante – die im Unterschied zur Genetischen Konferenz nur auf einem Lösungsvorschlag arbeitet – zwar zunächst schneller Verbesserungen als das verknüpfte Verfahren, stagniert dann aber oft in schlechteren lokalen Optima. Die GA-Variante ohne Delegation von Lösungsvorschlägen und Situations-spezifischer Operatorauswahl zeigt deutlich langsamere Verbesserungen.

Das verknüpfte Verfahren liefert ab ca. 500 ausgewerteten Lösungsvorschlägen im Durchschnitt die besten Ergebnisse.[17] In den einzelnen Optimierungsläufen können im Unterschied zur MAS-Variante meist auch noch im späteren Verlauf der Optimierung Verbesserungen erzielt werden.

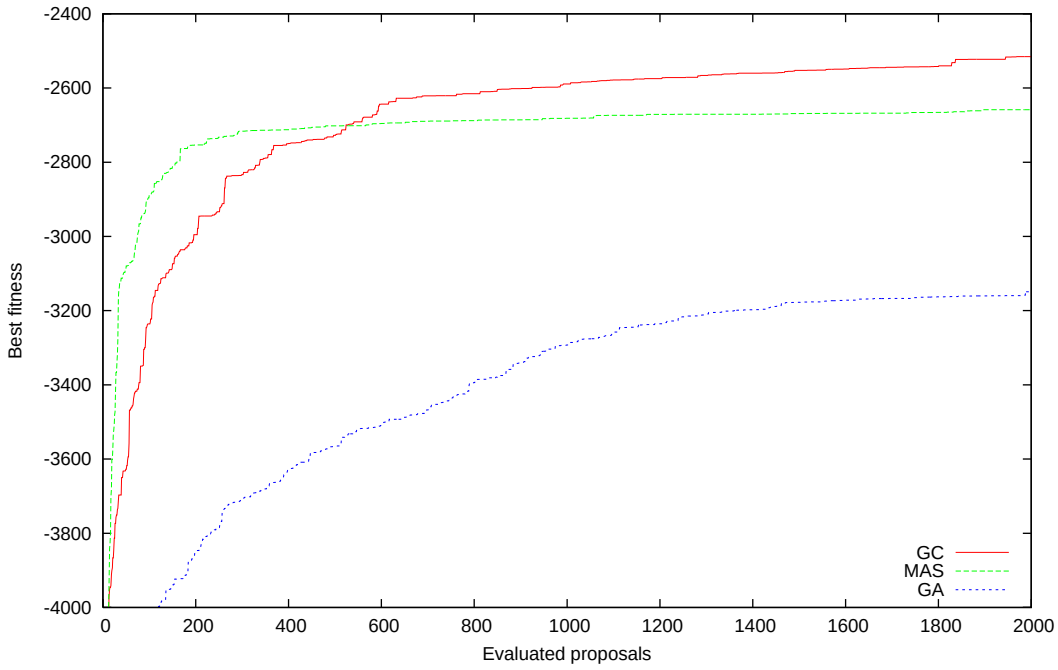


Abbildung 7.7: *Performanzvergleich der Genetischen Konferenz (GC) mit Simulationen eines Genetischen Algorithmus (GA) und eines Greedy-MAS (MAS) bei der Optimierung eines ITF*

[17] Aufgrund der relativ hohen Varianz unter den Ergebnissen einzelner Optimierungsläufe ist der Unterschied der Mittelwerte der besten gefundenen Lösungen nach 2000 ausgewerteten Vorschlägen zwischen dem verknüpften Verfahren und der MAS-Variante allerdings erst ab einer Irrtumswahrscheinlichkeit von $\alpha = 2,7\%$ signifikant (*Student'scher t-Test*).

7.4 Anwendungsbeispiel III: Anpassung für die Zuteilung zu den Rotationskursen

Letztes Anwendungsbeispiel für die Anpassung des Framework stellt das Problem der Verteilung von Praktikumsplätzen bei der so genannten *Klinischen Rotation* der Tierärztlichen Fakultät der LMU München dar (vgl. 3.4). Im Unterschied zu den anderen Anwendungsfällen handelt es sich hier um ein Problem ohne Raumbezug, das jedoch ebenfalls die Möglichkeit zur partiellen Bewertung von Lösungsvorschlägen bietet.

7.4.1 Vorüberlegungen zur Anpassung des Framework

Modellierung der Nebenbedingungen
Beim Problem der Zuteilung zu den Rotationskursen ist aufgrund der vielen, stark einschränkenden Nebenbedingungen allein schon das Auffinden einer *gültigen* Lösung ein schwieriges Problem. Um den Algorithmus bei der Suche gültiger Lösungen zu unterstützen, sollen – genauso wie bei der beschriebenen Zuteilung von Hand – ungültige Lösungen nicht generell verworfen werden, sondern stattdessen zunächst zugelassen und möglichst im Laufe des Verfahrens in tatsächlich gültige Lösungen umgewandelt werden. Daher wird insbesondere die Nebenbedingung zunächst nicht als *zwingend* betrachtet, dass einem Studierenden in allen Bereichen, in denen er ein Praktikum besuchen muss, auch tatsächlich einen Praktikumsplatz zugewiesen wird. Die Übertretung dieser Nebenbedingungen (*soft constraints*) geht stattdessen durch *Strafkosten* in die Bewertung des Vorschlags ein (vgl. auch Abschnitt 4.3.4). Der mehrmalige Besuch desselben Praktikums, der gleichzeitige Besuchs mehrerer Praktika und die Zuteilung zu bereits voll belegten Praktika wird dagegen unterbunden, indem die Initialisierung- und Variationsoperatoren entsprechend implementiert werden (*hard constraints*).

Die Integration mehrerer Ziele
Ein guter (bezüglich der *hard constraints* gültiger) Lösungsvorschlag soll mehrere unterschiedliche Bedingungen erfüllen:

(1) Der Vorschlag soll bezüglich der zugeteilten Praktika *gültig* sein (Einhaltung der *soft constraints*).

(2) Die *Präferenzen bezüglich des Wahlpraktikums* sollen berücksichtigt werden.

(3) Die *Präferenzen bezüglich der Verteilung der Pausen* sollen berücksichtigt werden.

(4) Die Kurse sollen *gleichmäßig ausgelastet* sein.

Aufgrund der von (1) nach (4) stark abfallenden Bedeutsamkeit der Bedingungen bietet sich eine Bearbeitung an, bei der die Ziele auch in dieser Reihenfolge Beachtung finden: Bei der Bewertung zweier gegen *soft constraints* verstoßender Lösungsvorschläge muss zunächst die Anzahl der übertretenen Nebenbedingungen betrachtet werden. Die gleichmäßige Auslastung der Kurse kann nur dann eine Rolle spielen, wenn beide Vorschläge bezüglich der Bedingung (1) die gleiche Bewertung erhalten. Um dies zu gewährleisten, wird ein Ansatz mit gewichteter Aggregation der Teilzielfunktionen für die Bedingungen (1) - (4) verfolgt, wobei die Gewichte stark abgestuft werden.

7.4.2 Parameter und Entscheidungsvariablen

Wie bei den vorausgegangenen Anwendungsbeispielen werden für Parameter und Hilfsfunktionen Bezeichnungen verwendet, die mit Kleinbuchstaben beginnen. Entscheidungsvariablen und Komponenten der Zielfunktion werden in GROSSBUCHSTABEN notiert. Erläuterungen zur konkreten Verwendung bestimmter Modellparameter werden in *Schrägschrift* gehalten.

Parameter

- Die Anzahl der Studierenden (*students*) betrage *sMax*, die Anzahl der Praktika (*courses*) sei *cMax*, die Anzahl der Wochen (*weeks*), in denen Praktika stattfinden, sei *wMax* und die Anzahl der Bereiche (*fields*), denen die Praktika zugeordnet sind, betrage *fMax*.

- Die Länge (in Wochen) eines Praktikums p, das in der Woche w beginnt, sei mit $length_{c,w}$ bezeichnet.
  Über die unterschiedlichen Längen eines gleichen – jedoch in verschiedenen Wochen beginnenden – Praktikums werden Praktikumspausen über die Weihnachtszeit etc. modelliert.

- Das maximale Platzangebot für jedes Praktikum c, das in der Woche w beginnt, sei $m_{c,w}$. Neben den unterschiedlichen maximalen Teilnehmerzahlen verschiedener Blockpraktika können über das maximale Platzangebot auch deren tatsächliche Starttermine modelliert werden: $m_{c,w} = 0$ zeigt an, dass *kein* Praktikum c angeboten wird, das in der Woche w beginnt. Um eine gleichmäßige Auslastung der Praktika gewährleisten zu können, wird verlangt, dass das maximale Platzangebot $m_{c,w}$ mindestens so hoch ist, wie der *durchschnittliche* Platzbedarf in den Praktika c über die Wochen w hinweg.

- Jedes Praktikum c wird mit dem Parameter $belongsTo_{c,f}$ einem Bereiche f zugeordnet:

$$belongsTo_{c,f} = \begin{cases} 1 & \text{Das Praktikum c ist dem Bereich } f \text{ zugeordnet.} \\ 0 & \text{sonst.} \end{cases}$$

  Ein Praktikum ist immer genau einem Bereich zugeordnet: $\forall c : \sum_{f=1}^{fMax} belongsTo_{c,f} = 1$.
  Im konkreten Anwendungsfall ergeben sich sechs Bereiche: Ein Bereich – im Folgenden „Bereich GSF“ genannt – umfasst die Praktika „Geflügel“,„Schwein“ und „Fische und Reptilien“, die anderen fünf enthalten jeweils nur ein Praktikum.

- Mit dem Parameter $n_{s,f}$ wird angezeigt, ob der Student s ein Praktikum des Bereichs f belegen muss ($n_{s,f} = 1$) oder nicht ($n_{s,f} = 0$).
  An der Klinischen Rotation nehmen auch so genannte „Querläufer“ teil, die gewisse Praktika bereits im Vorjahr absolviert haben oder erst im nächsten Jahr absolvieren müssen. Diese Besonderheiten können mit den Parametern $n_{s,f}$ modelliert werden.

- Mit dem Parameter $prio_{s,c}$ werden die Prioritäten eines Studierenden s bezüglich des zu besuchenden Praktikums c angegeben:

$$prio_{s,c} = \begin{cases} 0 & \text{Der Student hat das Praktikum } c \text{ mit erster Priorität gewählt.} \\ -1 & \text{Der Student hat das Praktikum } c \text{ mit zweiter Priorität gewählt.} \\ -2 & \text{sonst.} \end{cases}$$

Ein Student darf Priorität 1 und 2 pro Bereich nur einmal vergeben. Alle möglichen Prioritäten sollen auch tatsächlich verteilt werden. Wird in einem Bereich f beispielsweise *nur ein* Praktikum c angeboten, so gilt daher automatisch $\forall s : n_{s,f} \cdot prio_{s,p} = 0$.
Mit den Teilnahmewünschen können im konkreten Anwendungsfall die angesprochenen Präferenzen der Studierenden im Bereich GSF modelliert werden. In den übrigen Bereichen wird jeweils nur ein Praktikum angeboten.

- Die Präferenz der **Pausenverteilung** eines Studierenden s wird mit $pause_s$ bezeichnet ($pause_s \in \{-1, 0, 1\}$). Es soll gelten:

$$pause_s = \begin{cases} -1 & \text{Der Studierende } s \text{ wünscht sich kurze, verteilte Pausen.} \\ 1 & \text{Der Studierende } s \text{ wünscht sich möglichst zusammenhängende Pausen.} \\ 0 & \text{Der Studierende } s \text{ hat keine Präferenzen bezüglich der Verteilung.} \end{cases}$$

- Im voraus bereits anderweitig belegte Wochen w eines Studierenden s werden mit den binären Parametern $occupied_{s,w} \in \{0, 1\}$ erfasst. Dabei gilt: $occupied_{s,w} = 1 \Leftrightarrow$ Die Woche w ist beim Studierenden s bereits anderweitig belegt.
Im konkreten Anwendungsfall können damit die Termine der individuell vom Studierenden zu vereinbarenden Schlachthof- und Hygienepraktika modelliert werden.

Entscheidungsvariablen

Gesucht ist eine gültige und möglichst den Wünschen der Studierenden entsprechende Zuordnung (*assignment*) zu den Praktika in den unterschiedlichen Wochen. Dafür werden die binären Entscheidungsvariablen $A_{s,w,c}$ verwendet:

$$A_{s,w,c} = \begin{cases} 1 & \text{Der Studierende s beginnt in der Woche w mit dem Praktikum c.} \\ 0 & \text{Der Studierende s beginnt in der Woche w nicht mit dem Praktikum c.} \end{cases}$$

Hilfsvariablen und Hilfsfunktionen

Zur übersichtlicheren Darstellung der Zielfunktion und ihrer Nebenbedingungen sollen zunächst noch folgende Hilfsfunktionen eingeführt werden wobei auf die explizite Darstellung ihrer Berechnung an dieser Stelle verzichtet werden soll:

- Zur Auswertung der Einhaltung der Wünsche eines Studierenden s zur Verteilung der Pausen wird die Hilfsfunktion $pLength_s(A)$ eingeführt, die Dauer (in Wochen) der längsten zusammenhängenden Pause berechnet.

- Für die Auswertung der gleichmäßigen Auslastung der Praktika wird eine weitere Hilfsfunktion verwendet. Die Hilfsfunktion $diffAvg_{c,w}(A)$ messe die Differenz zwischen der Auslastung des Praktikums c in der Woche w von der durchschnittlichen Auslastung aller Praktika im entsprechenden Bereich über den Gesamtzeitraum. Auf die explizite Darstellung der Berechnung dieser Hilfsfunktion soll an dieser Stelle verzichtet werden.

7.4.3 Modellierung der Zielfunktion

Zu minimieren sind Strafkosten PLC für Studierende, denen in bestimmten Bereichen kein Praktikum zugewiesen werden konnte (*lack of courses*) sowie Strafkosten PU für ungleichmäßig (*uneven*) ausgelastete Praktika. Des Weiteren sollen die Wünsche der Studierenden beachtet werden, daher werden weitere Strafkosten PC und PP für Nichtbeachtung der Präferenzen zu Praktika und der Pausenverteilung vergeben. Die Zielfunktion $f(A)$ lässt sich dann formulieren als:

$$f(A) = \alpha \cdot PLC(B) + \beta \cdot PC(B) + \gamma \cdot PP(B) + \delta \cdot PU(A)$$

Die Parameter $\alpha, \beta, \gamma, \delta$ dienen zur Gewichtung. Als gültig werden Lösungen betrachtet, die die folgenden Nebenbedingungen einhalten:

- Ein Praktikum kann höchstens einmal absolviert werden: $\forall s, \forall c : \sum_{w=1}^{wMax} A_{s,w,c} \leq 1$
- In jeder Woche kann höchstens ein Praktikum gleichzeitig besucht werden, und zwar nur dann, wenn die Woche nicht anderweitig belegt ist: $\forall s, \forall w : busy_{s,w}(A) + occupied_{s,w} \leq 1$
- Die maximalen Platzangebote in den Praktika dürfen nicht überschritten werden: $\forall w, \forall c : \sum_{s=1}^{sMax} A_{s,w,c} \leq m_{c,w}$
- Die Anzahl der Praktika, die ein Studierender in einem Bereich belegt, ist 0, wenn der Studierende kein Praktikum aus dem Bereich belegen muss. Ansonsten kann sie nur 0 oder 1 betragen: $\forall s, \forall f : \sum_{w=1}^{wMax} \sum_{c=1}^{cMax} belongsTo_{c,f} \cdot A_{s,w,c} \leq n_{s,f}$

Die Strafkosten $PLC(A)$, $PC(A)$ und $PP(A)$ lassen sich folgendermaßen berechnen:

$$PLC(A) = \sum_{s=1}^{sMax} \sum_{f=1}^{fMax} \left(n_{s,f} - \sum_{w=1}^{wMax} \sum_{c=1}^{cMax} belongsTo_{c,f} \cdot A_{s,w,c} \right)$$

$$PC(B) = \sum_{s=1}^{sMax} \sum_{w=1}^{wMax} \sum_{c=1}^{cMax} prio_{s,c} \cdot A_{s,w,c} \qquad PP(A) = \sum_{s=1}^{sMax} \left(pLength_s(A) \cdot pause_s \right)$$

Keine Strafkosten PP werden für einen Studierenden verteilt, der keine Präferenz zur Verteilung der Pausen angemeldet hat. Bei Studierenden, die möglichst zusammenhängende Pausen wünschen, sinken die Strafkosten mit der Länge der längsten Pause, bei den übrigen steigen sie.[18]

Die Berechnung der Strafkosten $PU(A)$ könnte beispielsweise folgendermaßen erfolgen:

$$PU(A) = \sum_{w=1}^{wMax} \sum_{c=1}^{cMax} \left(diffAvg_{c,w} \right)^{\lambda}$$

Über den zusätzlichen Parameter λ könnte dann die Höhe der Abweichungen von der durchschnittlichen Auslastung unterschiedlich stark gewichtet werden. Es wird jedoch eine alternative Berechnung verwendet, die im nächsten Abschnitt eingeführt wird.

[18] Theoretisch sind so auch insgesamt negative Strafkosten möglich. Dies kann durch eine Korrektur-Konstante verhindert werden, deren Berechnung hier jedoch nicht ausgeführt werden soll.

7.4.4 Anpassung des Framework nach dem Vorgehensmodell

Schritt 1: Implementierung der Agenten und Spezifikation ihres Verhaltens

1.1: Die Identifikation der Agenten:
Als Objekte, deren Zuordnung Gegenstand der Optimierung ist, lassen sich die Studierenden identifizieren. Die Agenten repräsentieren daher jeweils einen dieser Studierenden und verwalten die Entscheidungsvariablen $A_{s,w,c}$.[19] Die Zielfunktion lässt sich fast vollständig in Teilfunktionen zerlegen, die jeweils einem der Agenten zugeordnet werden können:

$$f(A) = \sum_{s=1}^{sMax} (\alpha \cdot PLC_s(A) + \beta \cdot PC_s(A) + \gamma \cdot PP_s(A)) + \delta \cdot PU(A)$$

Unter $PLC_s(A)$, $PC_s(A)$ und $PP_s(A)$ sollen dabei die entsprechenden Summanden bei der Berechnung von $PLC(A)$, $PC(A)$ und $PP(A)$ verstanden werden, die den Strafkosten-Anteil des Studierenden s ausmachen. Zunächst nicht den Studierenden zuordnen lassen sich die Strafkosten $PU(A)$. Eine Berechnung dieser Kosten in der in Abschnitt 7.4.3 angegebenen Form durch die Restfunktion wäre möglich.
Stattdessen wird hier der Ansatz verfolgt, der die Implementierung einer Restfunktion überflüssig macht: Die Strafkosten werden auf die Agenten umgelegt, deren Studierende einem *überdurchschnittlich* ausgelasteten Praktikum zugewiesen wurden.[20] Die Höhe der Abweichung vom Durchschnitt kann durch den Exponenten λ unterschiedlich stark gewichtet werden.

$$\acute{PU}(A) = \sum_{s=1}^{sMax} \acute{PU}_s(A); \qquad \acute{PU}_s(A) = \sum_{w=1}^{wMax} \sum_{c=1}^{cMax} \acute{PU}_{s,w,c}(A)$$

$$\acute{PU}_{s,w,c}(A) = \begin{cases} A_{s,w,c} \cdot \mathit{diffAvg}_{c,w}(A)^{\lambda} & \text{falls } \mathit{diffAvg}_{c,w}(A) > 0 \\ 0 & \text{sonst} \end{cases}$$

1.2: Die Speicherung von Zwischenergebnissen der lokalen Bewertung
Tabelle 7.14 fasst die Zwischenergebnisse der partiellen Evaluation zusammen. Zusätzlich wird die Methode *getLocalEvaluationValue* überschrieben, um die Bestrafung der Nichteinhaltung von Nebenbedingungen (z_1) noch stärker zu gewichten.

1.3: Die Berechnung von Güte-Koeffizienten
Für jedes der genannten Zwischenergebnisse berechnet der Agent einen Güte-Koeffizienten, indem jeweils eine Basiszahl zwischen 0 und 1 mit dem Zwischenergebnis potenziert wird. Um die Nebenbedingungen (*soft constraints*) wiederum stärker zu gewichten, wird für den ersten Güte-Koeffizienten eine deutlich kleinere Basis gewählt als für die anderen Koeffizienten. Da der lokale Zufriedenheitswert automatisch aus dem Produkt der Güte-Koeffizienten berechnet wird (vgl. Abschnitt 7.1, Schritt 1.3), ist damit auch gewährleistet, dass vorrangig die Entscheidungsvariablen variiert werden, deren Belegungen gegen die *soft constraints* verstoßen.

[19] Allerdings wird bei der Implementierung statt den $wMax \cdot cMax$ binären Entscheidungsvariablen für jeden Agenten nur $wMax$ Entscheidungsvariablen verwendet (d.h. für jede Woche eine), in denen jeweils eine Referenz auf das Praktikum gespeichert wird, mit dem der Agent in der entsprechenden Woche beginnt.

[20] Dabei wird die Tatsache ausgenutzt, dass bei unterdurchschnittlicher Auslastung eines Praktikums anderswo überdurchschnittliche Auslastungen auftreten müssen – eine regelkonforme Zuteilung der Praktika vorausgesetzt.

ZE.	Beschreibung	Berechnung für einen Studenten s
z_1	Anzahl der Bereiche, in denen der Studierende kein Praktikum zugewiesen bekommen hat.	$z_1^s = PLC_s(A)$
z_2	Strafkosten für Zuteilungen zu nicht priorisierten Praktika	$z_2^s = PC_s(A)$
z_3	Strafkosten für ungünstige Pausenverteilung	$z_3^s = PP_s(A)$
z_4	Strafkosten für überdurchschnittliche Auslastung von Praktika	$z_4^s = \acute{PU}_s(A)$

Tabelle 7.14: *Zwischenergebnisse (ZE.) der Rotations-Agenten*

1.4: Die Initialisierung von Startlösungen:
Ein Agent initialisiert einen Lösungsvorschlag, indem er sich in jedem Bereich einen Platz zuteilt in dem Kurs, bei dem sein Teilnahmewunsch am höchsten war. Die Zuteilung der übrigen Studierenden wird von der Vervollständigungs-Funktion der Umgebung übernommen.

1.5: Die Integration lokaler Verbesserungsverfahren, Spezifikation von Situationen und Plänen
Die Funktion *getStatus* wurde so implementiert, dass sie auf Grundlage der Güte-Koeffizienten eine der in Tabelle 7.15 aufgeführten Situationen zurück liefert. Die Situations-Plan-Matrix kann Tabelle 7.16 entnommen werden, die entsprechenden Pläne werden in Tabelle 7.17 näher erläutert. Die bei der Ausführung eines Plans angewendeten Operationen liefern bezüglich der *hard constraints* ausschließlich gültige Lösungsvorschläge zurück.

Situation	Beschreibung
LacksCourse	Der vom Agenten repräsentierte Studierende hat in mindestens einem Bereich kein Praktikum zugewiesen bekommen – die Lösung verstößt also gegen *soft constraints*.
Unsatisfying-Lab	Der vom Agenten repräsentierte Studierende hat in einem Bereich nicht sein Wunschpraktikum erhalten.
LabTooFull	Ein zugewiesenes Praktikum ist überdurchschnittlich stark ausgelastet.
BadBreak-Distribution	Die Verteilung der Pausen entspricht nicht dem Wunsch des Studierenden.
Satisfied	Der Agent ist mit dem Lösungsvorschlag zufrieden, d.h. der Studierende hat in allen Bereichen Praktika erhalten, seine Präferenzen wurden berücksichtigt und die zugewiesenen Praktika sind gleichmäßig ausgelastet.

Tabelle 7.15: *Situationen der Rotations-Agenten.*

1.6: Die Rekombination von Vorschlägen
Zwei Vorschläge werden rekombiniert, indem ein Agent zunächst die eigenen Belegungen aus dem zweiten Ausgangsvorschlag übernimmt – dieser wurde dem Agenten zur Rekombination übergeben, da er lokal zufriedenstellend war. Stammt der Lösungsvorschlag aus einer Koalition, so werden auch die Belegungen der Koalitionsmitglieder übernommen. Im Anschluss werden – soweit möglich – die Belegungen der übrigen Studierenden aus dem ersten Vorschlag übernommen. Dabei wird ausgeschlossen, dass Lösungsvorschläge entstehen, die gegen *hard constraints* verstoßen.

	New-Courses	Get-Course	Switch-Course	Swap-Course	Switch-Week	Swap-Week	Swap-All
LacksCourse	0.1	0.6	0	0	0.1	0.1	0.1
UnsatisfyingLab	0.1	0.3	0.3	0.2	0	0	0.1
LabToFull	0.1	0	0.2	0.2	0.2	0.2	0.1
BadBreak-Distribution	0.1	0	0	0	0.3	0.3	0.3
Satisfied	0.1	0.15	0.15	0.15	0.15	0.15	0.15

Tabelle 7.16: *Situations-Plan-Matrix für das Problem der Klinischen Rotation*

Plan	**Beschreibung**
New-Courses	Alle Praktika des durch den Agenten repräsentierten Studierenden im Lösungsvorschlag werden gelöscht, im Anschluss werden neue Praktika zugewiesen.
Get-Course	Der Agent versucht, einen Platz für „seinen“ Studierenden für ein bestimmtes Praktikum zu bekommen und „verdrängt“ dazu – wenn dies aufgrund von Kapazitätsengpässen nötig ist – auch Komilitonen aus diesem Kurs. Vorher zugewiesene Plätze in anderen, sich überschneidenden Praktika, gibt er frei.
Switch-Course	Der Agent versucht, einen Kursplatz in einem Praktikum für einen freien Kursplatz in einem anderen Praktikum einzutauschen.
Swap-Course	Der Agent versucht, mit einem anderen Agenten einen Kursplatz zu tauschen.
Switch-Week	Der Agent versucht, den Zeitraum eines Praktikums zu verlegen, d.h. einen Platz für das gleiche Praktikum in einer anderen Woche zu bekommen.
Swap-Week	Der Agent versucht, seinen Praktikumsplatz mit einem weiteren Agenten zu tauschen, der dasselbe Praktikum in einem anderen Zeitraum belegt.
Swap-All	Der Agent tauscht seine gesamten Praktikumsplätze mit einem anderen Agenten.

Tabelle 7.17: *Pläne der Rotations-Agenten*

1.7: Die Auswahl von Kandidaten für die Bildung einer Koalition
Als Kandidaten zur Bildung einer Koalition liefert die implementierte Methode *getPossibleCoalitionMembers* alle Agenten zurück, die Studierende verwalten, die im übergebenen Lösungsvorschlag gemeinsame Praktika mit dem Agent zugewiesen bekommen haben.

Schritt 2: Die Implementierung der Umgebung

2.1: Implementierung der Vervollständigungs- und Reparaturfunktion
Die Funktion *completeAndRepairProposal* wurde so implementiert, dass freie Kursplätze, die bei einem Lösungsvorschlag noch ohne die Verletzung von *hard constraints* zugeteilt werden könnten, auch tatsächlich zugeteilt werden. Dazu wird für alle Agenten überprüft, ob noch Kursplätze fehlen und ob entsprechende Zeitfenster und Kurskapazitäten frei sind. Eine Reparatur von Vorschlägen wird dagegen nicht benötigt, da alle Operationen zur Initialisierung und Variation von Lösungsvorschlägen bereits die Einhaltung der (harten) Nebenbedingungen sicherstellen.

2.2: Implementierung der Restfunktion für die Bewertung von Lösungsvorschlägen
Die Restfunktion *remainderEvaluation* für die Bewertung von Lösungsvorschlägen wird nicht benötigt, die Zielfunktion wurde vollständig in partielle Evaluierungsfunktionen zerlegt. Die Abweichung durchschnittlicher und tatsächlicher Auslastungen von Praktika scheint zwar dann nicht in die Bewertung einzugehen, wenn ein Praktikum in einem Zeitintervall *keinen* Teilnehmer hat, in diesem Fall muss die Auslastung jedoch in mindestens einem anderen Intervall überdurchschnittlich hoch sein, so dass dort entsprechende Strafkosten auftreten.

Schritt 3: Implementierung des Konferenzmonitors

3.1: Graphische Darstellung von Lösungsvorschlägen
Mit der Methode *drawProposal* wird der jeweils beste Lösungsvorschlag dargestellt. Abbildung 7.8 zeigt die Darstellung eines Lösungsvorschlags, die in einer Matrix-artigen Form erfolgt. Jede Zeile der Matrix repräsentiert einen Studierenden, jede Spalte eine Kalenderwoche. Eine Zuteilung eines Studierenden zu einem Praktikum wird durch einen grauen, grünen, gelben oder roten Balken mit der Praktikumsnummer angezeigt. Die Länge des Balkens gibt die Dauer des Praktikums wieder, die Farbe gibt darüber Aufschluss, ob es sich um das von Studierenden bevorzugte Praktikum des entsprechenden Bereichs handelt.

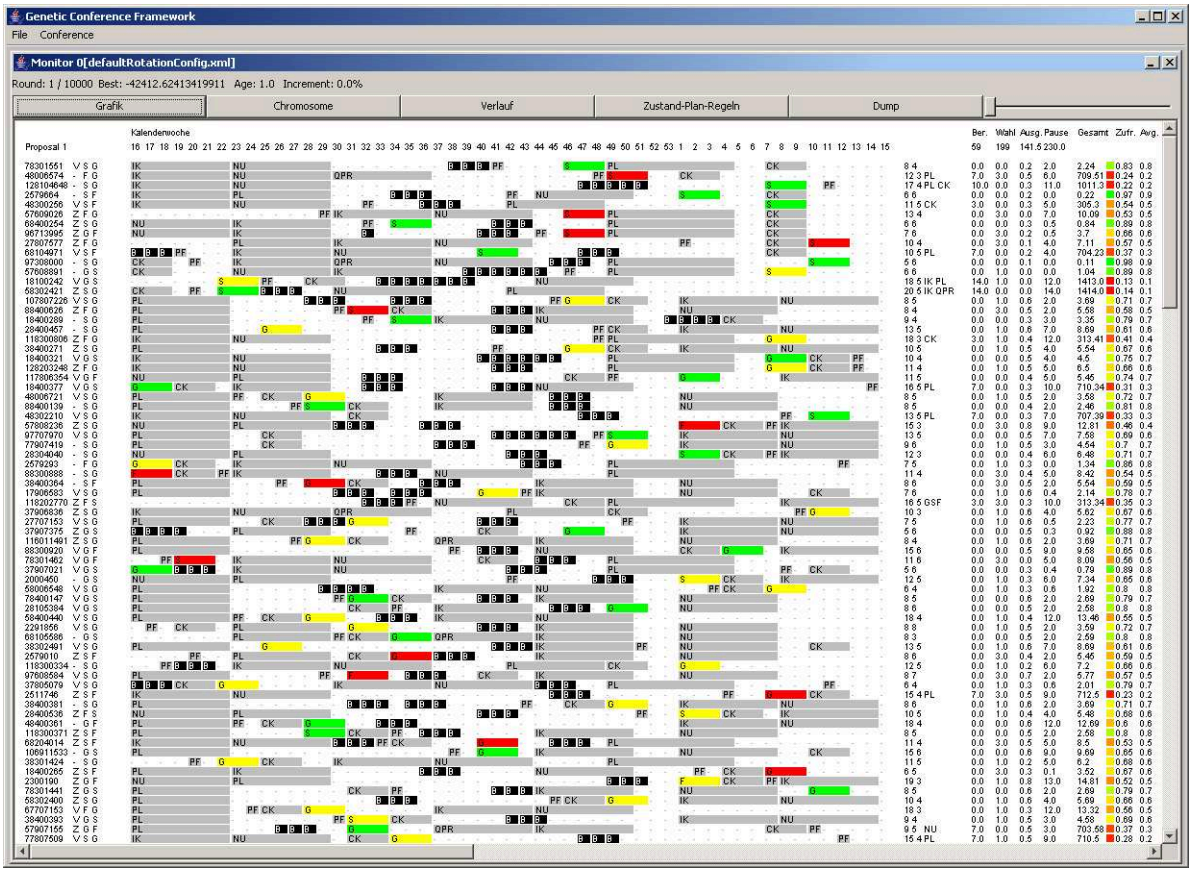


Abbildung 7.8: *Konferenzmonitor für Lösungsvorschläge der klinischen Rotation*

Anderweitig belegte Wochen werden durch schwarze Balken gekennzeichnet. Einer zusätzlichen Matrix (nicht in der Abbildung dargestellt) kann die Zahl der freien Plätze in den unterschiedlichen Praktika entnommen werden.

3.2: Textuelle Ausgabe von Lösungsvorschlägen
Die Methode *dumpProposal* schreibt die Inhalte eines Lösungsvorschlags in eine Matrix, deren Aufbau im Wesentlichen der oben beschriebenen graphischen Ausgabe entspricht.

7.4.5 Einsatz des angepassten Framework für die Rotation 2007 und 2008

Die problemspezifische Implementierung der Genetischen Konferenz für die klinische Rotation wurde im Herbst 2006 fertig gestellt, im Anschluss ausführlich getestet und bei der Verteilung der Kurse für den Zeitraum April 2007 bis April 2008 (Sommersemester 2007 und Wintersemester 2007/08), die im November 2006 stattfand, erstmals erfolgreich angewendet. Das Programm benötigte für eine bezüglich der *soft-* und *hard constraints* gültige Zuteilung der 280 Studierenden auf die insgesamt 1960 Praktikumsplätze ca. 3 Minuten.[21] Nach ca. einer halben Stunde konnte eine Zuteilung erreicht werden, bei der allen Studierenden das Praktikum mit höchstmöglicher Priorität zugeteilt wurde. Den Präferenzen der Studierenden hinsichtlich der Verteilung der Pausen konnte ebenfalls weitestgehend nachgekommen werden.

Aufgrund dieser überzeugenden Ergebnisse entschloss sich der Dekan der Tierärztlichen Fakultät der LMU München, Prof. Dr. Göbel, die Zuteilung zur Rotation 2008/09 direkt anschließend durchzuführen. Dabei wurde eine Erweiterung implementiert, die es den Studierenden erlaubt, zusätzlich bis zu drei Wünsche für freizuhaltende Wochen anzugeben. Diese Wünsche konnten bei nahezu allen Studierenden berücksichtigt werden.

Für die Anmeldung zur Rotation 2009/2010 ist wieder eine Erweiterung vorgesehen: Jetzt sollen die Studierenden jeweils Kommilitonen angeben können, mit dem sie ihre Praktika möglichst gemeinsam absolvieren wollen. Das Programm soll dann versuchen, die Plätze so zu verteilen, dass diese Studierenden möglichst viele Praktika gemeinsam besuchen können.

7.4.6 Performanzvergleich der Anpassung mit Simulationen der Basisverfahren MAS und GA

Die Anforderungen an das Optimierungsverfahren zur Zuteilung der Praktikumsplätze für die Klinische Rotation sind sehr speziell, so dass auf einen Vergleich der Anpassung des Framework mit anderen Lösungsverfahren für Zuordnungsprobleme an dieser Stelle verzichtet wird. Die Anpassung soll jedoch wieder in einem kleinen Performanztest mit Varianten verglichen werden, die einen Greedy-MAS, bei dem nur ein Lösungsvorschlag bearbeitet wird und einem GA ohne Delegation von Lösungsvorschlägen und Situations-spezifischer Operatorauswahl simulieren. Für die Tests wurde als Problemexemplar die Verteilung der Rotationskurse für das Jahr 2008/09 herangezogen, für jede der Varianten wurden 10 Optimierungsläufe durchgeführt.

Es ergeben sich ähnliche Resultate wie bei den vorausgegangenen Anwendungsbeispielen: Während die MAS-Variante zunächst sogar bessere Ergebnisse liefert, stagniert sie später deutlich

[21] Die Angaben beziehen sich auf einer Ausführung des Programms auf einer Workstation mit 2.4 GHz und einem Arbeitsspeicher von 1 GB.

öfter in schlechteren lokalen Optima. Dabei scheinen sich die Verlaufskurven für das integrierte Verfahren und der MAS-Variante untereinander nicht wesentlich zu unterschieden – während die GA-Variante deutlich unterlegen ist. Das überrascht nicht, denn ähnlich wie bei der Generalisierung thematischer Karten handelt es sich um ein Problem, bei dem sich die Zielfunktion von vornherein aus partiellen Bewertungsfunktionen zusammensetzt, so dass sich eine verteilte Bearbeitung mit einem MAS anbietet. Die Rekombination von Lösungsvorschlägen scheint dagegen zunächst weniger erfolgversprechend. Interessant ist jedoch, dass auch der „schlechteste“ der 10 Optimierungsläufe der Genetischen Konferenz mit einem Wert von ca. -189 „Strafpunkten“ besser abschneidet als der beste Lauf der MAS-Variante (ca. -214). Mit dem Student'schen t-Test auf Abweichung der Mittelwerte der Ergebnisse nach 5000 ausgewerteten Lösungsvorschlägen lässt sich noch bei einer Irrtumswahrscheinlichkeit von unter $10^{-5}\%$ eine signifikante Abweichung nachweisen. Insbesondere im späteren Verlauf der Optimierung, wenn bereits bezüglich der Nebenbedingungen gültige Lösungen gefunden wurden und der Fokus auf die Wünsche der Studierenden gelegt werden kann, scheint die Verfügbarkeit von Rekombinationsoperatoren also doch von Vorteil zu sein. Dafür spricht auch, dass bei den Läufen der Genetischen Konferenz über die Hälfte der in den Vorschlagspool eingebrachten Lösungsvorschläge (die die Selektion auch tatsächlich „überstehen“) aus Rekombinationen resultieren.

Die GA-Variante, bei der die Operatoren und ihre Anwendungsstellen zufällig ausgewählt werden, fällt bereits im frühen Verlauf der Optimierung deutlich ab und ist auch später den Varianten mit zielgerichteter Suche nach Verbesserungspotential und spezifischer Operatorauswahl unterlegen, wie der gemeinsamen Darstellung der Verlaufskurven für die drei Varianten (Abbildung 7.9) entnommen werden kann. Angezeigt werden jeweils die Mittelwerte der besten bisher gefundenen Lösungen.

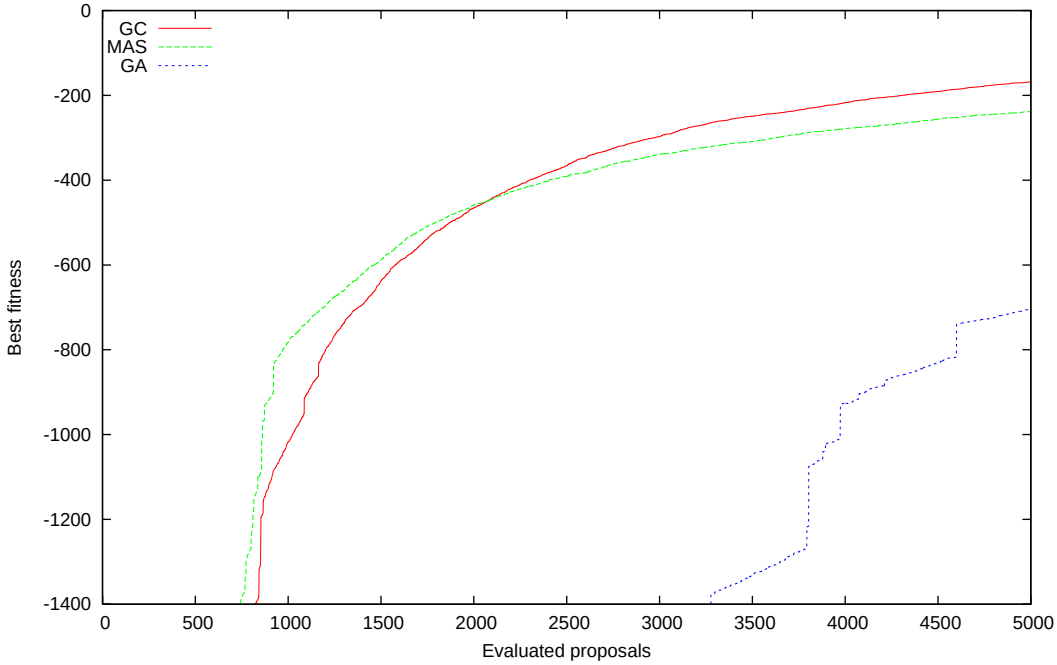


Abbildung 7.9: *Performanzvergleich der Genetischen Konferenz (GC) mit Simulationen eines Genetischen Algorithmus (GA) und eines Greedy-MAS (MAS) für die Zuteilung zu den Praktika*

Kapitel 8

Evaluierung verwendeter Konzepte anhand zweier TSP-Exemplare

Dieses Kapitel beschreibt die Ergebnisse einiger Performanztests, mit denen die Auswirkungen unterschiedlicher Konzepte der Genetischen Konferenz evaluiert werden.

Für die Anwendungsbeispiele wurden jeweils Testläufe durchgeführt, um das integrierte Verfahren mit Simulationen der Basisverfahren Greedy-MAS und GA zu vergleichen. Beschreibungen der Ergebnisse finden sich am Ende der jeweiligen Abschnitte. Hier sollen nun ausführlichere Tests anhand von Exemplare des bekannten *Traveling Salesman Problem* durchgeführt werden.

Dazu wird die in Abschnitt 7.1 beschriebene exemplarische Anpassung des Framework in unterschiedlichen Modi mit zwei Problembeispielen getestet, die auf der Website http://www.tsp.gatech.edu/ veröffentlicht sind [TSP06]. Es handelt sich um „Real World Problems“, d.h. reale Koordinaten von Städten jeweils eines Landes auf der Erde. Bei der Berechnung der Distanzen werden Luftlinienentfernungen zugrunde gelegt, die Problembeispiele sind also metrisch. Als Testdaten für die Anpassung der Genetischen Konferenz auf Traveling-Salesman-Probleme wurden die Datensätze der beiden kleinsten Probleme „Western Sahara“ und „Djibouti“ herangezogen.[1] Das Problembeispiel „Western Sahara“ ist das kleinste auf der genannten Website angebotene Problembeispiel. Es besteht aus 29 Städten. Die Länge der optimalen Tour beträgt 27601. Das Problembeispiel „Djibouti“ besteht aus 89 Städten, die Länge der optimalen Tour beträgt 6656.

Zur Berechnung der kürzesten Touren benötigte der TSP-Lösungsalgorithmus *concorde* auf einem AMD Athlon 1.33 GHz 0,09 bzw. 0,24 Sekunden [TSP06]. Bereits vorab sei bemerkt, dass Optimierungsläufe mit dem Framework deutlich langsamer sind. In der Regel werden globale Optima erst nach etwa 10 bzw. 20 Sekunden gefunden.

[1] Abbildung 7.1 auf Seite 134 zeigt die jeweils kürzesten Rundreisen für die beiden Problembeispiele „Western Sahara“ (links) und „Djibouti“ (rechts).

8.1 Testvarianten

Die Anpassung des Framework zur Bearbeitung metrischer TSP wurde mit verschiedenen Testvarianten auf die zwei Problembeispiele angesetzt. Die im Folgenden näher beschriebenen Testvarianten entstehen durch An- und Abschalten bestimmter Optionen in den XML-Konfigurationsdateien (vgl. 6.3.4). Tabelle 8.1 zeigt die Eigenschaften der Testvarianten im Überblick.

Genetische Konferenz (GC): Alle Optionen – bis auf das Lernen von Aktionswahrscheinlichkeiten[2] – werden angeschaltet. Die Agenten optimieren mit Gedächtnis, sie bilden Koalitionen und delegieren zu verbessernde Vorschläge untereinander je nach Höhe des jeweiligen lokalen Verbesserungspotentials. Anzuwendende Variationsoperatoren werden anhand der SP-Matrix ausgewählt.

Optimierung ohne Koalitionen (NO-COAL): Im diesem Modus dürfen im Unterschied zum GC-Modus keine Koalitionen unter den Agenten gebildet werden.

Optimierung ohne Gedächtnis (NO-MEM): Im NO-MEM-Modus wird im Unterschied zum GC-Modus die Memory-Funktion der Agenten abgeschaltet.

Optimierung ohne Berücksichtigung lokalen Verbesserungspotentials (NO-SAT): Im Unterschied zum GC-Modus berechnen die Agenten keine lokalen Zufriedenheitswerte, lokales Verbesserungspotential wird also nicht beachtet. Daher findet auch keine Delegation von Lösungsvorschlägen an andere Agenten statt und weder Gedächtnisse noch Koalitionen werden verwendet. Die zur Anwendung kommenden Variationsoperatoren werden jedoch Situations-spezifisch ausgewählt.

Hybrid-GA (HGA): Ein mit den problemspezifischen Mutationsoperatoren hybridisierter genetischer Algorithmus wird simuliert. Die Pläne zur Verbesserung von Vorschlägen werden von den Agenten zufällig (und nicht auf Basis der aktuellen Situation) ausgewählt. Eine Delegation von Lösungsvorschlägen findet nicht statt. Außerdem werden keine Vorschläge in Gedächtnisse aufgenommen und keine Koalitionen gebildet.

Greedy-MAS (MAS): Es wird nur an einem Lösungsvorschlag gearbeitet, allerdings können sich die Agenten Vorschläge merken und Koalitionen bilden. Eine Rekombination des bearbeiteten Lösungsvorschlags kann daher nur mit einem Vorschlag aus dem Gedächtnis oder einer Koalition durchgeführt werden.

GC mit Permutations-basierten Operatoren (R-GC): Um den Einfluss problemspezifischer Operatoren auszuschalten und so speziell die Auswirkung partieller Evaluationen zu messen, werden in diesem Modus nur zufällige, Permutations-basierte Mutations-Operatoren verwendet, die kein Wissen über die Entfernungen der Städte untereinander ausnutzen. Konkret werden der zufällige Positionstausch zweier Städte und das zufällige

[2] Die Wahrscheinlichkeiten in der SP-Matrix wurden zwar anhand mehrerer Durchläufe mit lernenden Agenten festgelegt (vgl. Abschnitt 5.4.8). Um zusätzliche Einflüsse auf die Performanz der Testvarianten zu vermeiden, wurde die Anpassung der Werte in der Matrix jedoch abgeschaltet. Zur Beurteilung der Auswirkungen der Option auf die Performanz des Algorithmus – insbesondere bei Fällen, in denen sich der Nutzen bestimmter Pläne in einer Situation im Verlauf der Optimierung ändert – stehen Untersuchungen noch aus.

Verschieben einer Stadt in der Reihenfolge der Rundreise als Mutations-Operatoren verwendet. Eine Situations-spezifische Operatorauswahl findet nicht statt. Ansonsten werden alle Konzepte der Genetischen Konferenz genutzt.

Greedy-MAS mit Permutations-basierten Operatoren (R-MAS): Mit dieser Variante wird ein Greedy-MAS mit rein Permutations-basierten Operatoren und ohne Situations-spezifische Operatorauswahl simuliert.

GA mit Permutations-basierten Operatoren (R-GA): Ein genetischer Algorithmus, der nur Permutations-basierte Variationsperatoren verwendet, wird simuliert.

	GC	NO-COAL	NO-MEM	NO-SAT	HGA	MAS	R-GC	R-MAS	R-GA
Delegation von Lösungen	x	x	x			x	x	x	
Bildung von Koalitionen	x		x			x	x	x	
Verwendung von Gedächtnissen	x	x				x	x	x	
Spezifische Operatorauswahl	x	x	x	x		x			
Problemspezifische Operatoren	x	x	x	x	x	x			
Mehrpunkt-verfahren	x	x	x	x	x		x		x

Tabelle 8.1: Eigenschaften der unterschiedlichen Testvarianten

Mit Ausnahme der Testvarianten MAS und R-MAS wird jeweils eine Populationsgröße von 20 verwendet. MAS und R-MAS arbeiten dagegen nur auf einem Lösungsvorschlag. Bei der Selektion kommt eine Elite-Auswahl (n=5) zur Anwendung. Die Anzahl der Vorschläge in den individuellen Gedächtnissen der Agenten ist – sofern überhaupt Gedächtnisse verwendet werden – auf 10 beschränkt.

8.2 Performanzvergleich der Varianten ohne problemspezifische Operatoren

Um die Performanz der Genetischen Konferenz im Vergleich mit den Basisverfahren zunächst ohne den Einfluss problemspezifischer Operatoren zu testen, wurden jeweils 20 Optimierungsläufe mit den drei Modi R-GC, R-MAS und R-GA für die beiden Problembeispiele durchgeführt.

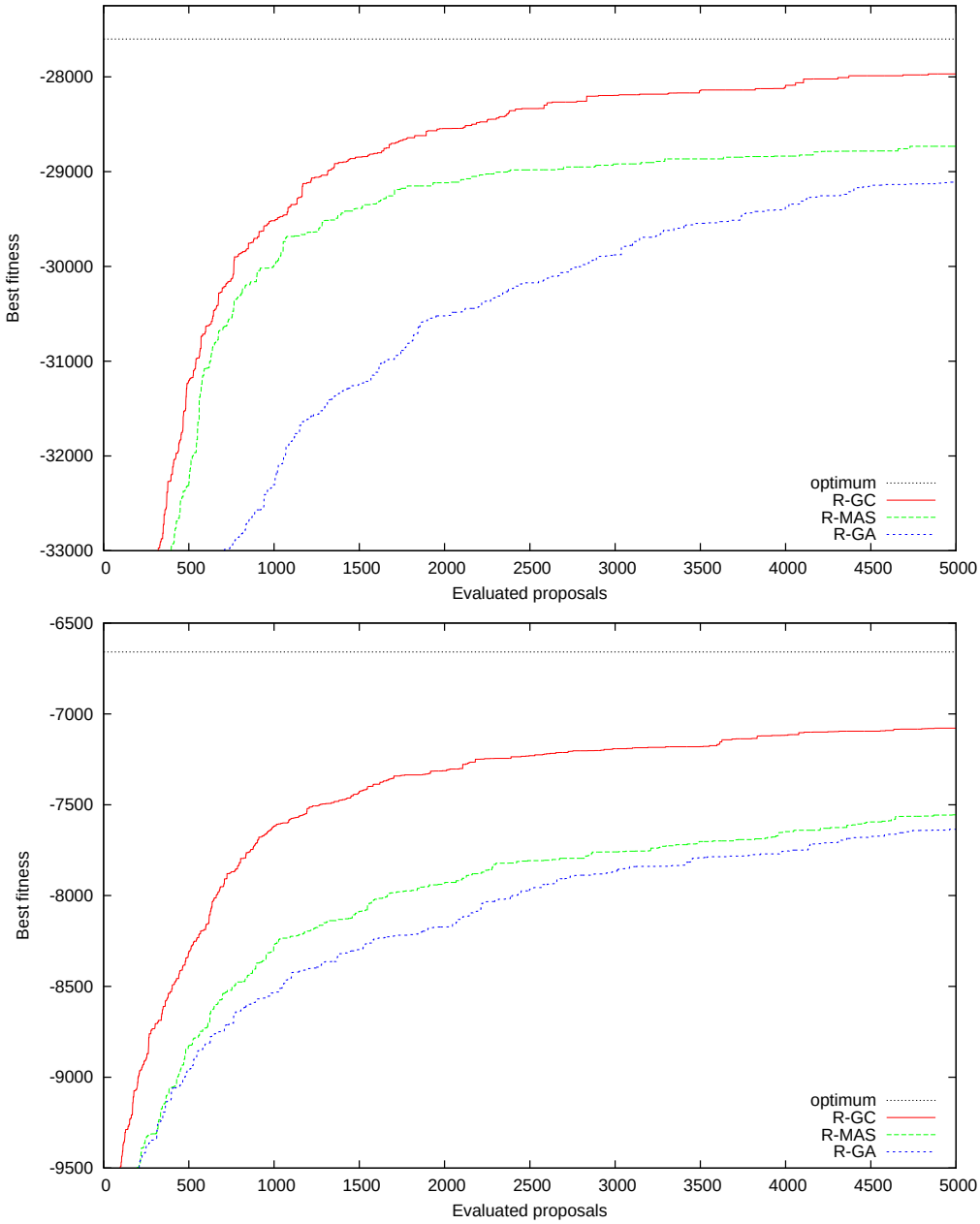


Abbildung 8.1: *Vergleich der Testvarianten R-GC, R-GA und R-MAS für die Exemplare „Western Sahara“ (oben) und „Djibouti“ (unten): Mittelwerte von jeweils 20 Optimierungsläufen.*

Der Abbildung 8.1 kann entnommen werden, dass durch die Integration der Konzepte MAS und GA deutliche Beschleunigungen bei der Suche nach guten Lösungen gegenüber den isolierten Verfahren erreicht werden.

Beim kleineren Problembeispiel (in der Abbildung oben) zeigen sich zudem auffällige Unterschiede beim Verhalten der beiden Modi R-MAS und R-GA: Während beim R-MAS-Modus zu Beginn der Optimierung deutliche Qualitätssteigerungen des (einzigen) bearbeiteten Vorschlags zu erkennen sind, können im späteren Verlauf kaum noch Verbesserungen erreicht werden – das Verfahren stagniert oft in relativ schlechten lokalen Optima. Im R-GA-Modus sind die Verbesserungen dagegen gleichmäßiger über die Generationen verteilt. Das integrierte Verfahren R-GC zeigt zu Beginn der Optimierung ebenso rasche Fortschritte wie das Basisverfahren R-MAS. Allerdings können – ebenso wie bei der R-GA-Variante – auch im späteren Verlauf der Optimierung noch signifikante Verbesserungen erzielt werden. Hier die parallele Bearbeitung *mehrerer* (Alternativ-)Vorschläge bei der Vermeidung der (endgültigen) Stagnation in schlechten lokalen Optima.

Beim größeren Problembeispiel ist die Überlegenheit des integrierten Verfahrens gegenüber den isolierten Basisverfahren sogar noch deutlicher zu erkennen. Hier sind dagegen die Unterschiede in den Verlaufskurven von R-GA und R-MAS untereinander weniger stark ausgeprägt. Offenbar kompensieren sich hier die Vor- und Nachteile der Basisverfahren über den Verlauf der Optimierung – jedenfalls wenn die Werte die Fitness der jeweils besten Lösungsvorschläge über mehrere Läufe gemittelt wird.

8.3 Performanzvergleich der Varianten mit problemspezifischen Operatoren

Um die Auswirkungen der einzelnen Konzepte wie Delegation von Lösungsvorschlägen, Situations-spezifische Operatorauswahl, Verwendung von Gedächtnissen und Bildung von Koalitionen zu evaluieren, wurden des Weiteren Tests mit den Varianten durchgeführt, die die unter 7.1 beschriebenen *problemspezifischen Operatoren* verwenden. Wiederum wurden für das Problembeispiel „Western Sahara“ und das Problembeispiel „Djibouti“ jeweils 20 Testläufe durchgeführt. Abbildung 8.2 zeigt die Verlaufskurven für die durchschnittliche Güte der jeweils besten Vorschläge im Vorschlagspool, gemittelt über die Testläufe.

Man erkennt, dass wie bei den Performanztests für die Anwendungsbeispiele zunächst mit der MAS-Variante am schnellsten gute Ergebnisse erzielt werden. Ab einer gewissen Lösungsgüte stagniert das Verfahren in dieser Variante jedoch bei beiden Problembeispielen oft in lokalen Optima und wird von den Varianten GC, NO-COAL und NO-MEM nach ca. 500 bis 1000 ausgewerteten Vorschlägen eingeholt (die drei Varianten sind in unterschiedlichen Rottönen dargestellt). Dieses Verhalten lässt sich dadurch erklären, dass bei der MAS-Variante nur an einem Vorschlag gearbeitet wird. Da die Variationsoperatoren zielgerichtet eingesetzt werden, werden anfangs nur wenige Evaluationen benötigt, um gute Ergebnisse zu erzielen. Im späteren Verlauf der Optimierung werden dann aber kaum noch Verbesserungen erzielt, da die erreichten lokalen Optima durch Anwendung der lokalen Variationsoperatoren nur selten verlassen werden können und Rekombinationen nur eingeschränkt möglich sind. Die Varianten GC, NO-COAL und NO-MEM arbeiten auf mehreren Vorschlägen. Aufgrund des geringeren Selektionsdrucks finden sie

dadurch zwar anfangs etwas langsamer gute Lösungen, stagnieren dafür später weniger oft in schlechten lokalen Optima und liefern dann im Durchschnitt bessere Ergebnisse. Sie sind daher der MAS-Variante (Einpunkt-Verfahren) überlegen.

Steigerungen der Performanz der Genetischen Konferenz durch die Verwendung von Gedächtnissen und Koalitionen sind nur beim größeren Problembeispiel deutlich zu erkennen, die Kurven

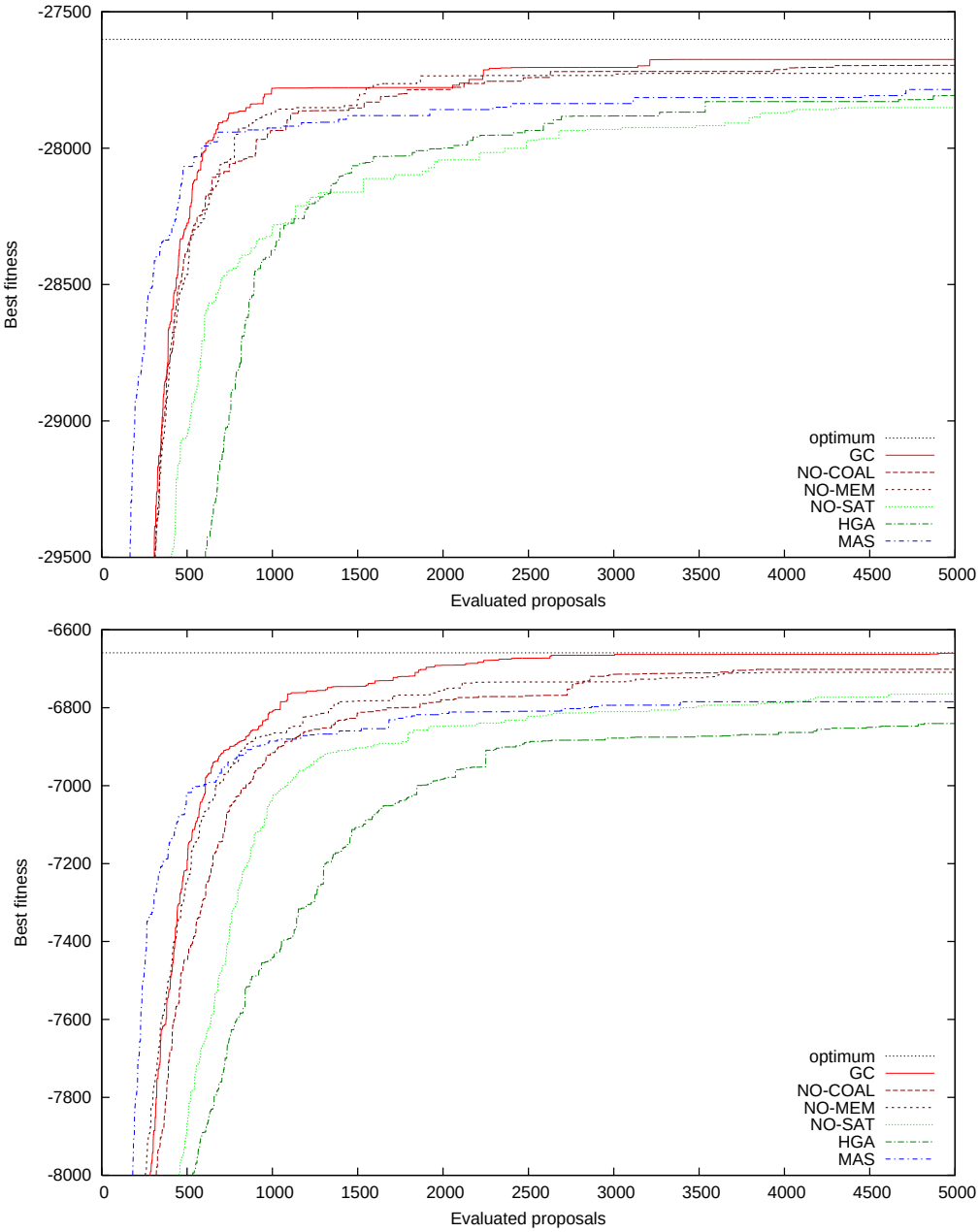


Abbildung 8.2: *Vergleich der Testvarianten „GC“, „NO-MEM“, „NO-COAL“, „NO-SAT“, „HGA“ und „MAS“ für die TSP-Exemplare „Western Sahara“ (oben) und „Djibouti“ (unten): Mittelwerte von jeweils 20 Optimierungsläufen.*

der Varianten GC, NO-COAL und NO-MEM unterscheiden sich beim kleineren Beispiel dagegen kaum – auch wenn die GC-Variante auch hier fast über den gesamten Verlauf durchschnittlich die besten Ergebnisse erzielt. Offenbar wirkt sich die zusätzliche Speicherung von Lösungsvorschlägen, die partiell besonders gut evaluiert wurden, erst mit steigender Zahl *möglicher* Konfigurationen günstig aus.

Die Varianten ohne die Beachtung lokalen Verbesserungspotentials (NO-SAT) und ohne Situations-spezifische Operatorauswahl (HGA) fallen bei beiden Problembeispielen deutlich zurück. Dabei fällt die Operatorauswahl offenbar beim größeren Anwendungsbeispiel mehr ins Gewicht, signifikante Unterschiede in den Verlaufskurven lassen sich nur dort erkennen.

8.4 Konvergenzsicherheit der Testvarianten

Zuletzt soll die Zuverlässigkeit der Testvarianten analysiert werden, d.h. wie oft das tatsächliche globale Optimum erreicht wurde.

Konvergenzsicherheit bei der Bearbeitung des Problembeispiels „Western Sahara“

Beim Problembeispiel Western Sahara beträgt das globale Optimum 27601. In den Testläufen konnte dieses globale Optimum im GC-Modus bei 17 von 20 Fällen nach höchstens 5000 ausgewerteten Vorschlägen erreicht werden. Beim No-MEM-Modus konnte das globale Optimum in 16 Fällen, ohne die Bildung von Koalitionen in 17 Fällen erreicht werden. Ohne Beachtung der lokalen Zufriedenheitswerte (NO-SAT-Modus) wurde das globale Optimum in 14 Fällen erreicht. Die Simulation des Hybrid-GA (HGA-Modus), bei dem die problemspezifischen Operatoren mit gleicher Wahrscheinlichkeit unabhängig von der Situation des Agenten angewendet wurden, führte in 8 Fällen zum globalen Optimum. Während die Unterschiede zu den Varianten ohne Beachtung von lokalem Verbesserungspotential relativ deutlich ausfallen, haben Gedächtnisse und Koalitionen offenbar weniger große Auswirkungen auf die Konvergenzsicherheit.

Anzahl evaluierter Vorschläge	GC	NO-COAL	NO-MEM	NO-SAT	HGA	MAS	R-GC	R-MAS	R-GA
500	2	0	0	0	0	5	0	0	0
1000	9	4	10	2	0	7	0	0	0
1500	12	7	11	8	3	8	0	0	0
2000	13	10	14	10	5	8	0	0	0
2500	16	11	16	10	7	9	3	0	0
3000	16	13	16	10	7	9	7	0	0
3500	17	13	17	11	7	10	8	1	1
4000	17	13	17	12	8	10	8	1	1
4500	17	15	17	14	8	11	8	1	1
5000	17	16	17	14	8	12	9	2	2

Tabelle 8.2: *Western Sahara: Häufigkeit des Erreichens des globalen Optimums bei jeweils 20 Testläufen*

Die MAS-Variante erreichte das globale Optimum 12 mal, bei einem Viertel der Testläufe wurde es bereits nach weniger als 500 ausgewerteten Vorschlägen gefunden. Bei den übrigen Testläufen stagniert das Verfahren jedoch früh in relativ schlechten lokalen Optima.

Bei den Testvarianten mit rein permutationsbasierten Operatoren ergeben sich erheblichere Unterschiede: Während die Varianten, die die Basisverfahren MAS und GA simulieren, jeweils nur zweimal das globale Optimum finden konnten, gelang das mit der Variante, die die Ansätze integriert, fast bei der Hälfte der Testläufe (9 mal).

Tabelle 8.2 zeigt die Anzahl erreichter globaler Optima für die verschiedenen Varianten im Verlauf der Optimierung. Dazu ist in Intervallen von 500 ausgewerteten Vorschlägen jeweils die Zahl der Testläufe angegeben, bei denen das globale Optimum bereits gefunden wurde.

Konvergenzsicherheit bei der Bearbeitung des Problembeispiels „Djibouti“

Beim größeren Problembeispiel „Djibouti“ fallen die Unterschiede zwischen dem GC-Modus und den Varianten ohne Verwendung von Gedächtnissen und Koalitionen etwas deutlicher aus. Während im GC-Modus in 17 Testläufen nach spätestens 5000 Lösungsvorschlägen das globale Optimum von 6659 erreicht war, wurde dieses ohne Bildung von Koalitionen nur 15 mal und ohne Gedächtnisse nur 13 mal gefunden werden.

Bei Nichtbeachtung lokalen Verbesserungspotentials (NO-SAT) konnte das globale Optimum nur noch 10 mal erreicht werden, während ohne Situations-spezifische Auswahl von Variationsoperatoren nur noch in 6 Fällen die kürzeste Rundreise gefunden wurde.

Betrachtet man nur die Häufigkeiten des Erreichens globaler Optima, schneidet die MAS-Variante im Vergleich zum kleineren Problembeispiel mit nur 4 „erfolgreichen“ Testläufen relativ schlecht ab. Hier scheinen sich also die fehlenden Rekombinationsmöglichkeiten noch stärker auszuwirken, viele Testläufe stagnieren in lokalen Optima (siehe unten).

Anzahl evaluierter Vorschläge	GC	NO-COAL	NO-MEM	NO-SAT	HGA	MAS	R-GC	R-MAS	R-GA
500	0 (0)	0 (0)	0 (1)	0 (0)	0 (0)	1 (2)	0 (0)	0 (0)	0 (0)
1000	5 (4)	3 (1)	2 (5)	0 (0)	0 (0)	2 (5)	0 (0)	0 (0)	0 (0)
1500	7 (5)	6 (3)	4 (6)	1 (4)	0 (1)	3 (6)	0 (0)	0 (0)	0 (0)
2000	9 (6)	7 (4)	5 (7)	4 (4)	1 (2)	3 (6)	0 (0)	0 (0)	0 (0)
2500	13 (4)	10 (4)	8 (5)	5 (4)	2 (3)	3 (6)	0 (0)	0 (0)	0 (0)
3000	14 (4)	10 (5)	10 (3)	8 (2)	3 (3)	4 (6)	1 (0)	0 (0)	0 (0)
3500	14 (5)	14 (2)	10 (4)	8 (2)	4 (2)	4 (7)	1 (0)	0 (0)	0 (0)
4000	15 (5)	14 (3)	13 (3)	8 (3)	4 (2)	4 (7)	1 (1)	0 (0)	0 (0)
4500	15 (5)	14 (3)	13 (3)	9 (2)	5 (3)	4 (7)	1 (1)	1 (0)	0 (0)
5000	17 (3)	15 (2)	13 (3)	10 (2)	6 (3)	4 (7)	1 (1)	1 (0)	0 (0)

Tabelle 8.3: *Djibouti: Häufigkeit des Erreichens des globalen Optimums von 6659 bei jeweils 20 Testläufen. In Klammern zusätzlich angegeben ist die Anzahl der Testläufe, die bereits das sehr gute lokale Optimum von 6664 erreicht haben.*

Die Varianten ohne problemspezifische Variationsoperatoren finden das globale Optimum fast nie. Im Unterschied zum kleineren Problemexemplar sind dabei kaum Unterschiede zwischen den Varianten R-GC, R-GA und G-MAS zu erkennen.

Das Problemexemplar Djibouti weist ein lokales Optimum auf, das mit einem Zielfunktionswert von 6664 weniger als 0,1% schlechter ist, als das globale Optimum. Dieses lokale Optimum ist insbesondere bei der Analyse der Testläufe der MAS-Variante interessant, da es von 7 der 20 Testläufen erreicht wird. Das lokale Optimum wird dabei meist schon relativ früh gefunden (und dann nicht wieder verlassen). Betrachtet man das lokale Optimum als ausreichend gut, so können insgesamt 11 Optimierungsläufe als „erfolgreich“ betrachten werden. Damit sind die Resultate wiederum sehr ähnlich, wie die des kleineren Problembeispiels.

Alle drei Testläufe der GC-Variante, die das globale Optimum nicht finden, liefern immerhin auch das nur unwesentlich schlechtere lokale Optimum zurück.

8.5 Zusammenfassung der Evaluationsergebnisse

Die Integration der Basisverfahren MAS und GA wirkt sich sowohl mit, als auch ohne Verwendung problemspezifischer Operatoren günstig auf die Performanz aus. Werden gleichzeitig Variationsoperatoren gezielt auf Lösungsteile angesetzt, die Verbesserungspotential aufweisen *und* mehrere Lösungen parallel bearbeitet (und auch rekombiniert), ergeben sich in beiden Fällen deutlich schneller gute Ergebnisse, als wenn nur einer der Ansätze allein verfolgt wird.

Vom MAS-Ansatz übernimmt die Genetische Konferenz das Konzept der Erkennung lokalen Verbesserungspotentials, das offenbar insbesondere zu Beginn der Optimierung schnell zu guten Resultaten führt. Mit den Testvarianten, die nur auf einem Lösungsvorschlag arbeiten, werden jedoch im späteren Verlauf der Optimierung kaum noch Verbesserungen erreicht.

Die parallele Bearbeitung *mehrerer* Lösungsvorschläge und die Verwendung von Rekombinationsoperatoren hat die Genetische Konferenz mit dem Ansatz der Genetischen Algorithmen gemeinsam. Diese Konzepte helfen, lokale Optima öfter zugunsten besserer Lösungen wieder zu verlassen zu können, und damit auch im späteren Verlauf der Optimierung noch Verbesserungen zu erzielen.

In den Tests wirkt sich die Erkennung von Verbesserungspotential insbesondere in Verbindung mit der Situations-spezifischen Auswahl von Variationsoperatoren positiv aus. Die Resultate der entsprechenden Varianten sind für beide Problembeispiele relativ deutlich denen anderer Varianten überlegen.

Performanzsteigerungen durch die Verwendung von Gedächtnissen und die Bildung von Koalitionen lassen sich vor allem beim größeren Problemexemplar erkennen. Dort werden bei Verwendung *beider* Konzepte bessere Ergebnisse erzielt, als wenn eine der Optionen abgeschaltet bleibt. Beim kleineren Exemplar sind die Unterschiede geringer.

Trotz der Integration von Greedy-Strategien bei der Auswahl der Verbesserungsstelle und dem Variationsoperator wird die *Konvergenzsicherheit* der Genetischen Konferenz gegenüber einem Hybrid-GA ohne entsprechender Situations-spezifischer Auswahl nicht verringert – im Gegenteil erreicht die Variante, bei der alle Konzepte der Genetischen Konferenz zur Anwendung kommen, bei beiden Problembeispielen (mit) am Häufigsten das globale Optimum. Dagegen stagnieren

die Testvarianten MAS und R-MAS, die nur an einer Lösung arbeiten, relativ früh in *lokalen* Optima.

Abschließend soll noch einmal darauf hingewiesen werden, dass die vorgestellte problemspezifische Implementierung des Framework nur *eine mögliche* Vorgehensweise darstellt. Mit anderen Variationsoperatoren sowie abweichenden Formen der Messung der lokalen Zufriedenheit sind sicherlich auch andere Ergebnisse zu erwarten.

Da für das TSP sehr gute problemspezifische Heuristiken existieren – man beachte dazu die genannten Rechenzeiten für die Problemexemplare mit dem *concorde*-Lösungsalgoritmus – wurde von einer weitergehenden Untersuchung guter Variationsoperatoren für TSP-Exemplare in dieser Arbeit jedoch abgesehen – Ziel war hier nur, die Optimierungsgeschwindigkeit und Konvergenzsicherheit der Genetische Konferenz gegenüber den Basisverfahren zu testen sowie die Auswirkungen der Verwendung unterschiedlicher Konzepte zu evaluieren.

Kapitel 9

Zusammenfassung und Ausblick

9.1 Zusammenfassung

Ziel dieser Arbeit war die Entwicklung eines Optimierungsverfahrens für Kombinatorische Optimierungsprobleme (KOP) aus der Praxis und seine prototypische Implementierung in einem Framework. Mit dem Verfahren sollen Optimierungsprobleme bearbeitet werden können, bei denen der Anwender zwar über eingeschränktes Wissen zu partieller Bewertbarkeit von Lösungsvorschlägen verfügt sowie Auswahlmechanismen für problemspezifische Variationsoperatoren kennt, sich eine Bearbeitung von Hand aber – jedenfalls ab einer gewissen Problemgröße – nicht mehr anbietet. Als Anwendungsbeispiele für das Verfahren wurden ein Standortplanungsproblem, ein Fahrplanoptimierungsproblem sowie ein Zuteilungsproblem herangezogen.

Grundlage des Verfahrens ist ein *Genetischer Algorithmus (GA)*. GA arbeiten auf einer *Population* von Lösungsvorschlägen, die einem Variations- und Selektionsprozess unterworfen werden. Wie für alle universellen Verfahren gelten auch für GA die *No-Free-Lunch*-Theoreme: Gemittelt über alle Optimierungsprobleme ist ein GA nicht schneller als bloße Zufallssuche. Um die Arbeitsweise eines GA zu beschleunigen, werden die ursprünglichen randomisierten Variationsoperatoren eines *kanonischen* GA meist durch problemspezifische Operatoren ersetzt oder ergänzt. Durch diese Art der *Hybridisierung* verliert der GA zwar seine universelle Einsetzbarkeit, übertrifft jedoch bei der Anwendung auf die entsprechende Problemklasse universelle Verfahren in der Regel bezüglich der Performanz bei weitem.

Das entwickelte Verfahren stellt eine wesentliche Erweiterung solcher *Hybridsysteme* dar, da zusätzlich Anwenderwissen über die Situations-spezifische *Auswahl* passender Operatoren und erfolgversprechender *Anwendungsstellen* für einen konkreten Lösungsvorschlag integriert werden kann. Dabei orientiert sich das Verfahren an einem intuitiven Schema bei der Bearbeitung von Hand: Der Anwender sucht ausgehend von einer bisherigen Lösung einen *Lösungsteil*, der noch Verbesserungspotential aufweist. Anschließend wendet er genau auf diesen Lösungsteil einen geeigneten Operator an.

Bei der automatisierten Bearbeitung von Optimierungsproblemen wird nun ein *Multiagentensystem (MAS)* als Wissensträger verwendet, um Kenntnisse des Anwenders in das Verfahren einzubringen. Die Agenten des MAS verwalten jeweils eines der dem KOP zugrunde liegenden Objekte und sorgen für eine individuell günstige Belegung der entsprechenden Entscheidungsva-

riablen. Anders als etwa bei Ameisen- oder anderen Schwarmalgorithmen verwalten die Agenten also nicht individuelle *Lösungsvorschläge* sondern die *Variablen*. Ähnliche Ansätze wurden für die verteilte Bearbeitung von Constraint-Satisfaction-Problemen vorgeschlagen, sie werden hier auf das allgemeinere Anwendungsgebiet der KOP übertragen und durch die Integration der Konzepte Genetischer Algorithmen wesentlich erweitert: Es wird nicht mehr nur an einem Lösungsvorschlag gearbeitet, sondern an einer ganzen *Population*, die einem Selektionsprozess unterliegt. Durch diese *Kombination* aus MAS und GA entsteht ein performantes und gleichzeitig robustes Optimierungsverfahren.

In der Arbeit finden sich eine formale Definition des verknüpften Optimierungsverfahrens sowie detaillierte Erläuterungen zum Ablauf. Dieser lässt sich metaphorisch vergleichen mit einer Konferenz unter Vertretern unterschiedlicher Interessensgruppen, die zu einem Thema nach einer für alle annehmbaren Kompromisslösung suchen. Das Verfahren wird daher – und in Anlehnung an das Basisverfahren GA – als *Genetische Konferenz (GK)* bezeichnet.

Die Genetische Konferenz wurde prototypisch implementiert und in ein Framework für die Bearbeitung Kombinatorischer Optimierungsprobleme eingebunden. Ein Vorgehensmodell für die problemspezifische Anpassung des Framework wurde geliefert und anhand des Standardbeispiels TSP erläutert. Die Implementierungen des Framework für die Bearbeitung der drei Anwendungsfälle *Standortplanung*, *Fahrplanoptimierung* und *Zuteilung von Studierenden zu Praktika* wurden anhand dieses Vorgehensmodells durchgeführt. Sie liefern jeweils sehr gute Ergebnisse auf Exemplaren realistischer Größe und unterstreichen damit die praktische Relevanz dieser Arbeit. Es wurden Tests durchgeführt, die die verbesserte Performanz im Vergleich mit den Basisverfahren MAS und GA für alle drei Anwendungsfälle aufzeigen.

Die exemplarische Anpassung für das TSP wurde des Weiteren in unterschiedlichen Varianten getestet, um die Auswirkungen der verwendeten Konzepte zu evaluieren. In den Testläufen wurden bei der Bearbeitung zweier kleiner TSP-Exemplare (29 bzw. 89 Städte) deutliche bessere Ergebnisse erzielt, wenn Lösungsvorschläge partiell bewertet und auf Grundlage der Bewertung die Variationsstelle und der Variationsoperator jeweils spezifisch ausgewählt werden. Durch die Verwendung individueller *Gedächtnisse* und die Bildung von *Koalitionen* konnte die Optimierungsgeschwindigkeit bei einem der Exemplare sogar noch weiter gesteigert werden.

Das entwickelte Framework stellt das zentrale, praktische Ergebnis der Arbeit dar. Seine wesentlichen Vorteile seien abschließend noch einmal zusammengefasst:

- Der Anwender kann das universell definierte Framework auf Probleme aus ganz unterschiedlichen Bereichen anpassen und auf seine individuellen Bedürfnisse zuschneiden. So kann er Probleme automatisiert bearbeiten, für die bisher kein passendes Verfahren zur Verfügung steht, ohne ein solches von Grund auf neu entwickeln zu müssen.
- Der Anwender kann sein problemspezifisches Wissen einbringen und die Optimierung so erheblich gegenüber universellen Verfahren beschleunigen.
- Die Anpassung des Framework auf konkrete Problemexemplare ist einfach durchführbar. Es müssen lediglich drei abstrakte *Java*-Klassen implementiert werden.

9.2 Ausblick

9.2.1 Weitere Anwendungsfälle für das entwickelte Framework

Im Laufe der Entwicklung des Verfahrens ergaben sich weitere interessante Anwendungsfälle für das Framework, von denen hier einige erwähnt werden sollen.

Bereits im Einsatz ist eine problemspezifische Implementierung des Framework für die *Zusammenstellung von Prüfungsgruppen bei Promotionsverfahren* der Fakultät für Tiermedizin an der LMU München. Die Zusammensetzungen der Prüfungsgruppen müssen einigen Nebenbedingungen genügen, beispielsweise muss einer der Zweitbetreuer sich in der Prüfungsgruppe eines Kandidaten befinden, während der Erstbetreuer einer anderen Gruppe zugeteilt werden muss.

Diese Anpassung hat sich in den letzten beiden Semestern bewährt, so dass eine Ausweitung der Anwendung auch für die Zusammenstellung anderer Prüfungen der Fakultät für Tiermedizin geplant ist. Ziel ist dabei unter anderem, den Studierenden bei der Auswahl ihres Prüfers und dem Prüfungstermin soweit wie möglich entgegen zu kommen, während eine gleichmäßige „Auslastung“ der Dozenten gewährleistet bleibt.

Weitere Anwendungsfälle ergeben sich auch im Bereich der Standortplanung und der Optimierung von Taktfahrplänen. In Abschnitt 7.2.5 wurden bereits die zusätzlichen Planungsszenarios *Filialstandortplanung* und *Vertriebsgebietsplanung* beschrieben, auf die das Framework nach entsprechender Anpassung eingesetzt werden soll. Die Weiterentwicklung und Anpassung erfolgt hier in Zusammenarbeit mit der Firma WIGeoGIS. Auch die Implementierung für die Optimierung Integraler Taktfahrpläne soll erweitert werden, so dass neben der Grobplanung auch die Feinplanung von ITF sowie die mehrstufige Planung für große Netze durchgeführt werden kann.

Eine weitere Anpassung ist für einen Anwendungsfall geplant, der den in Abschnitt 2.1 kurz beschriebenen *Vehicle Routing Problemen (VRP)* zuzuordnen ist. Bei VRP wird ähnlich wie beim TSP nach kurzen Touren gesucht, typischerweise müssen jedoch zusätzliche Nebenbedingungen wie bestimmte Kapazitätsbeschränkungen eingehalten werden. Die Anpassung wird voraussichtlich im Zuge einer Diplomarbeit vorgenommen werden, die vom Autor mitbetreut wird.

9.2.2 Mögliche Erweiterungen des Optimierungsverfahrens

Auch wenn sich der Algorithmus in seiner derzeitigen Form in der Praxis bereits sehr bewährt, sind unter anderem folgende Erweiterungen angedacht:

- Raumbezug spielt bei der Mehrzahl der Anwendungen bereits eine große Rolle. Speziell für diesen Anwendungsbereich könnte eine Erweiterung nützlich sein, die zusätzliche Agenten für *Teilräume des Untersuchungsgebiets* vorsieht („Teilraum-Agenten“). Ausgehend von der Annahme, dass die Probleme eine gewisse (räumliche) Dekomponierbarkeit aufweisen, ist eine Rekombination von Teillösungen für unterschiedliche Teilräume vielversprechend – die entsprechenden Teillösungen könnten von den Teilraum-Agenten verwaltet werden. Möglicherweise bietet sich auch eine hierarchische Gliederungen der Teilräume an, so dass die Verwendung von Agenten auf unterschiedliche Ebenen denkbar wäre.
- Noch nicht ausgeschöpft sind die Möglichkeiten der speziellen Behandlung multikriterieller Probleme. Die Agenten berechnen zwar bereits mehrere Güte-Koeffizienten für unterschied-

liche Kriterien, allerdings wird die Pareto-Dominanz bei der Selektion von Lösungsvorschlägen (für den Generationenübergang und für die Reproduktion) noch nicht berücksichtigt. Möglich wäre hier eine Erweiterung, bei der beispielsweise alle dominierten Lösungsvorschläge aus der Population verdrängt werden, um möglichst einen großen Teil der Pareto-Front abzudecken. Ebenso möglich wäre eine Aufspaltung der Population ähnlich wie beim VEGA-Ansatz von Schaffer [Scha85], die Anwendung von *sharing*-Konzepten oder die Einführung von *mating-restrictions* [Gol89].

- Die Agenten könnten versuchen, noch stärker auf die Zusammensetzung des Vorschlagspool zu reagieren. Um beispielsweise die Stagnation des Optimierungsverfahrens in lokalen Optima zu vermeiden, könnten sie bestimmte Belegungen bzw. Kombinationen von Belegungen von Entscheidungsvariablen verbieten – ähnlich einem Tabu-Attribut bei der Tabu-Suche. Ein solches Verbot könnte ein Agent dann aussprechen, wenn keine oder nur geringe Variation dieser Belegungen im Vorschlagspool gegeben ist und die einzige vorkommende Belegung (bzw. Kombination von Belegungen) für den Agenten nicht zufriedenstellend ist.

- Bei der Modellierung der Genetischen Konferenz wurden durch die Verwendung nebenläufiger Prozesse (*Java-Threads*) die Grundlagen für eine *parallele* Erstellung und Bewertung von Lösungsvorschlägen geschaffen. Die Prozesse der Agenten können auf die unterschiedlichen Prozesse eines Mehrprozessorsystems verteilt werden.

  Daher liegt der nächste Schritt nahe, die Implementierung so zu erweitern, dass die Agentenprozesse *verteilt* auf *unterschiedlichen Rechnern* ausgeführt werden können. Abbildung 9.1 zeigt ein Modell mit einem Primärrechner, auf dem der Moderator-Prozess ausgeführt wird und mehreren Sekundärrechnern, auf denen jeweils die Agentenprozesse laufen. Die Kommunikation unter den Prozessen auf den unterschiedlichen Rechnern kann weiterhin über Methodenaufrufe erfolgen, wobei nun auch Methoden von Objekten aufgerufen werden müssen, die sich nicht auf demselben Rechner befinden (*Remote Method Invocation, RMI*). Bei der Implementierung dieser Erweiterung können entsprechende *Java*-Pakete genutzt werden.

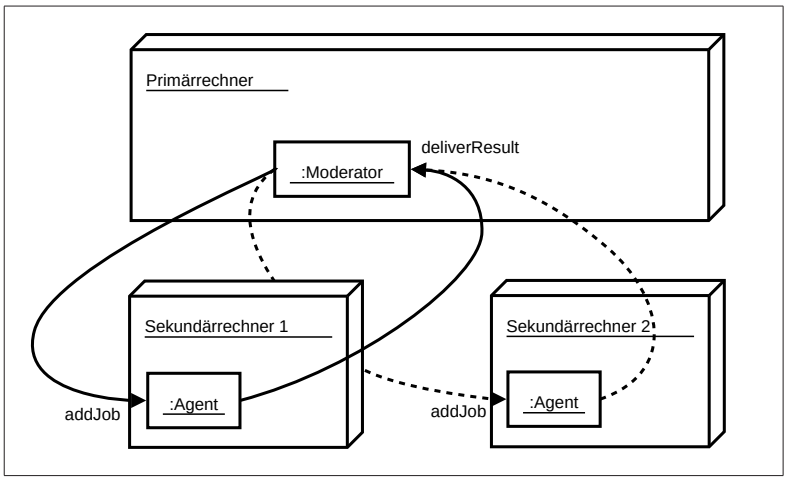


Abbildung 9.1: *Die Genetische Konferenz auf einem verteilten System*

Anhang A

Weiterführende Erläuterungen und Beispiele

Im diesem ersten Teil des Anhangs finden sich einige weiterführende Erläuterungen und Beispiele zu den theoretischen Grundlagen der Arbeit (Kapitel 4).

A.1 Bekannte Metaheuristiken, die auf lokaler Suche basieren

A.1.1 First-Choice-Hill-Climbing

Beim *First-Choice-Hill-Climbing* wird die möglicherweise zeitintensive Evaluation *aller* Vorschläge aus der Nachbarschaft vermieden. Stattdessen wird jeweils zufällig *ein* Vorschlag aus der Nachbarschaft ausgewählt und sofort zu diesem übergegangen, wenn er besser evaluiert wird, als der momentane Vorschlag.

A.1.2 Kernighan-Lin Algorithmus

Der *Kernighan-Lin Algorithmus*[1] [LK73] versucht, dem „Tradeoff“ zwischen Aufwand für die Evaluation und zu erwartender Lösungsgüte auf andere Weise zu begegnen: Basierend auf einer kleinen Nachbarschaft – für das TSP wird die 2-change-Nachbarschaft gewählt – wird versucht, in einem Suchschritt durch iterative Anwendung *mehrerer* kleinerer Transformationen eine *entfernte*, bessere Lösung zu finden, ohne *alle* Lösungen einer entsprechend großen Nachbarschaft auszuwerten. Stattdessen wird mittels einer problemspezifischen „greedy-Strategie“ die lokal bestmögliche Transformation ausgeführt. Dabei dürfen sich *zwischenzeitlich* (d.h. während eines Suchschritts) auch schlechtere Lösungen als die Ausgangslösung ergeben, wie bei anderen *Hill-Climbing*-Verfahren finden aber *Übergänge* nur zu *insgesamt besseren* Lösungen statt. Die Anzahl Transformationen bis zu einem Übergang ist variabel, das Verfahren wird daher auch als *Variable-Depth Search* bzw. bei Anwendung auf das TSP als *Variable-k-change*-Verfahren bezeichnet. Eine detaillierte Erklärung des Algorithmus findet sich z.B. in [Hro03].

[1] Helsgaun [Hel00] entwickelte mit *LKH* eine effektive Implementierung des Kernighan-Lin Algorithmus für TSP. Der Algorithmus an sich ist jedoch auch auf andere Probleme übertragbar (vgl. z.B. [Mer00, S. 68].

A.1.3 Tabu-Suche (TS)

Bei der *Tabu-Suche* [Glo86] wird ausgehend von einer momentanen Lösung die Nachbarschaft durchsucht und zur bestmöglichen Nachbarlösung gewechselt – dabei werden im Unterschied zu einfachen *Hill-Climbing-* (bzw. *Hill-Descending-*) Verfahren auch Verschlechterungen in Kauf genommen, um möglicherweise lokale Optima wieder verlassen zu können. Zur Vermeidung von Zyklen werden bereits evaluierte Lösungen in eine Gedächtnisliste (Tabu-Liste) aufgenommen. Diese Lösungen dürfen zumindest eine gewisse Zeit lang nicht wieder besucht werden (*Tabu*). Entscheidend für die Laufzeit einer Tabu-Suche ist die effiziente Implementierung der Tabu-Liste. Dabei spielt die geeignete Auswahl der Tabulistenlänge eine wichtige Rolle. Die explizite Speicherung *aller* besuchten Lösungen ist ab einer gewissen Problemgröße technisch nicht mehr durchführbar. Eine Alternative ist die Speicherung so genannter *Tabu-Attribute.* Der Algorithmus entscheidet anhand gespeicherter Lösungsattribute, ob eine Lösung tabu ist. Eine Tabu-Suche endet, wenn eine vorgegebene Anzahl von Iterationen überschritten ist, seit einer vorgegebenen Anzahl von Zügen keine Verbesserungen mehr erreicht werden konnten, alle Nachbarlösungen tabu sind oder wenn eine befriedigende Lösung erreicht wurde. Zurückgegeben wird die beste im Verlauf des Verfahrens gefundene Lösung – dies gilt auch für die Verfahren SA, TA und SI.

A.1.4 Simulated Annealing (SA)

Simulated Annealing (SA) wurde unabhängig voneinander von Cerny [Cer83] und Kirkpatrick et al. [KGV83] entwickelt. Vorbild ist der natürliche Abkühlungsprozess einer Schmelze zum Festkörper. SA ist eine Weiterentwicklung des Metropolis-Verfahrens, die Darstellung erfolgt hier nach [MAK07]: Ausgehend von einer momentanen Lösung x wird eine zufällig ausgewählte Nachbarlösung $\acute{x}$ evaluiert. Die Nachbarlösung wird dann als momentane Lösung akzeptiert, wenn sie mindestens ebenso gut ist – d.h. für ein Maximierungsproblem $f(\acute{x}) \geq f(x)$ gilt – oder wenn die Verschlechterung nicht zu gravierend ist, also unter einer bestimmten Schwelle liegt. Dieses Verfahren wird iterativ angewendet. Eine schlechtere Lösung wird beim Schritt k mit einer gewissen *Wahrscheinlichkeit* W angenommen, deren Höhe abhängig ist vom Grad der Verschlechterung und einem Kontrollparameter, der so genannten „*Temperatur*“ t_k:

$$W(\text{accept } \acute{x}) = \begin{cases} 1 & \text{wenn } f(\acute{x}) \geq f(x) \\ exp\left(\frac{f(\acute{x})-f(x)}{t_k}\right) & \text{wenn } f(\acute{x}) < f(x) \end{cases}$$

Während der Kontrollparameter t_k beim ursprünglichen Metropolisverfahren fest ist, wird er beim SA im Laufe des Verfahrens immer weiter abgesenkt („Abkühlung“). Für SA kann asymptotische Konvergenz auf ein globales Optimum nachgewiesen werden, ein entsprechender Beweis findet sich zum Beispiel in [MAK07].

A.1.5 Threshold Accepting (TA)

Threshold Accepting [DS90] ist dem Simulated Annealing sehr ähnlich, aber einfacher zu implementieren. Wiederum wird die momentane Lösung zufällig lokal variiert und das Ergebnis wird immer akzeptiert, wenn dadurch eine bessere Lösung entsteht. Beim Threshold Accepting wird im Gegensatz zu SA aber keine *Wahrscheinlichkeit* für die Annahme einer schlechteren Lösung

berechnet. Stattdessen wird eine schlechtere Lösung im Schritt k *immer* akzeptiert, wenn die Verschlechterung unter einem Schwellenwert t_k liegt, der im Verlauf des Optimierungsprozesses wieder sukzessive bis auf Null verringert wird. Zum Vergleich sei wiederum die Akzeptanzwahrscheinlichkeit in derselben Notation wie bei der Beschreibung von SA angegeben:

$$W(\text{accept } \acute{x}) = \begin{cases} 1 & \text{wenn } f(\acute{x}) + t_k > f(x) \\ 0 & \text{wenn } f(\acute{x}) + t_k \leq f(x) \end{cases}$$

A.1.6 Sintflut-Algorithmus (SI)

Auch beim *Sintflut-Algorithmus* [DSW93] wird iterativ von einer momentanen Lösung aus eine zufällig ausgewählte Nachbarlösung ausgewertet. Zur neuen Lösung wird im Schritt k immer dann übergegangen, wenn sie besser als ein bestimmter Schwellenwert („Wasserstand“) t_k ist. Der Schwellenwert t_k beim Sintflut-Algorithmus bezieht sich also nicht auf die Differenz der Zielfunktionswerte (wie beim TA) sondern nur auf die Güte der neuen Lösung. Den „virtuellen Wasserstand“ lässt man zunächst schnell und dann immer langsamer ansteigen bzw. für Minimierungsprobleme absinken. In der für SA und TA verwendeten Notation lässt sich die Akzeptanzwahrscheinlichkeit dann wie folgt ausdrücken:

$$W(\text{accept } \acute{x}) = \begin{cases} 1 & \text{wenn } f(\acute{x}) > t_k \\ 0 & \text{wenn } f(\acute{x}) \leq t_k \end{cases}$$

A.2 Terminologie Evolutionärer Algorithmen

Die wichtigsten Begriffe aus der Terminologie Evolutionärer Algorithmen[2] seien an dieser Stelle aufgeführt (vgl. [Nis97, S. 13]):

Individuum: Ein Individuum repräsentiert einen Lösungsvorschlag für ein Problembeispiel, der mit der Zielfunktion evaluiert werden kann. Wird das Individuum für die Bearbeitung mit einem EA speziell kodiert, so spricht man auch von *Genotyp* (kodierte Lösung) und vom *Phänotyp* (dekodierte Lösung) eines Individuums.

Fitness: Die Güte des Lösungsvorschlags, den ein Individuum repräsentiert, wird auch als die Fitness des Individuums bezeichnet.

Chromosom, Gen, Genort, Allel: Die Entscheidungsvariablen werden in der Regel in Form von Vektoren oder Zeichenketten vom EA bearbeitet. Dabei können verschiedenartige Kodierungen der Entscheidungsvariablen (binär, ganzzahlig usw.) verwendet werden. Bei Genetischen Algorithmen werden die (dort meist binär) kodierten Individuen als *Chromosome* bezeichnet. Ein Individuum wird (im Unterschied zum natürlichen Vorbild) normalerweise nur mit einem einzelnen Chromosom kodiert. Die einzelnen Komponenten eines Chromosoms (bei binärer Kodierung die einzelnen Bits) bezeichnet man auch als Gene. Mit Allelen sind die möglichen Ausprägungen eines Genorts (der Stelle im Chromosom) gemeint.

[2] Die Begriffe *Chromosom*, *Gen*, *Genort* und *Allel* sind eigentlich nur für Genetische Algorithmen gebräuchlich.

Mutation: Als Mutation bezeichnet man die (oft nur geringfügige) Veränderung eines einzelnen Individuums, bei den Genetischen Algorithmen meist sogar nur eines einzelnen Gens in einem Chromosom. Oft kommen rein zufallsbasierte Operatoren zur Anwendung. Bei nichtbinärer Kodierung wird die Wahrscheinlichkeitsverteilung für die neue Belegung aber oft auch von der bisherigen Belegung abhängig gemacht (bei *Evolutionsstrategien* zum Beispiel durch so genannte Mutationsschrittweiten [Rec94]).

Rekombination und Crossover: Da ein Individuum bei EA ohnehin meist nur durch ein *einzelnes* Chromosom kodiert wird, entfällt die eigentliche *Rekombination* elterlicher Chromosome. Im Unterschied zum natürlichen Vorbild werden daher bei EA die Begriffe *Rekombination* und *Crossover* meist synonym verwendet für die Kombination von Elementen der „Eltern"-Individuen zu einem neuen Individuum. Dies gilt auch für die vorliegende Arbeit.

A.3 Der Ablauf eines kanonischen GA

In diesem Abschnitt wird auf die Ablaufschritte eines kanonischen GA eingegangen, die in Abschnitt 4.3.3 nur im Überblick dargestellt sind.

Schritt 1: Wahl einer geeigneten Kodierung
Holland kodierte beim kanonischen GA Individuen generell *binär*, also in endlichen Folgen der Werte 0 und 1.[3] Eine solche Bitkette kann als Vektor x der folgenden Form dargestellt werden:

$$x = \langle x_1, x_2, ..., x_n \rangle \, ; x_i \in \{0, 1\}$$

Diese Art der Kodierung kommt dem biologischen Kode, der auf Basensequenzen beruht und daher ebenfalls diskret ist (allerdings über vier Buchstaben verfügt), recht nahe. Analog zum Vorbild der natürlichen Evolution spricht man auch bei GA vom Genotyp (kodierte Form) und vom Phänotyp (dekodierte Form) eines Individuums.

Bei ES werden die Entscheidungsvariablen dagegen als reelle Zahlen kodiert - einer der wesentlichen Unterschiede der beiden EA-Ausprägungen. Auswirkungen hat die unterschiedliche Kodierung allerdings nur in der Rechengeschwindigkeit sowie bei der Anwendung der Rekombinations- und Mutationsoperatoren. Werden diese Operatoren entsprechend angepasst und bedenkt man, dass jede Zahl im Computer ohnehin binär repräsentiert wird, so verschwindet der Unterschied zwischen binärer und reellwertiger Repräsentation. Oft bietet sich aus praktischen Gründen an, auf binäre Kodierung zu verzichten und den GA stattdessen auf Lösungsrepräsentationen arbeiten zu lassen, *„die sich möglichst ‚natürlich' aus der bearbeiteten Problemstellung ergeben"* [Nis97, S. 46].

Die Wahl einer geeigneten Kodierung der Entscheidungsvariablen kann im Zusammenhang mit den eingesetzten Variationsoperatoren entscheidend sein für die spätere Performanz des EA. Grund dafür sind unter anderem die wechselseitigen Abhängigkeiten der Entscheidungsvariablen im Hinblick auf die Zielfunktion (Epistasie), die bei unterschiedlichen Kodierungen in Art und Ausmaß variieren.

[3] Um bei der Kodierung natürlich-zahliger Entscheidungsvariablen in Binärzahlen den Hammingabstand der Kodes benachbarter Zahlen gering zu halten, können Graykodes verwendet werden.

Bei ES werden im Chromosom nicht nur die Entscheidungsvariablen, sondern zusätzlich so genannte Mutationsschrittweiten für jede Variable kodiert. Eine Mutationsschrittweite ist ein Parameter für die Anwendung des Mutationsoperators auf die kodierte Variable: Sie legt fest, wie stark sich die Werte der Eltern von denen der Nachkommen unterscheiden können. Da sie ebenfalls dem „Evolutionsprozess“ unterworfen ist, wird so die Fortbewegungsgeschwindigkeit des Algorithmus im Suchraum adaptiv gesteuert [Rec94].

Schritt 2: Initialisierung der Startpopulation
Die Startpopulation wird normalerweise stochastisch initialisiert. Um frühzeitige Konvergenz des Algorithmus zu vermeiden, sollte für möglichst große Streuungsbreite in der Ausgangspopulation gesorgt werden. Die Startpopulation kann – möglichst nach dieser Vorgabe – alternativ auch von anderen Algorithmen erzeugt werden. Man spricht dann von einer Hybridisierung des GA.

Schritt 3a: Dekodierung und Bewertung
Bei der Anwendung eines GA wird implizit angenommen, dass durch die Kombination guter (Teil-)lösungen wieder gute oder sogar noch bessere (Gesamt-)lösungen entstehen. Die „fitteren“ Individuen der Population sollen daher mit größerer Wahrscheinlichkeit (bzw. häufiger) am Reproduktionsprozess teilhaben können als schlechtere. Daher müssen die Individuen einer Population zunächst bewertet werden, bevor sie dem Rekombinationsprozess unterzogen werden.

Um die Bewertungsfunktion auf die Individuen anwenden zu können, müssen die Genotypen der Individuen erst mittels einer Dekodierungsfunktion in den Phänotyp überführt werden – diese Dekodierung ist nicht nötig, wenn der GA direkt auf den Entscheidungsvariablen in ihrer ursprünglichen Form arbeitet. Die Bewertungs- oder Fitness-Funktion ordnet in Anschluss jedem Lösungsvorschlag einen Wert zu. Anhand dieses Wertes werden später sowohl die Überlebenswahrscheinlichkeit als auch die Reproduktionswahrscheinlichkeit des entsprechenden Individuums bestimmt. Die Bewertungsfunktion ist damit die einzige Verknüpfung eines (kanonischen) GA mit dem konkreten Problem(exemplar).

Schritt 3b: Anwendung von Heiratsschema und Rekombinationsoperatoren
Mit dem Heiratsschema wird festgelegt, welche Individuen zur Erzeugung von Nachkommen herangezogen werden. Dies geschieht in der Regel in zwei Schritten. Zunächst wird anhand einer *Selektionsfunktion* entschieden, welche Individuen der aktuellen Generation wie oft am Reproduktionsprozess teilhaben dürfen. Meist wird dies durch einen so genannten „mating pool“ realisiert, in den Individuen (auch mehrfach) eingesetzt werden. Im zweiten Schritt werden Paare (oder auch größere Subpopulationen)[4] aus dem „mating-pool“ zusammengeführt und dann dem Rekombinationsprozess unterworfen. Das Heiratsschema ist neben dem Ersetzungsschema eine der zwei Selektionsfunktionen, die beim GA zur Anwendung kommen. Die Selektionsfunktionen haben großen Einfluss auf das Verhalten des GA. Ziel der Selektionsfunktionen ist es, einen guten Kompromiss zwischen „*exploration*“ und „*exploitation*“ zu finden [Nis94]: „Gute“ Bereiche im Suchraum sollen möglichst aufgespürt werden (*exploration*), besonders erfolgversprechende Gebiete aber auch intensiv durchmustert werden (*exploitation*). Ist der „Selektionsdruck“ zu hoch, d.h. nur wenige Individuen haben die Chance, sich zu reproduzieren, so erhöht sich die Wahrscheinlichkeit für frühzeitige Konvergenz in lokalen Optima. Ist er dagegen zu niedrig, unterscheidet sich der GA im Ergebnis kaum von zufälliger Suche.

[4] Üblicherweise werden zur Rekombination analog zum natürlichen Vorbild genau zwei „Eltern“-Individuen ausgewählt, es können aber auch größere Subpopulationen herangezogen werden. Eine Mutation kann auch als Erzeugung eines Nachkommens von einem einzelnen Elternteil – genannt Elter – angesehen werden.

Ein Standardverfahren für die Elternauswahl ist das *„Roulette-Wheel"-Verfahren*. Die Eltern werden jeweils mit einer Wahrscheinlichkeit proportional zu ihrer Fitness ausgewählt.Beim *Tournament-Verfahren* wird stattdessen von k zufällig bestimmten Kandidaten das Individuum mit der höchsten Fitness ausgewählt. Bei der *Marriage-Selection* wird ein beliebiges Individuum ausgewählt und k-1 mal versucht, per Zufall ein „fitteres" zu finden. Findet man ein solches, so wird es verwendet, ansonsten verwendet man das ursprüngliche Individuum.

Im Zusammenhang mit dem Kodierungsproblem ist auch die Suche nach effizienten Crossover-Operatoren ein zentrales Forschungsgebiet der GA-Theoretiker. Geeignete Operatoren, die dazu führen, dass Regionen des Suchraums mit hoher Güte schneller erreicht werden, können den Optimierungsprozess erheblich beschleunigen. Ein „guter" Rekombinationsoperator erhält *zusammengehörende Teillösungen* der Eltern und kombiniert diese zu neuen, guten Gesamtlösungen. Einige der Standard-Operatoren seien an dieser Stelle genannt:

k-Punkt Crossover: k-Punkt Crossover-Operatoren ahmen weitgehend den chromosomalen *crossing over* der natürlichen geschlechtlichen Fortpflanzung nach. Ein k-Punkt Crossover-Operator zerlegt durch k Schnitte die Chromosomen der Eltern in k+1 jeweils korrespondierende Teilsequenzen und erzeugt durch alternierende Konkatenation „mütterlicher" und „väterlicher" Teilsequenzen die Chomosome zweier neuer Individuen. Der einfachste Vertreter der k-Punkt-Crossover-Operatoren ergibt sich für k = 1. Dieser Operator spielt eine besondere Rolle in der GA-Theorie, weil Holland ihn für seinen kanonischen GA verwendet und auch das Schema-Theorem [Hol92] auf seiner Verwendung beruht. Seien $x = \langle x_1, ..., x_n \rangle$ und $y = \langle y_1, ..., y_n \rangle$ die Chromosome der Eltern. Der 1-Punkt-Crossover-Operator $C(x, y)$ kann dann mathematisch folgendermaßen formuliert werden:

$$C(x,y) = \begin{cases} \langle x_1, ..., x_i, y_{i+1}, ..., y_n \rangle \\ \text{und/oder} \qquad\qquad i \in \{1...n\} \text{ zufällig gleichverteilt zwischen 1 und n} \\ \langle y_1, ..., y_i, x_{i+1}, ..., x_n \rangle \end{cases}$$

Uniform Crossover: Beim Uniform Crossover wird das Chromosom eines Nachkommens gebildet, indem für jedes Gen mit gleicher Wahrscheinlichkeit eines der Allele der Eltern eingesetzt wird.

Intermediäre Rekombination: Bei ES spielt die Rekombination von Individuen eine untergeordnete Rolle. Allerdings verwenden Evolutionsstrategen intermediäre Rekombination (vgl. Abschnitt 4.2.3) für die in der Regel reell kodierten Mutationsschrittweiten (so genannte Schrittweitenanpassung). Für diskret kodierteVariablen bieten sich intermediäre Rekombinationen dagegen meist weniger an.

Problemspezifische Operatoren: Für viele Probleme werden problemspezifische Crossover-Operatoren verwendet. Für das TSP und ähnliche Reihenfolgeprobleme verwendet man beispielsweise den Order-Based-Crossover (OBX) oder den Edge-Based-Crossover (EBX). Ziel solcher Operatoren ist, gute Teillösungen (beim TSP beispielsweise Teile von Rundreisen) in Nachkommen möglichst zu erhalten. Durch die Verwendung problemspezifischer Operatoren, die nicht mehr auf Basis der Chromosome sondern auf Grundlage dekodierter Individuen arbeiten, verliert der GA in der Regel seine universelle Einsetzbarkeit.

Schritt 3c: Die Mutation
Anhänger der GA konzentrieren sich eher auf Crossover-Operatoren und messen der *Mutation* weniger Bedeutung bei. Allerdings können auch bei GA Mutationen notwendig sein, um die Erreichbarkeit aller Punkte im Suchraum zu gewährleisten. Sind beispielsweise bei binärer Kodierung alle in der aktuellen Population vorkommenden Allele eines Gens gleich, so können durch alleinige Anwendung des Crossover-Operators Nachkommen mit anderen Allelen niemals produziert werden. Bei kanonischen GA besteht die Mutation $M(x)$ eines Chromosoms $x = \langle x_1, ..., x_n \rangle$ einfach aus dem „Kippen“ eines beliebigen Bits:

$$M(x) = \langle x_1, ..., (1 - x_i), ..., x_n \rangle \,; i \in \{1, .., n\} \text{ zufällig gleichverteilt zwischen 1 und n}$$

Evolutionsstrategen richten - im Gegensatz zu Anhängern der GA - ihr Hauptaugenmerk auf die Mutation und verwenden allein diese als Suchoperator. Bei ES entsteht ein Nachkomme, indem die (reellwertig kodierten) Variablenbelegungen e_1 bis e_i eines Elters übernommen und durch Addition $(0 - \sigma_i)$-normalverteilter Zufallszahlen variiert werden. Die σ_i werden „Mutationsschrittweiten“ genannt und sind mit im Lösungsvorschlag kodiert. Die Wahrscheinlichkeit für die Lage des Nachkommens im Suchraum ist damit durch ein n-dimensionales Hyperellipsoid beschrieben [Schw77, S.127].

Varianten der Mutation sind die auch im biologischen Vorbild nachweisbaren Inversionen und Translationen. Hier werden ganze Teilsequenzen des Chromosoms herausgeschnitten und an anderer Stelle und/oder in umgekehrter Reihenfolge wieder eingefügt. Beim TSP käme beispielsweise eine Translation oder Inversion bestimmter Routenteile als sinnvoller Operator in Frage.

Schritt 3d: Ersetzung der aktuellen Generation durch Nachkommen
Das Ersetzungsschema stellt – neben dem Heiratsschema – eine zweite Form der Selektion von Individuen in der Population dar. Hier wird entschieden, wie sich die Population einer neuen Generation aus Individuen der letzten Generation und neuen Individuen zusammensetzt.

Beim *echten Generationenprinzip* (*generational replacement*) werden Individuen der Elterngeneration komplett durch die Nachkommen ersetzt. Dadurch können bisher beste Individuen verdrängt werden, ohne dass gleich gute oder bessere nachrücken. In diesem Fall steigt die Güte des jeweils besten Individuums über die Generationen hinweg nicht mehr monoton an. Vorteil ist dafür, dass die Dominanz von „Super-Individuen“ durch vollständiges Ersetzen der Population und dadurch vorzeitige Konvergenz in lokalen Optima vermieden werden kann. Der kanonische GA verwendet dieses Ersetzungsschema.

Von *Elitismus* spricht man dagegen, wenn die jeweils n besten Individuen der Elterngeneration unverändert übernommen werden. Dadurch steigt zwar die Güte des jeweils besten Individuums im Verlauf der Optimierung monoton an, es kann jedoch eher zu unerwünschter Konvergenz in einem lokalen Optimum kommen. Üblicherweise wählt man n daher nicht viel größer als 1. Gegenüber dem kanonischen GA hat ein GA, der Elitismus verwendet, einen (eher theoretischen) Vorteil: Man kann – wie bei Simulated Annealing – beweisen, dass der GA für $t \to \infty$ auf ein globales Optimum konvergiert. Der kanonische GA ohne Elite hat diese Eigenschaft nicht.

Beim *Prinzip des stetigen Ersetzens* werden die Nachkommen sequenziell erzeugt und in die aktuelle Population eingefügt, sofern sie besser als das jeweils schlechteste Individuum sind. Dieses schlechteste Individuum wird dadurch aus der Population verdrängt. Dieses Prinzip entspricht am ehesten der natürlichen Evolution - mit der Einschränkung, dass in diesem Modell die Nachkommen *sofort* mit ihren Eltern konkurrieren.

A.4 Ein Beispiel für Schemata-erhaltende Rekombination

Die Auswirkungen der Wahl eines Schemata-erhaltenden bzw. eines Schemata-zerstörenden Crossover-Operators soll an einem einfachen Beispiel veranschaulicht werden. Zu maximieren seien die Zielfunktionen $f_n(x)$ über Vektoren $x = \langle x_1..x_n \rangle$ mit $x_i \in \{0, 1, 2, ..., 9\}$:

$$f_n(x) = -\sum_{i=0}^{n/2-1} |x_{2i+1} - x_{2i+2}| \text{ für } n = 2k, k \in \mathbb{N}$$

Offensichtlich lässt sich die Menge der Entscheidungsvariablen x_i in n/2 Paare partitionieren, so dass zwischen den Teilmengen keine (externe) Epistasie, innerhalb einer Teilmenge jedoch maximale (interne) Epistasie besteht. Günstige Belegungen aus denselben Teilmengen („Building Blocks“) sollten bei der Rekombination daher möglichst nicht getrennt werden.

Zum Test wurden GA mit drei verschiedenen 1-Punkt-Crossover-Varianten auf die Maximierung von f_{10} bis f_{50} angesetzt. Dabei teilt Crossover-Variante 1 nur nach geraden Positionen (erhält also alle Building Blocks), Crossover-Variante 2 teilt nur nach ungeraden Positionen (zerstört also immer einen Block) und Crossover-Variante 3 teilt beliebig. Bei der Mutation wird jeweils eine zufällig ausgewählte Entscheidungsvariable auf einen zufälligen neuen Wert gesetzt. Die Mutationsraten wurden bei allen Varianten gleich hoch gewählt (1%). Die Populationsgröße beträgt 50, das beste Individuum wird jeweils in die neue Generation übernommen (Elite).

Abbildung A.1 zeigt die Konvergenzgeschwindigkeiten der unterschiedlichen Varianten, d.h. die Anzahl der Generationen bis ein globales Optimum erreicht wurde (Median von 15 Optimierungsläufen). Offensichtlich findet der GA mit der Building Block-erhaltenden Variante für fast alle Funktionen schneller ein globales Optimum als der mit dem „zufälligen“ Crossover. Deutlich schlechter schneidet die Crossover-Variante ab, die immer genau einen Block trennt.

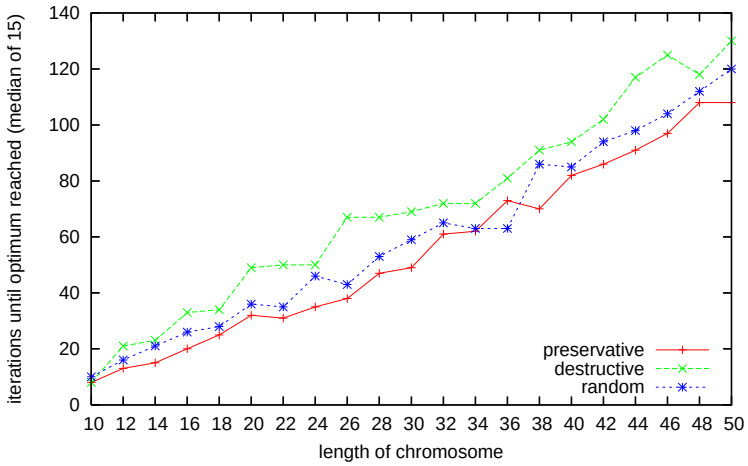


Abbildung A.1: *Testläufe mit verschiedenen Crossover-Varianten*

A.5 Maßzahlen für den Zusammenhang zwischen Distanz und Lösungsgüte

A.5.1 Räumliche Autokorrelation

Das Aussehen der Fitness-Landschaft wird von den Unterschieden der Zielfunktionswerte benachbarter Lösungsvorschläge geprägt. Haben benachbarte Lösungen eher ähnliche Werte, so wird die Fitnesslandschaft ein eher „ebenes“ Aussehen haben. Sind die Werte eher unterschiedlich, so wird sie „zerklüftet“ sein. Man spricht deshalb auch von „ebenen“ und „unebenen“ oder „rauen“ Fitness-Funktionen [KL87].

Weinberger [Wei90] schlägt als Maß für die Unebenheit einer Fitness-Funktion die räumlichen Autokorrelationsfunktion $\rho(d)$ vor. Die Funktion misst die Stärke des Zusammenhangs zwischen räumlicher Nähe und Fitness der Lösungen. Seien N_d^2 alle Paare von Lösungen mit der Distanz d. $\bar{f}$ bezeichne den Mittelwert und $\sigma^2(f)$ die Varianz der Fitness-Werte. Die Autokorrelationsfunktion $\rho(d)$ für die Distanz d berechnet sich dann folgendermaßen:

$$\rho(d) = \frac{1}{\sigma^2(f)\left|N_d^2\right|} \sum_{(x,y)\in N_d^2} \left(f(x) - \bar{f}\right)\left(f(y) - \bar{f}\right)$$

Das Maß kann zwar nur bei Kenntnis *aller* Zielfunktionswerte exakt berechnet werden, zur Schätzung kann jedoch eine Stichprobe von jeweils benachbarten Lösungspaaren herangezogen werden.

Je höher die Korrelation, desto größer ist die Chance, in der Umgebung guter Lösungen weitere gute - vielleicht noch bessere - Lösungen zu finden. Je weniger die Nähe zweier Lösungen mit der Ähnlichkeit der Zielfunktionswerte korreliert, desto weniger kann die Güte einer Lösung als Anhaltspunkt für die Güte von benachbarten Lösungen dienen – und desto geringer sind die Erfolgsaussichten eines Suchverfahrens, das auf einer entsprechenden Nachbarschaftsdefinition beruht.

A.5.2 Korrelationslänge

Neben der *Stärke* ist auch die *Länge* der Korrelation ein entscheidendes Maß für die Unebenheit einer Fitness-Funktion. Mit der Länge der Korrelation wird die maximale Distanz zu einer Lösung bezeichnet, innerhalb derer noch ähnliche Fitness-Werte zu erwarten sind. Weinberger [Wei90] schlägt vor, die Korrelationslänge τ zu berechnen als

$$\tau = -\frac{1}{ln(|\rho(1)|)} \text{ für } \rho(1) \neq 0$$

Andere definieren die Korrelationslänge als die Distanz d, ab der der Autokorrelationskoeffizient $\rho(d)$ einen bestimmten Wert, normalerweise 0.5, unterschreitet (vgl. z.B. [MWS91]). Die Korrelationslänge kann Aufschluss über die Geschwindigkeit geben, mit der ein lokales Verbesserungsverfahren (bei gegebener Nachbarschaftsdefinition) das nächste lokale Optimum findet. Weder räumliche Autokorrelation noch Korrelationslänge können jedoch allein als Eignungskriterium für lokale Verbesserungsverfahren zur Lösung eines Optimierungsproblems gelten: Bei gegebener Definition der Nachbarschaft lassen sich sowohl „rauhe“ Funktionen finden, die leicht zu optimieren sind, als auch „ebene“ Funktionen, die schwer zu optimieren sind.

A.5.3 Fitness-Distanz-Korellation (FDC)

Ein anderes Maß für die Eignung von Heuristiken – in diesem Fall ursprünglich von Genetischen Algorithmen – für die Bearbeitung bestimmter Problemexemplare schlagen Jones und Forrest mit der *Fitness Distance Correlation (FDC)* vor [JF95]. Sie misst den linearen Zusammenhang zwischen der Güte einer Lösung und ihrer Nähe zum nächsten[5] globalen Optimum.

Um die FDC zu berechnen, müssen zunächst die globalen Optima bekannt (und das Optimierungsproblem somit gelöst) sein. Die FDC kann dann wiederum für alle Lösungen berechnet oder aufgrund einer Stichprobe von Lösungen geschätzt werden.

Für die Lösungsmöglichkeiten x_i wird Fitness $f(x_i)$ und Abstand $d_{opt}(x_i)$ zum jeweils nächsten globalen Optimum bestimmt. $\bar{f}$ und $\bar{d}_{opt}$ seien die Mittelwerte und $\sigma(f)$ und $\sigma(d_{opt})$ die Standardabweichungen der Funktionswerte und Distanzen zum jeweils nächsten globalen Optimum. Die FDC berechnet sich dann über die Kovarianz $cov(f, d_{opt})$ zwischen Fitness und Distanz zum nächsten Optimum (hier: Maximum), die mit den Standardabweichungen von Fitness und Distanz zum Optimum normiert wird:

$$cov(f, d_{opt}) = \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - \bar{f})(d_{opt}(x_i) - \bar{d}_{opt})$$

$$FDC = \frac{cov(f, d_{opt})}{\sigma(f)\sigma(d_{opt})}$$

FDC-Werte nahe an +1 deuten darauf hin, dass die Güte von Lösungen im Allgemeinen in Richtung eines globalen Optimums wächst. Solche Probleme sind mit lokalen Verbesserungsverfahren normalerweise eher leicht zu bearbeiten.

Einen Wert nahe 0 nimmt die FDC an, wenn gute Lösungen überall im Suchraum mit ähnlicher (und nicht etwa in der Nähe von globalen Optima mit höherer) Wahrscheinlichkeit gefunden werden können.

Werte kleiner 0 geben an, dass gerade in der Nachbarschaft globaler Optima besonders schlecht bewertete Lösungen liegen und gute Lösungen eher große Entfernungen zu globalen Optima aufweisen. Solche Probleme sind oft besonders schwer zu lösen, da Lösungsalgorithmen, die auf eine positive Korrelation zwischen Fitness und Entfernung zum globalen Optimum setzen, in Bereiche des Suchraums gelenkt werden, in denen kein globales Optimum liegt (so genannte *deceptive Problems*).

Allerdings kann die FDC nicht allgemein als Maß gelten für die Schwierigkeit, eine bestimmte Funktion zu optimieren. Quick, Rayward-Smith und Smith [QRS98] konstruieren zum Beispiel eine Funktion über binäre Entscheidungsvariablen mit – je nach Länge des Bitstrings – geringer oder sogar negativer FDC, die für Hill-Climbing-Verfahren sehr einfach zu optimieren ist.

[5] Die Fitness-Funktion könnte *mehrere* globale Optima besitzen, wenn sie ihr Maximum bzw. Minimum an mehreren Stellen annimmt.

A.6 Davidors Epistasievarianz: Kritikpunkte und alternative Berechnungsmethoden

Die Eignung der Epistasievarianz als Klassifikationsmaß für die Schwierigkeit von Optimierungsproblemen ist aus mehreren Gründen umstritten. Zunächst ist die Epistasievarianz – wie die unter A.5 beschriebenen Kennzahlen – ein eher theoretisches Maß, da ihre exakte Berechnung im Allgemeinen (ebenso wie die Lösung des Optimierungsproblems selbst) exponentielle Zeit benötigt. Ihr Wert kann daher in der Praxis nur mittels einer Stichprobe geschätzt werden, wodurch sich ein nicht unerheblicher Schätzfehler (*sampling bias*) ergeben kann.

Des Weiteren eignet sich die Epistasievarianz nicht, um die *Schwierigkeit eines Problem(exemplar)s* an sich zu beschreiben: Durch geeignete *Substitution* der Variablen (gleichbedeutend mit einer Transformation der Achsen des Suchraums bzw. mit einer alternativen Kodierung) ist es möglich, ihren Wert zu beeinflussen. Beasley et al. [BBM93b] schlagen beispielsweise vor, Probleme mit vielen Nebenbedingungen („*all-or-nothing-problems*“) in Sub-Probleme mit zusätzlichen lokalen Nebenbedingungen (*local constraints*) aufzuteilen, deren Lösungen individuell evaluiert werden können (*local fitness*) und mit problemspezifischen Algorithmen zu einer Gesamtlösung zusammengesetzt werden (*merging algorithm*). Dadurch wird der Lösungsraum des Problems zwar vergrößert, die Epistasie kann jedoch reduziert werden. Die Höhe der Epistasie ist also nicht vom Problem(exemplar) selbst, sondern nur von der Lösungsrepräsentation im Optimierungsverfahren abhängig. Allerdings ist die Suche nach einer Transformation im Sinne einer *Epistasie-freien* Kodierung der Entscheidungsvariablen mindestens ebenso aufwändig wie die Lösung des Exemplars selbst [LV90].

Aber auch bei fester Lösungsrepräsentation ist die Epistasievarianz als Eignungsmaß für populationsbasierte Heuristiken (speziell GA) nur bedingt geeignet: Es lassen sich etwa Problembeispiele mit hoher Epistasie finden, bei denen mit einem GA leicht ein globales Optimum gefunden wird (vgl. z.B. [MC92] und [NV99]). Außerdem lassen sich unterschiedlich schwer zu optimierende Funktionen mit derselben Epistasie konstruieren [Jan00, S.34ff].

Kritik an Davidors Berechnungsmethode gründet vor allem auf folgenden Ansatzpunkten:

Kritikpunkt 1: Die Epistasievarianz misst nur die *Stärke* der Interaktion von Variablen, nicht die *Richtung* [RW99]. ϵ_P^2 misst nicht, in welchem Maße die (lokalen und globalen) Optima der Zielfunktion f an den gleichen Stellen wie die ihrer Näherung erster Ordnung ξ liegen und in wie weit das Gefälle der Funktion f an denselben Stellen in dieselbe Richtung wie das der Funktion ξ zeigt. Insbesondere kann mit ϵ_P^2 nicht ausgedrückt werden, ob die Variablen untereinander nur die Stärke oder auch die Richtung ihres Einflusses auf die Zielfunktion beeinflussen.

Kritikpunkt 2: Die Berechnung der Schätzwerte $\xi(s)$ erlaubt nur die Messung der Stärke *paarweiser Interaktionen insgesamt. Mehrstellige Interaktionen* werden dabei nicht getrennt, sondern nur gemittelt erfasst.

Beasley et al. [BBM93] schlagen eine eigene Definition der Epistasie im Kontext Genetischer Algorithmen vor und unterscheiden in diesem Zusammenhang drei *Level der Interaktion* von (hier binär kodierten) Entscheidungsvariablen: Level 0 („no interaktion“) liegt vor, wenn eine Änderung des Wertes einer Entscheidungsvariablen unabhängig von den Werten anderer Variablen immer die gleiche Änderung des Zielfunktionswertes zur Folge hat. Bei Funktionen des Interaktionslevels 1 (milde Interaktion) ist die Änderung des Zielfunktionswertes zwar verschieden stark,

weist aber immer in dieselbe Richtung. Epistasie (Interaktionslevel 2) liegt nach Beasley et al. erst dann vor, wenn neben der Stärke auch die Richtung beeinflusst wird. Diese alternative Definition der Epistasie zielt auf den ersten Kritikpunkt ab. Anstatt auf die *Stärke* der Interaktionen unter den Variablen wird das Augenmerk auf die *Richtung* dieser Interaktionen gelenkt.

Eher in Richtung des zweiten Kritikpunktes geht der Vorschlag von Manela und Campbell [MC92]. Sie schlagen vor, statt der Stärke der *paarweisen* Interaktionen eher die *Anzahl der einwirkenden Variablen* zu messen. So soll eher die *Ordnung* der Interaktionen in den Vordergrund treten. Diese Ordnung wird bei der Epistasie-Definition von Davidor nicht beachtet, da nur die Stärke der paarweisen Interaktionen gemessen wird. Allerdings wäre das Konzept der Epistasie-Varianz im Prinzip auf Interaktionen höherer Ordnungen übertragbar. Für dreistellige Interaktionen würde sich $\xi_2(s)$ dann beispielsweise als Mittelwert der Fitness-Werte aller Lösungen berechnen, bei denen jeweils *zwei* Entscheidungsvariablen mit dem gleichen Wert wie im Lösungsvorschlag s belegt sind.

Eine Erweiterung des Konzepts der Epistasie schlagen Seo, Choi und Moon vor [SCM04]. Sie führen *interne* und *externe* Epistasie für Partitionen von Entscheidungsvariablen ein. Als interne Epistasie bezeichnen sie die gewichtete Summe der Epistasie von Entscheidungsvariablen innerhalb der jeweiligen Untermengen, wobei jedes Gewicht mit der *Signifikanz* der Untermenge korrespondiert. Die Signifikanz einer Menge von Entscheidungsvariablen spiegelt den Informationswert dieser Entscheidungsvariablen für den Fitness-Wert insgesamt wieder (Entropie). Die externe Epistasie dagegen bezieht sich auf Entscheidungsvariablen aus unterschiedlichen Untermengen. Abbildung A.2 zeigt eine Partition der Entscheidungsvariablen in drei Untermengen V_1, V_2 und V_3. Epistatische Variablenpaare sind durch Kanten gekennzeichnet. Die blauen Kanten kennzeichnen die interne, rote Kanten die externe Epistasie.

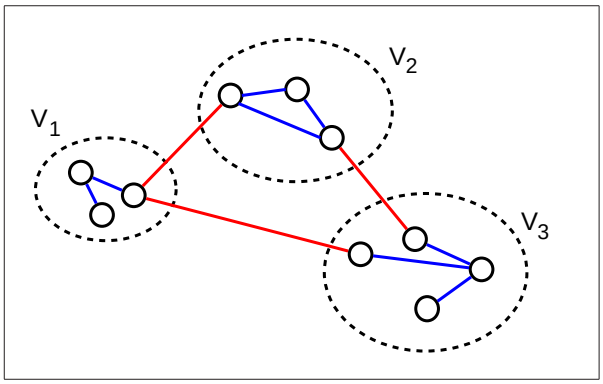


Abbildung A.2: *Externe und interne Epistasie nach Seo et al. [SCM04]*

A.7 Beweisskizze für die NFL-Theoreme

Die Richtigkeit der NFL-Theoreme wird schon intuitiv schnell klar: In einer *beliebigen* Fitnesslandschaft können bisher besuchte Lösungen nicht als Anhaltspunkte für die Güte übriger Lösungen dienen. Die Wahrscheinlichkeit, im Lösungsraum direkt neben einer guten Lösung eine weitere gute Lösung zu finden, ist nicht höher als die, dort auf eine schlechte zu stoßen; analoges gilt für die Rekombination guter Lösungen. Der Beweis von Wolpert und Macready soll hier nur kurz skizziert werden, eine ausführliche Beweisführung findet sich in [WM97].

Wolpert und Macready fassen einen Optimierungsalgorithmus a als Abbildung evaluierter Punkte d im Lösungsraum X auf einen einzelnen neuen, bisher nicht besuchten und als nächstes zu besuchenden Punkt x auf:

$$a : d \mapsto \{x | x \in X \backslash d\}$$

Sie definieren eine Auswahl d_m von m Punkten, die in der Reihenfolge, in der sie generiert wurden, geordnet sind:

$$d_m \equiv \{(d_m^x(1), d_m^y(1)), ..., (d_m^x(m), d_m^y(m))\}$$

$d_m^x(i)$ bezeichnet die Belegung der Entscheidungsvariablen des i-ten Punkts („Lösungsvektor“), $d_m^y(i)$ den zugehörigen Wert der Fitness-Funktion. Eine universelle Verbesserungsheuristik erzeugt im Laufe der Optimierung eine geordnete Menge d^x von Lösungsvorschlägen, die jeweils durch Auswertung der Fitness-Funktion evaluiert werden. Die Folge von Fitness-Werten, die die Zielfunktion nach m Schritten zurückgeliefert hat, kann als d_m^y geschrieben werden. Auf diese Folge von Fitness-Werten zielt der Beweis ab: Wolpert und Macready zeigen, dass die Wahrscheinlichkeit dafür, dass ein Algorithmus eine bestimmte Folge $d_m^y(i)$ produziert, für jedes Paar von Algorithmen a_1 und a_2 gleich hoch ist, wenn man über alle möglichen Fitness-Funktionen mittelt:

$$\sum_f P(d_m^y | f, m, a_1) = \sum_f P(d_m^y | f, m, a_2)$$

Der Beweis wird per Induktion geführt. Zunächst zeigen Wolpert und Macready, dass das Theorem für $m = 1$ gilt, dass also die Wahrscheinlichkeiten für die Produktion einer (Start-)Lösung mit Fitness d_1^y gemittelt über alle Zielfunktionen unabhängig vom Algorithmus a ist (Induktionsanfang). Der Induktionsschritt basiert auf der Tatsache, dass die bei gleich-wahrscheinlichem Auftreten *aller* Zielfunktionen die bisherige Performanz eines Algorithmus beim Schritt m – d.h. die Folge von Fitness-Werten d_m^y bisher evaluierter Lösungsvorschläge – keinen Einfluss hat auf die Fitness des als nächstes (im Schritt m+1) produzierten Lösungsvorschlags $d_{(m+1)}^y(m+1)$. Da sie über *alle* Zielfunktionen *mitteln*, setzen sie für die Gültigkeit ihrer Theoreme implizit das gleich-wahrscheinliche Auftreten aller möglichen Zielfunktionen voraus.

A.8 Spezielle Bearbeitungsverfahren für Traveling Salesman Probleme

Verweise zu genaueren Beschreibungen der hier zusammengestellten sowie weiterer Ansätze finden sich beispielsweise auf der TSPBIB-Homepage von Pablo Moscato [TSP96]. Stützle et al. vergleichen in [SGLR00] speziell „naturanaloge“ Heuristiken (*nature inspired heuristics*) wie Evolutionäre Algorithmen auf einer größeren Menge von TSP-Exemplaren.

Nearest-Neighbour-Heuristik: Für metrische TSP kleinerer Größe liefert die sehr einfache Nearest-Neighbour-Konstruktionsheuristik oft relativ gute Ergebnisse: Ausgehend von einer beliebigen Stadt wird immer zur verbleibenden Stadt „gereist“, die momentan am nächsten liegt, bis nur noch die Ausgangsstadt übrig bleibt.

k-opt-Heuristik: Die k-opt-Verbesserungsheuristik besteht in der lokalen Suche in der problemspezifischen k-change-Nachbarschaft (vgl. Abschnitt 4.2.1). Durch Anwendung einer solchen Heuristik entsteht eine so genannte *k-optimale* Rundreise, d.h. eine Rundreise, die nicht mehr durch einen Tausch von höchstens k Kanten verbessert werden kann.

Lin-Kernighan-Heuristik: Die Lin-Kernighan-Heuristik wird in Abschnitt A.1.2 näher erläutert. Statt eines fest vorgegebenen k wie beim k-opt-Verfahren werden hier jeweils *variabel* viele Kanten ausgetauscht – die Heuristik wird daher auch als *variable k-opt-Heuristik* bezeichnet.

Branch and Bound Als *lower-Bound*-Funktion für die Anwendung eines Branch-and-Bound-Verfahrens (vgl. Abschnitt 4.6.1) bietet sich beispielsweise die Zielfunktion des mit dem TSP korrespondierenden Zuordnungsproblems, bei dem die Bildung von Subtouren erlaubt ist. Das Zuordnungsproblem kann mit Aufwand $O(n^3)$ gelöst werden (vgl. [BT85]).

Ameisenalgorithmen: TSP stellen den klassischen Anwendungsfall für die von Dorigo [DMC91, Dor92] eingeführten Ameisenalgorithmen dar. Virtuelle „Ameisen“ nutzen ihre Kenntnis über die Entfernungen zwischen den Städten aus, indem sie „momentan besser sichtbare“, d.h. vom aktuellen Standpunkt aus nähere Städte bevorzugen. Die Ameisen kommunizieren über so genannte Pheromone miteinander und können so Informationen über günstige (Teil-)wege austauschen (vgl. Abschnitt 4.7.2).

A.9 Begriffe aus der Agententheorie

A.9.1 Der Agentenbegriff, Eigenschaften von Agenten

Das deutsche Wort „Agent“ leitet sich etymologisch aus dem lateinischen Verb *agare* (=handeln) ab. In den Wirtschaftswissenschaften bezeichnet man mit „Agent“ jemanden, der im Auftrag oder als Stellvertreter eines anderen handelt. In der Informatik wird der Begriff in ähnlicher Bedeutung für Hard- und Softwaresysteme verwendet, allerdings haben sich hier je nach Arbeitsschwerpunkt teilweise unterschiedliche Auffassungen bezüglich der Beschaffenheit von Agenten gebildet [FG96].

Wooldridge und Jennings [WJ95] identifizieren vier Basiseigenschaften von Hardware- oder (häufiger) Softwaresystemen, für die der Begriff „Agent“ in einer allgemeineren Form verwendet wird:

Autonomie: Agenten haben Kontrolle über ihren Zustand und eigene Handlungen und können daher auch ohne direkte Fremdeinwirkung agieren. Sie entscheiden also gewissermaßen „selbstständig“, welche Aktionen sie – z.B. in Abhängigkeit ihrer Sensorwerte – durchführen.[6] Meist wird die Auswahl auszuführender Handlungen von einem individuellen, veränderbaren *Zustand* beeinflusst. Dazu kann auch ein *Gedächtnis* zählen, in dem beispielsweise frühere Aktionen und Umgebungszustände abgelegt werden.

Soziale Fähigkeiten: Agenten *interagieren* untereinander und/oder mit Menschen, beispielsweise über eine spezielle Agenten-Kommunikations-Sprache (engl.: *agent communication language*) [GK94]). In Systemen mit mehreren Agenten ist auch eine Kommunikation über die gegenseitige Wahrnehmung von Zuständen und Aktionen möglich.

Reaktivität: Agenten reagieren auf Veränderungen in ihrer Umgebung. Die Aktionen eines Agenten sind also ohne Kenntnis des Umgebungszustandes nicht vorhersehbar sondern werden durch die spezielle *Situation* bestimmt, in denen sich der Agent befindet. Implizit ist damit ein weiteres wichtiges Merkmal, die so genannte „*Situiertheit*“, ebenfalls genannt: Mit Situiertheit bezeichnet man das „sich Befinden“ des Agenten in einer bestimmten Umgebung, deren Zustand er ganz oder teilweise über Sensoren wahrnimmt und über Effektoren verändern kann. Diese Umwelt kann eine reale Umgebung sein, wenn es sich beispielsweise um eine Art Robotersystem handelt. So genannte *Software-Agenten* handeln dagegen in „virtuellen Umgebungen“ wie Dateisystemen oder dem Internet. Russel und Norvig [RN03] definieren Agenten über diese Interaktionen in ihrer Umgebung: „*An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators*“ (S.32). Umgebungen können hinsichtlich ihrer Zugänglichkeit, ihrer Determiniertheit und ihrer Dynamik klassifiziert werden [RN03, Klu01].

Proaktivität: Agenten können auch „selbständig“ zielgerichtet handeln. Sie reagieren also nicht nur auf Veränderungen in ihrer Umgebung sondern agieren auch „von sich aus“ um bestimmte Ziele zu erreichen. In diesem Zusammenhang wird oft auch die „Rationalität“ als Eigenschaft von Agenten genannt. Ein rationaler Agent verfolgt bestimmte Ziele und führt

[6] Klügel [Klu01] unterscheidet dabei Kontroll- und Verhaltensautonomie, je nachdem ob der Agent nur die Ausführung seines Verhaltensprogramms oder die Programmierung seines Verhaltens insgesamt kontrolliert (S.15).

möglichst die Aktionen aus, die ihn diesen Zielen näher bringt. Dafür benötigt der Agent ein Art von Performanz-Maß, das die Güte des momentanen bzw. potenziellen zukünftigen Zustands misst – ähnlich der Zielfunktion bei Optimierungsproblemen. Rationale Agenten versuchen dann, Aktionen durchzuführen, die dieses Performanz-Maß maximieren.

Der engere Agentenbegriff – so Wooldridge und Jennings [WJ95] – wird vor allem von Wissenschaftlern im Bereich der Künstlichen Intelligenz (KI) verwendet für Computersysteme, bei denen versucht wird, über die genannten Eigenschaften hinaus „menschliche“ Konzepte – insbesondere im mentalen Bereich („Wissen“, „Glauben“, „Emotion“ usw.) – zu imitieren. Man spricht in diesem Zusammenhang auch von „anthropomorphen“ Agenten.

A.9.2 Agententypen

Je nach Ausprägung der beschriebenen Eigenschaften sowie ihrer Struktur und der Art und Weise ihrer Interaktion mit der Umwelt können unterschiedliche Agententypen differenziert werden. Russel und Norvig [RN03] beispielsweise unterscheiden fünf Agententypen (S. 44ff):

Einfache Reflexagenten ohne internen Zustand (*simple reflex agents*): Diese Agenten werten Sensordaten aus und erhalten dadurch ein (meist unvollständiges) Bild ihrer Umwelt. Sie wählen ihre Aktionen dann ausschließlich in Abhängigkeit bestimmter Situations-Aktions-Regeln aus.

Reflexagenten mit internem Zustand (*model based agents*): Solche Agenten verfügen über einen internen Zustand, in dem sie Informationen über die Umwelt ablegen können. Nach Auswertung ihrer Sensordaten passen sie diesen Zustand zunächst an, bevor sie Aktionen in Abhängigkeit dieses Zustandes auswählen. Im Unterschied zu einfachen Reflexagenten können also auch länger zurückliegende Zustände der Umwelt in die Aktionsauswahl einfließen.

Zielbasierte Agenten (*goal based agents*): Zielbasierte Agenten verfügen über eine explizite Repräsentation ihrer Ziele und wählen Aktionen so aus, dass diese Ziele erreicht werden. Dazu können sie beispielsweise auch komplexere Aktionssequenzen planen und sind so Reflexagenten mit ihren einfachen Situations-Aktions-Regeln überlegen.

Nützlichkeitsbasierte Agenten (*utility based agents*): Sie verfügen über eine „utility function“ (Nützlichkeitsfunktion), um Ziele nicht nur effektiv, sondern auch effizient zu erreichen. Außerdem können sie in Situationen, in denen sie auch mit langen Aktionssequenzen kein Ziel direkt erreichen können, wenigstens lokale Verbesserungen ihres Zustandes bewirken.

Lernende Agenten (*learning agents*): Sie versuchen, über die Zeit Wissen über die Nützlichkeit bestimmter Aktionen bzw. Aktionssequenzen in bestimmten Situationen anzusammeln und dieses Wissen bei der Auswahl zukünftiger Aktionen zu nutzen. Dazu erhalten die Agenten nach der Durchführung von Aktionen „Feedback“-Informationen und können dann beispielsweise ihre Situations-Aktions-Regeln entsprechend anpassen.

Anhang B

Die wichtigsten Klassen des Pakets *Genetic Conference* im Detail

In diesem zweiten Teil des Anhangs sollen die wichtigsten Attribute und Methoden der Klassen aus den unterschiedlichen Komponenten des Pakets *Genetic Conference* im Detail erläutert werden. Dabei wird komponentenweise vorgegangen. Mit Rücksicht auf die Implementierung wurden für die verwendeten Datentypen die Bezeichnungen der Programmiersprache *Java* verwendet.

B.1 Klassen der Komponente *POOL*

Abbildung B.1 zeigt den Aufbau der POOL-Komponente im Überblick. Die zentrale Klasse stellt der Vorschlag *(Proposal)* dar. Vorschläge werden in einem *ProposalPool*-Objekt gespeichert. In einer Schlüssel-Wert-Tabelle wird zu den bereits evaluierten Vorschlägen der entsprechende Zielfunktionswert eingetragen. ProposalPool realisiert die Schnittstelle *MaipulateablePool*, über die auf den Vorschlagspool zum Selektieren, Einfügen und Entfernen von Vorschlägen zugegriffen werden kann. Dazu stehen die Methoden *addNewProposal*, *removeProposalAt*, *selectProposalAt* und *getFitnessOf* bereit.

Der Vorschlag (POOL.Proposal)

In einem Vorschlag werden einerseits die Belegungen der Entscheidungsvariablen in zweistufigen Zuordnungstabellen gehalten (vgl. Abschnitt 6.2.3). Daneben werden noch folgende weitere Informationen über die Entstehung des Vorschlags und den Stand seiner Evaluierung gespeichert:

***ParentProposal*:** Für jeden Vorschlag wird eine Referenz auf einen „Vorgänger"-Vorschlag gespeichert. Entstand der Vorschlag durch Rekombination zweier Vorgänger, so werden Referenzen auf beide gespeichert.

***ModifiedAgent,ModificationState, ModificationPlan*:** Der Vorschlag trägt eine Referenz auf den Agenten, der den Vorschlag als letztes modifiziert und dann in die Konferenz eingebracht hat. Zusätzlich werden Informationen über den Plan, den der Agent für die Erzeugung des Vorschlags ausgeführt hat, sowie die Bezeichnung seiner Situation, die zur

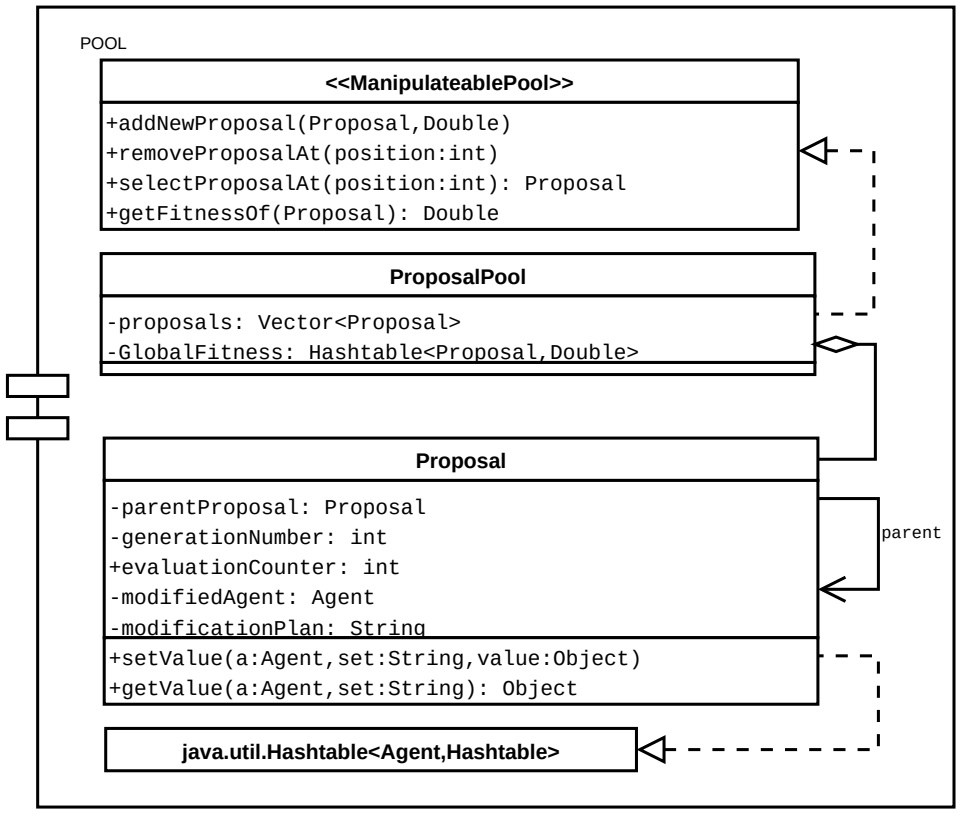


Abbildung B.1: *Die Komponente POOL*

Auswahl des Plans führte, beim Vorschlag eingetragen. Auf diese Weise kann im Nachhinein die Güte-Differenz durch die Veränderungen vom alten zum neuen Vorschlag ermittelt werden. Diese Differenz wird für die Anpassung der Wahrscheinlichkeiten in der Situation-Plan-Matrix („lernende Agenten“) benötigt.

***GenerationNumber*:** Beim Vorschlag wird die Nummer der Generation eingetragen, in der er erzeugt wurde. Die Generationsnummer dient unter anderem zur Ermittlung des Alters eines Vorschlags, d.h. der Anzahl der Generationen, die sich dieser Vorschlag bereits in der Konferenz hält. Darüber kann auch abgeschätzt werden, ob sich das Verfahren bereits in einer Phase befindet, in der Verbesserungen nur noch mit geringer Wahrscheinlichkeit zu erwarten sind.

***EvaluationCounter*:** Die Evaluation der Lösungsvorschläge erfolgt parallel durch die Agenten. Um später entscheiden zu können, ob bereits alle Agenten einen Lösungsvorschlag bewertet haben, wird dieser Zähler nach einer erfolgten lokalen Bewertung inkrementiert. Entspricht der Wert des Zählers der Anzahl der an der Konferenz beteiligten Agenten, so kann der Moderator einen aggregierten Gesamt-Gütewert berechnen.

Die beschriebenen Informationen können auch über den Konferenzmonitor des Framework angezeigt werden und so zu Analysezwecken dienen.

B.2 Klassen der Komponente *GA*

Die *GA*-Komponente bietet Methoden für die *CTRL*- und die *MAS*-Komponente zur Durchführung von „GA-spezifischen“ Operationen auf dem Vorschlagspool, die in einem Interface *ProposalPoolManager* zusammengefasst sind. Die Klasse *GA-OP* implementiert diese Methoden: Die Methode *addProposal* fügt einen neuen Vorschlag in den Vorschlagspool ein. Die Methode *generationStep* führt einen Generationsübergang bei den Vorschlägen durch. Sie verdrängt dabei zunächst Vorschläge aus dem Pool und füllt ihn mit neuen Vorschlägen auf. Beide Methoden greifen auf die Schnittstelle *ManipulateablePool* zu. Die Referenz auf ein Objekt, das dieses Interface implementiert, wird bei Aufruf der *init*-Methode des GA-Op-Objekts übertragen und als Attribut *pool* gespeichert. Außerdem wird die Methode *selectProposal* implementiert, die mittels Turnierselektion unter den Vorschlägen der aktuellen Generation einen Vorschlag mit hohem Gesamt-Gütewert auswählt.

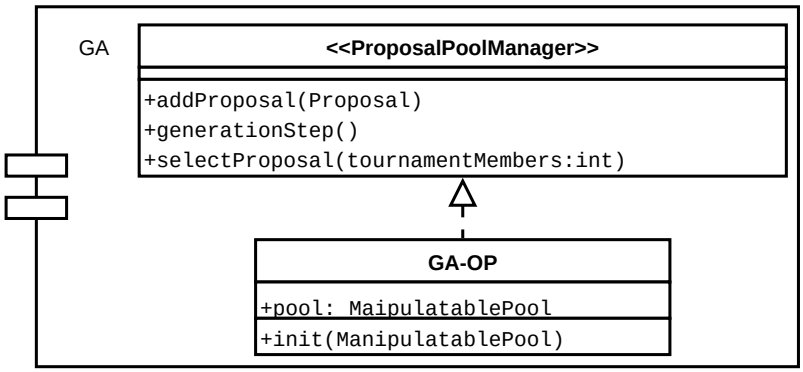


Abbildung B.2: *Die Komponente GA*

B.3 Klassen der Komponente *MAS*

Abbildung B.3 zeigt die Klassen der MAS-Komponente im Überblick. Aufgrund des Umfangs sind nicht alle Attribute und Methoden der zentralen Agenten-Klasse abgebildet.

Die Auftrags-Klasse (*Job*) und das Interface *JobReceiver*

Ein Auftrag (*Job*) besteht entweder in der Initialisierung, der partiellen Bewertung oder der Variation eines Vorschlags oder in der Bildung einer Koalition auf Grundlage dieses Vorschlags. Ein Auftrags-Objekt verfügt über zwei Attribute: Einen *Modus*, der die Art des Auftrags festlegt und einen *Vorschlag*, der bewertet oder variiert werden soll bzw. der die Grundlage der zu bildenden Koalition darstellt.

Das Interface *JobReceiver* bietet die Methode *addJob(Job)*, mit der ein neuer Auftrag in die Warteschlange des jeweiligen (Agenten-)Objekts eingetragen werden kann, das die Schnittstelle realisiert.

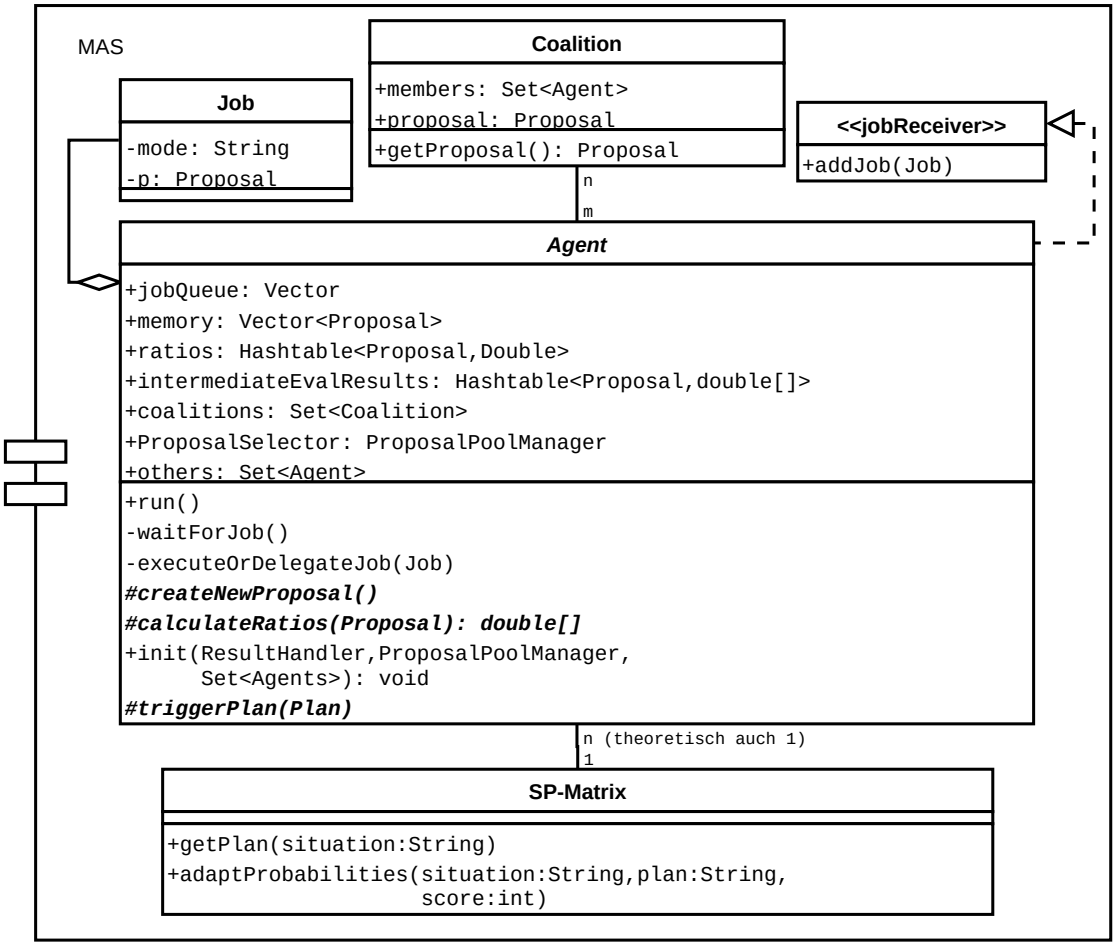


Abbildung B.3: *Die Komponente MAS*

Der Agent (*Agent*)

Agenten werden im GC-Paket als abstrakte Klasse modelliert. Einige ihrer Methoden sind also nicht ausformuliert, sondern müssen für konkrete Anwendungsbeispiele jeweils speziell implementiert werden – es handelt sich dabei um die Methoden zur Initialisierung, Variation und partiellen Bewertung von Vorschlägen sowie zur Auswahl potenzieller Koalitionspartner.

Die wichtigsten Attribute der Agenten-Objekte
Als Attribute verfügen die Agenten in jedem Fall über eine Referenz auf das problemspezifische Umgebungs-Objekt aus der ENV-Komponente. Des Weiteren halten sie jeweils eine Tabelle mit Evaluations-Zwischenergebnissen (*intermediateEvalResults*) sowie mit den Güte-Kennziffern (*ratios*) für bereits evaluierte Vorschläge.
Die Situations-Plan-Matrix *(SP-Matrix)* mit den Wahrscheinlichkeiten für die Auswahl bestimmter Pläne zur Verbesserung von Vorschlägen in bestimmten Situationen ist ein weiteres Attribut eines Agenten. Bei den Kardinalitäten der Assoziation zwischen Agent und SA-Matrix kommen theoretisch zwei Varianten in Frage: Entweder die Agenten nutzen eine gemeinsame SP-Matrix

(n zu 1), oder jeder Agent greift auf eine eigene Matrix zu (1 zu 1)[1]. Auswirkungen hat dieser Unterschied, wenn die Wahrscheinlichkeiten in der Matrix von den Agenten angepasst werden können („lernende Agenten“): Während der Agent bei der 1-zu-1-Variante nur aus dem Feedback zu eigenen Aktionen lernen kann *(„decentralized learning“)*, können die Agenten bei der n-zu-1-Variante auch die Ergebnisse der übrigen Agenten in ihre Aktionsauswahl einbeziehen *(„centralized learning“)*. Implizit findet dann eine Kommunikation über eine Art gemeinsamer Wissensbasis statt. Die Implementierung bietet zunächst nur letztere Variante an.

Ein weiteres Attribut der Agenten ist eine Warteschlange, in die zu bearbeitende Aufträge (Jobs) vom Moderator eingetragen werden können. Zuletzt verfügen die Agenten über eine Referenz auf die Menge von Koalitionen, an denen sie beteiligt sind. Bei der problemspezifischen Implementierung der Agenten für konkrete Optimierungsprobleme ist natürlich die Aufnahme zusätzlicher Attribute möglich.

Die wichtigsten problemübergreifend ausformulierten Methoden
Die folgenden Methoden sind bereits problemübergreifend ausformuliert, können aber natürlich bei Bedarf überschrieben werden:

void init(ResultHandler, ProposalPoolManager, Set⟨Agent⟩): Mit dieser Methode wird der Agent initialisiert. Über die Referenzen auf den *ResultHandler* und den *ProposalPoolManager*, die als Attribute gespeichert werden, kann später auf die erforderlichen Schnittstelle zur *CTRL*-Komponente und zur *GA*-Komponente zugegriffen werden.

void run(): In der *run*-Methode des Agenten-Prozesses werden in einer Schleife die Methoden *waitForJob* und *executeOrDelegateJob* ausgeführt – und zwar solange, bis die Konferenz beendet wird.

void waitForJob(): Die Methode *waitForJob* lässt den Agenten-Prozess auf einen Auftrag (*Job*) in der Warteschlange warten.

void executeOrDelegateJob(): Die Methode *executeOrDelegateJob* holt zunächst den ersten Job aus der Warteschlange zur Bearbeitung oder Delegation und entfernt diesen Auftrag aus der Warteschlange. Je nach Art des Auftrags werden im Anschluss die folgenden weiteren Methoden aufgerufen:

- Die Methode *createNewProposal* wird aufgerufen, sofern es sich um einen Auftrag zur Initialisierung eines neuen Vorschlags handelt.
- Beinhaltet der Auftrag dagegen die partielle Bewertung eines bestehenden Vorschlags, so werden die (abstrakten) Methoden *calcIntermediateEvalResults* und *calcRatios* für diesen Vorschlag in dieser Reihenfolge aufgerufen.
- Im Fall eines Auftrags zur Variation eines Vorschlags entscheidet der Agent zunächst, ob er den Vorschlag selbst variieren möchte, weil er Verbesserungspotential erkennt. Falls nicht, ermittelt der Agent durch ein Turnierverfahren einen anderen Agenten, der mit dem Vorschlag (möglichst) noch unzufrieden ist. Im Anschluss wird die Me-

[1] Um eine *parallele* Bearbeitung mehrerer Problemexemplare eventuell unterschiedlicher Optimierungsprobleme zu ermöglichen, kommt eine Modellierung als *statisches* Attribut dagegen nicht in Frage

thode *modifyProposal* des ursprünglichen oder entsprechend ausgewählten Agenten-Exemplars ausgeführt.[2]

- Eine letzte Auftragsart besteht in der Aufforderung zur Bildung einer Koalition. In diesem Fall ruft der Agent die Methode *buildCoalition* auf – oder „delegiert" den Auftrag wiederum an einen Agenten, der diesmal jedoch anhand *hoher* Zufriedenheit ausgewählt wird.

Proposal createProposalByModification(Proposal): In dieser Methode wird zunächst der als Argument übergebene Ausgangs-Vorschlag kopiert und der Agent trägt sich als Urheber beim neuen Vorschlag ein. Im Anschluss entscheidet der Agent anhand vorgegebener Wahrscheinlichkeiten, ob eine Mutation dieser Kopie oder eine Rekombination der Kopie mit einem weiteren Vorschlag durchgeführt werden soll. In beiden Fällen wird die variierte Kopie zuletzt als neu in die Konferenz einzubringender Vorschlag zurückgeliefert.

Im ersten Fall wird die Methode *mutateProposal* mit der Kopie als Argument aufgerufen. Dann wird beim Vorschlag eingetragen, in welcher Situation sich der Agent befand und welcher Plan zur Mutation angewendet wurde – diese Informationen dienen später zur Anpassung der Wahrscheinlichkeiten in der SP-Matrix. Im zweiten Fall wird mittels Aufruf der Methode *selectSatisfyingProposal* zunächst ein zweiter Vorschlag ausgewählt, mit dem die Kopie rekombiniert werden soll. Im Anschluss wird die Methode *recombineProposals* aufgerufen, die beiden Vorschlägen werden als Argumente übergeben.

void mutateProposal(Proposal): Zur lokalen Variation eines Vorschlags wird zunächst die abstrakte Methode *getSituation* aufgerufen, um auf Grundlage der Güte-Koeffizienten die Situation des Agenten bezüglich des Vorschlags zu ermitteln. Durch Aufruf der Methode *selectPlan* wird dann für die Situation ein Plan zur Verbesserung ausgewählt. Dieser Plan wird dann durch Aufruf der Methode *triggerPlan* auf den Vorschlag angewendet.

Proposal selectSatisfyingProposal(): Diese Methode wählt aus den bereits evaluierten Vorschlägen einen aus, mit dem der Agent besonders zufrieden war. Eine solche Auswahl eines (lokal) zufriedenstellenden Vorschlags wird für die Rekombination mit einem insgesamt gut bewerteten, jedoch lokal nicht zufriedenstellenden Vorschlags benötigt.

Zunächst entscheidet der Agent (anhand vorgegebener Wahrscheinlichkeiten), ob dieser Vorschlag aus dem Pool bereits evaluierter Vorschläge, dem Gedächtnis des Agenten oder von einer Koalition ausgewählt wird, die der Agent im Vorfeld eingegangen ist. Um frühzeitige Konvergenz des Verfahrens zu vermeiden, wird dann ein nicht-deterministisches Verfahren für die konkrete Auswahl verwendet: Anstatt immer den Vorschlag zurück zu liefern, bei dem die Zufriedenheit am höchsten war, findet eine Wettkampfselektion statt.

[2] Aus Sicht einer „ästhetischen" Modellierung der Agenten mit eigenständigen Prozessen zur Initialisierung, Evaluierung und Variation von Vorschlägen sollte die Variation eines Vorschlags durch einen Agenten eigentlich auch vom Prozess des entsprechenden Agenten-Exemplars erfolgen. Möglich wäre beispielsweise, den Variations-Job einfach an den entsprechenden Agenten weiterzugeben. Technisch hat eine solche Modellierung jedoch den Nachteil, dass dann die Bearbeitung der Variationsaufträge voraussichtlich nicht gleichmäßig auf die Agenten-Prozesse verteilt und damit der Performanzvorteil durch Verteilung dieser Prozesse auf unterschiedliche Prozessoren nicht optimal ausgenutzt würde. Daher erfolgt die tatsächliche Ausführung eines solchen Auftrags immer im Prozess des ursprünglich „beauftragten" Agenten-Exemplars, gewissermaßen „in Stellvertretung" des tatsächlich ausgewählten Agenten.

void buildCoalition(Proposal): Diese Methode wird nach der partiellen Bewertung eines Vorschlags durch alle Agenten vom Konferenzleiter aufgerufen und stößt einen Agenten an, auf Grundlage des Vorschlags eventuell eine Koalition *(Coalition)* zu bilden. Der Agent ruft dazu die abstrakte Methode *getPossibleCoalitionMembers* auf, die eine Menge potenzieller Koalitionsmitglieder zurück liefert. Anschließend wird durch Aggregation der Werte, die die Funktion *getAggregatedSatisfaction* dieser Agenten zurück liefert, ein Gesamt-Zufriedenheitswert für diese Agentengruppe berechnet. Ist diese Zufriedenheit ausreichend hoch[3], so wird eine neue Koalition auf Grundlage des Vorschlags gebildet.

double receiveFeedback(Proposal): Diese Methode untersucht, wie sich die Güte eines Vorschlags im Bezug zur Güte des/der Vorgängervorschlags/-vorschläge verhält, ob sich also durch den Plan, der zur Erzeugung eines Vorschlags geführt hat, Verbesserungen oder Verschlechterungen ergeben haben. Entsprechend dieser Veränderungen werden die Wahrscheinlichkeitswerte im *SP-Matrix*-Objekt durch Aufruf der Methode *adaptProbabilities* angepasst.

double getLocalEvaluationValue(double[]): Die Methode *getLocalEvaluationValue* liefert einen Aggregationswert der Evaluations-Zwischenergebnisse zurück, die durch die abstrakte Methode *calcIntermediateEvalResults* berechnet wurden. Die vordefinierte Methode summiert diese Zwischenergebnisse mit gleicher Gewichtung. Sollen die Zwischenergebnisse auf andere Weise aggregiert werden so muss die Methode überschrieben werden.

double getAggregatedSatisfaction(Proposal): Mit dieser Methode wird aus den in der entsprechenden Tabelle abgelegten Güte-Koeffizienten (*ratios*) zu einem Vorschlag ein aggregierter Zufriedenheits-Wert bestimmt, indem das Produkt der Koeffizienten (Gleichgewichtung) berechnet wird. Für eine andere Form der Aggregation der Koeffizienten muss die Methode überschrieben werden. Dabei muss darauf geachtet werden, dass der resultierende Zufriedenheitswert zwischen 0 (unzufrieden) und 1 (zufrieden) skaliert sein muss.

Abstrakte (problemspezifische) Methoden
Die folgenden Methoden der Agenten sind abstrakt und müssen vom Anwender erst für das konkrete Problembeispiel implementiert werden.

abstract Proposal createNewProposal(): Diese Methode erstellt einen neuen Vorschlag – beispielsweise durch Ausführung einer problemspezifischen Konstruktionsheuristik – und muss vom Anwender implementiert werden.

abstract double[] calcIntermediateEvalResults(Proposal): Diese Methode dient dazu, die Zwischenergebnisse bei der lokalen Bewertung eines Vorschlags zu berechnen. Die Ausführungen dieser (möglicherweise rechenintensiven) Methode für einen Vorschlag soll für alle Agenten nebenläufig erfolgen. Damit die Zwischenergebnisse später ohne zeitaufwändige Neuberechnung abrufbar sind, werden die in der lokalen Tabelle *intermediateEvalResults* zwischengespeichert. Sie werden bei der Berechnung von Güte-Koeffizienten und bei der Berechnung der Gesamtgüte eines Vorschlags herangezogen.

abstract double[] calculateRations(double[], Proposal): Mit dieser Methode werden die Güte-Koeffizienten zu einem Vorschlag berechnet. Diese Berechnung kann wiederum nebenläufig

[3]Entsprechende Schwellenwerte können einerseits vom Anwender eingestellt werden, andererseits können bisherige Koalitionen mit geringerer Gesamt-Zufriedenheit „verdrängt“, d.h. gelöscht werden.

erfolgen. Für die Berechnung der Koeffizienten kann die Methode auf die Zwischenergebnisse der partiellen Evaluation zurückgreifen. Die berechneten Koeffizienten werden für spätere Abrufe ebenfalls zwischengespeichert, und zwar in der Zuordnungstabelle *ratios*.

abstract Set ⟨Agent⟩ getPossibleCoalitionMembers(Proposal): Über diese Methode kann der Anwender spezifizieren, welche Agenten auf Grundlage eines gegebenen Vorschlags eine Koalition miteinander eingehen können.

abstract String getSituation(Proposal): Diese Methode ermittelt die Situation eines Agenten bezüglich eines Lösungsvorschlags auf Grundlage der Güte-Koeffizienten. Anhand der Situation wird im Anschluss ein Plan zur lokalen Verbesserung des Vorschlags ausgewählt.

abstract void triggerPlan(Proposal, Plan): Mit dieser Methode spezifiziert der Anwender den Ablauf der Anwendung eines ausgewählten Plans zur Mutation eines Vorschlags.

abstract void recombineProposals(Proposal, Proposal): Über diese Methode spezifiziert der Anwender, wie ein Vorschlag, bei dem mit lokalem Verbesserungspotential zu rechnen ist, mit einem lokal zufriedenstellenden Vorschlag rekombiniert werden kann.

Die Koalition (*Coalition*)

Jeder Agent kann mit der Methode *buildCoalition* eine Koalition unter Agenten auf Grundlage eines Vorschlags bilden. Eine solche Koalition besteht daher aus einer Menge von Agenten, die als Attribut *members* gespeichert wird, sowie einem „Basis-Vorschlag“, auf den ebenfalls eine Referenz als Attribut gehalten wird. Über die Methode *getProposal* kann der Basis-Vorschlag abgefragt werden.

Die Situation-Plan-Matrix (*SP-Matrix*)

Das Situation-Plan-Matrix-Objekt *(SP-Matrix)* repräsentiert eine Wahrscheinlichkeitsmatrix. Für jede Situation, in der sich ein Agent bezüglich eines Vorschlags befinden kann, wird für jeden der möglichen Pläne zur lokalen Variation des Vorschlags angegeben, mit welcher Wahrscheinlichkeit dieser Plan zur Anwendung kommen soll.[4] Die Wahrscheinlichkeiten können vom Anwender im Vorfeld über entsprechende Inputdateien spezifiziert und im Laufe des Optimierverfahrens variiert werden („Lernprozesses“). Die Klasse verfügt über folgende Methoden:

String selectPlan(String): Diese Methode wählt für eine bestimmte Situation aufgrund der Wahrscheinlichkeiten in der Matrix einen Plan zur Mutation eines Vorschlags aus und liefert diesen zurück.

void adaptProbabilities(String, String, double): Diese Methode passt die Wahrscheinlichkeit für die zukünftige Auswahl eines bestimmten Plans in einer bestimmten Situation entsprechend der erreichten Verbesserung (bzw. Verschlechterung) bei der letzten Ausführung an.

[4] Wie bereits erwähnt, könnten für die Agenten jeweils eigene Matritzen verwendet werden. Bisher arbeiten aber alle Agenten auf derselben Matrix.

B.4 Klassen der Komponente *CTRL*

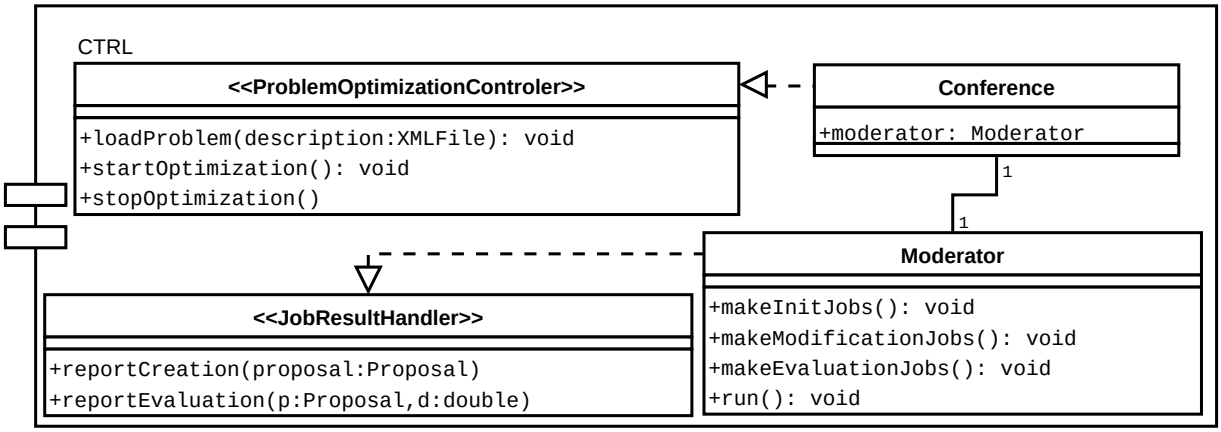


Abbildung B.4: *Die Komponente CTRL*

Die Konferenz (*Conference*) und das Interface *ProblemOptimizationControler*

Die *CTRL*-Komponente stellt über Methoden des Interface *ProblemOptimizationControler* Schnittstellen zur Initialisierung eines Problemexemplars und zum Start eines Optimierungslaufs zur Verfügung. Dieses Interface wird von der *Conference*-Klasse implementiert. Einem *Conference*-Objekt wird durch Aufruf der Methode *loadProblem* mit einem XML-File als Parameter die nötige Information zur Initialisierung aller beteiligten Objekte übertragen.[5] Der Aufruf der Methode *startOptimization* startet dann einen solchen Lauf. Dazu wird die *run*-Methode des Diskussionsleiters und der Agenten-Objekte angestoßen. Ein Optimierungslauf kann über die Methode *stopOptimization* abgebrochen werden. Abbildung B.4 zeigt die Klassen der *CTRL*-Komponente im Überblick.

Der Diskussionsleiter (*Moderator*) und das Interface *JobResultHandler*

Ein *Moderator*-Objekt wird über eine *init*-Methode initialisiert, die später notwendigen Referenzen auf die Schnittstelle zur *GA*-Komponente (ProposalPoolManager) und auf die *MAS*-Komponente (Job-Receiver) werden als Parameter übergeben. Wie für die Agenten, so wird auch für den Konferenzleiter ein eigener Prozess gestartet. Die Moderator-Klasse bietet dazu eine *run*-Methode. In dieser Methode wird zunächst die Methode *createInitJobs* aufgerufen. Dann stößt der Moderator die Selektion unter den Vorschlägen an, indem er auf die Schnittstelle GA-Operator zugreift (*generationStep*). Im Anschluss werden Aufträge zur Modifikation bestehender Vorschläge an die Agenten vergeben (*makeModificationJobs*). Der Konferenzleiter stellt außerdem die Methoden *reportNewProposal* und *reportEvaluation* für die Agenten bereit. Mit diesen Methoden können die Agenten die erfolgreiche Bearbeitung eines entsprechenden Auftrags melden.

[5] Die entsprechenden Schnittstellen wurden in Abbildung 6.1 der Übersichtlichkeit halber vernachlässigt.

B.5 Klassen der Komponente *ENV*

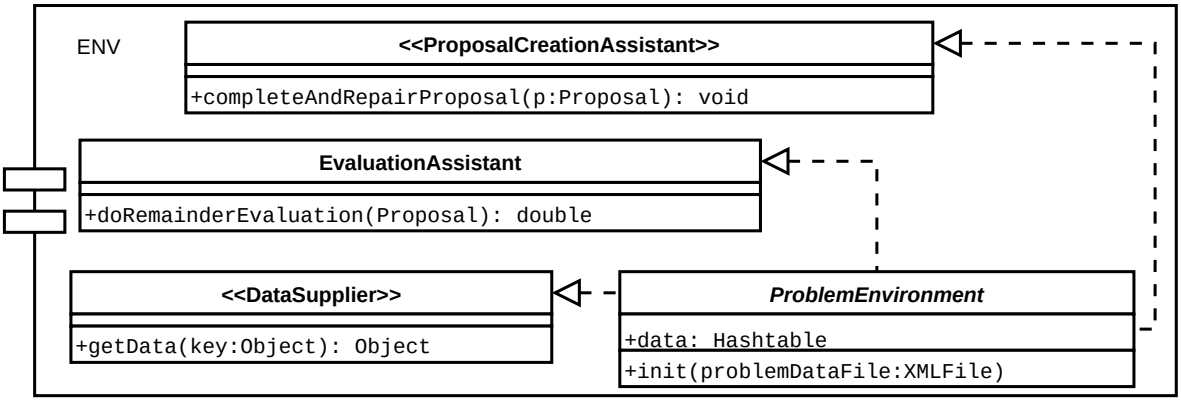


Abbildung B.5: *Die Komponente ENV*

Die Umgebungs-Komponente bietet die Schnittstellen *DataSupplier*, *ProposalCreationAssistant* und *EvaluationAssistant*. Die Klasse *ProblemEnvironment* implementiert diese Schnittstellen und ist abstrakt. Drei Methoden müssen für die Anwendung des Framework auf problemspezifische Weise implementiert werden.

init(XMLFile): Über diese Methode wird die Umgebung der Agenten initialisiert. Dazu wird ein XML-File übergeben. Problemspezifische Daten können in einer Zuordnungstabelle *data* abgelegt werden. Diese Daten können dann später von den Agenten über die Methode *getData* abgerufen werden.

completeAndRepairProposal(Proposal): Diese Methode wird aufgerufen, nachdem ein Vorschlag (durch Initialisierung oder Variation) neu erstellt wurde. Hier kann die Einhaltung bestimmter Nebenbedingungen überprüft werden und möglicherweise können Reperaturmechanismen eingesetzt werden, wenn dies nicht durch die Agenten selbst erfolgen kann bzw. soll. Ebenso ist es möglich, zusätzliche „Sätze“ von Entscheidungsvariablen mit Werten zu belegen. Dieses Vorgehen wurde beispielsweise beim Anwendungsfall „Standortplanung“ gewählt (vgl. Abschnitt 3.2). Die Methode kann auch verwendet werden, um einen Vorschlag im Sinne eines Memetischen Algorithmus durch Anwendung einer lokalen Suchheuristik (vgl. Lokale Suche, Abschnitt 4.2.2) in ein lokales Optimum umzuwandeln.

remainderEvaluation(Proposal): Über diese Methode kann ein zusätzlicher Evaluationswert für einen Vorschlag berechnet werden, wenn eine Dekomposition der Zielfunktion in partielle Funktionen (die von den Agenten ausgewertet werden) nicht vollständig möglich war. Der berechnete Wert fließt dann mit in die Gesamtbewertung des Vorschlags ein.

Literaturverzeichnis

[Bau99] Baumgärtel, H.: Verteiltes Lösen von Constraint-Problemen in Multiagenten-Systemen zur optimierten Planung in einer Fließfertigung. Dissertationen zur Künstlichen Intelligenz, Band 209. Infix-Verlag, 1999.

[BBM93] Beasley, D., Bull, D., Martin, R.: An Overview of Genetic Algorithms: Part 2, Research Topics. In: University Computing 15 (4), 1993; S. 170-181.

[BBM93b] Beasley, D., Bull D., Martin, R.: Reducing epistasis in combinatorial problems by expansive coding. In: [For93]; S. 400-407.

[BGS02] Barvinok, A., Gimadi, E., Serdyukov, A.: The maximum Traveling Salesman Problem. In: [GP02]; S. 585-607.

[BK05] Bearpark, K., Keane, A. J.: The use of collective memory in genetic programming. In: [Jin05]; S. 15-36.

[BL95] Brimberg, J., Love, R.: Estimating Distances. In: [Dre95]; S. 9-32.

[Blu05] Blunck, S.: Modellierung und Optimierung von Hub-and-Spoke-Netzen mit beschränkter Sortierkapazität. Wissenschaftliche Berichte des Institutes für Fördertechnik und Logistiksysteme der Universität Karlsruhe (TH), Band 65, 2005.

[BRJ06] Booch, G., Rumbaugh, J., Jacobson, I.: Das UML Benutzerhandbuch. Addison-Wesley, 2006.

[Bru96] Bruns, R: Wissensbasierte genetische Algorithmen - Integration von genetischen Algorithmen und Constraint - Programmierung zur Lösung kombinatorischer Optimierungsprobleme. Infix-Verlag, 1996.

[BT85] Balas, E., Toth, P.: Branch and bound methods. In: [LLRS85]; S. 361-401.

[Buc05] Buch, P.: Modellierung des Distributionsnetzes eines großen deutschen Hausgeräteherstellers zur Optimierung der Umschlagpunktstruktur mit Hilfe von GIS-Technologie. Diplomarbeit, LMU München, 2005.

[CDG99] Corne, D., Dorigo, M., Glover, F.: New Ideas in Optimization. McGraw Hill, 1999.

[Cer83] Cerny, V.: Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. In: Journal of Optimization Theory and Applications 45 (1), 1985; S. 41-51.

[Chr85] Christofides, N.: Vehicle Routing. In: [LLRS85]; S. 431-448.

[Coo64] Cooper, L.: Heuristic Methods for Location-Allocation Problems. In: SIAM Review 6 (1), 1964; S. 37-53.

[Cor91] Corkill, D.: Blackboard Systems. In: AI Expert 6 (9), 1991; S. 40-47.

[DAGP90] Deneubourg, J.-L., Aron, S., Goss, S., Pasteels, J.-M.: The self-organizing exploratory pattern of the argentine ant. In: Journal of Insect Behaviour 3, 1990; S. 159-168.

[Dav85] Davis, L.: Job Shop Scheduling with Genetic Algorithms. In: Grefenstette, J. (Hrsg.): Proceedings of the First International Conference on Genetic Algorithms, 1985; S. 136-140.

[Dav90] Davidor, Y.: Epistasis Variance: A Viewpoint on GA-Hardness. In: Rawlins, G. (Hrsg.): Proceedings of the First Workshop on Foundations of Genetic Algorithms, July 1990. Morgan Kaufmann, 1991; S. 23-35.

[Dav91] Davis, L. (Hrsg.): Handbook of Genetic Algorithms. Van Nostrand Reinhold, 1991.

[Daw76] Dawkins, R.: The Selfish Gene. Oxford University Press, 1976.

[DC99] Dorigo, M.; Di Caro, G.: The Ant Colony Optimization Meta-Heuristic. In: [CDG99]; S. 11-32.

[DD96] Domschke, D., Drexl, A.: Logistik: Standorte. 4. Auflage, Oldenbourg, 1996.

[DG97] Dorigo, M., Gambardella, L. M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. In: IEEE Transactions on Evolutionary Computation 1, 1997; S. 53-66.

[Dij59] Dijkstra, E. W.: A Note on two Problems in Connexion with Graphs. In: Numerische Mathematik 1, 1959; S. 269-271.

[DMC91] Dorigo, M., Maniezzo, V., Colorni, A.: Ant System: An autocatalytic optimizing process. Working paper Nr. 91-016 Revised, Politecnico di Milano, 1991.

[DMC96] Dorigo, M., Maniezzo, V., Colorni, A.: The Ant System: Optimization by a colony of cooperating agents. In: IEEE Transactions on Systems, Man and Cybernetics Part B, Band 26, 1996; S. 29-41.

[Dom89] Domschke, H.: Schedule synchronization for Public Transit Networks. In: OR Spektrum 11, Springer, 1989; S. 17-24.

[Dor92] Dorigo, M.: Optimization, learning and natural algorithms. Dissertation, Politecnico di Milano, 1992.

[Dre95] Drezner, Z.: Facility Location. A Survey of Applications and Methods. Springer, 1995.

[Dre95b] Drezner, T.: Competitive Facility Location in the Plane. In: [Dre95]; S. 285-300.

[DS90] Dueck, G., Scheuer, T.: Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. In: Journal of Computational Physics 90 (1), 1990; S. 161-175.

[DSW93] Dueck, G., Scheuer, T., Wallmeier, H.M.: Toleranzschwelle und Sintflut: neue Ideen zur Optimierung. In: Spektrum der Wissenschaft 3, 1993; S. 42-51.

[DS03] Dorigo, M., Stützle, T.: The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In: Glover F., Kochenberger, G. (Hrsg.): Handbook of Metaheuristics. International Series in Operations Research and Management Science, 2003; S. 251-285.

[Dur99] Durfee, E. H.: Distributed Problem Solving and Planning. In: [Wei99]; S. 121-164.

[EK04] Engelhardt-Funke O., Kolonko, M.: Analysing Stability and Investments in Railway Networks Using Advanced Evolutionary Algorithms. In: International Transactions in Operational Research 11 (4), 2004; S. 381-394.

[FF93] Fonseca C. M., Fleming, P. J.: Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: [For93]; S. 416-423.

[FF95] Fonseca C. M., Fleming, P. J.: An overview of evolutionary algorithms in multiobjective optimization. In: Evolutionary Computation, Band 3, 1995; S. 1-16.

[FG96] Franklin, S., Graesser, A.: Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. In: Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96), Springer, 1996; S. 21-35.

[Fog00] Fogel, D. B.: Evolutionary Computation. IEEE Press, 2nd edition, 2000.

[For93] Forrest, S. (Hrsg.): Proceedings of the Fifth International Conference on Genetic Algorithms. Morgan Kaufmann, 1993.

[Fou85] Fourman, M.P.: Compaction of symbolic layout using genetic algorithm. In: [Gre85]; S. 141-153.

[FOW66] Fogel, L. J., Owens, A. J., Walsh, M. J.: Artificial Intelligence through Simulated Evolution. John Wiley, 1966.

[Gal03] Galanda, M.: Automated Polygon Generalization in a Multi Agent System. Dissertation, Universität Zürich, 2003.
http://www.carto.net/papers/martin_galanda/martin_galanda_polygon_generalization.pdf (Stand: Mai 2007).

[GK94] Genesereth, M., Ketchpel, S.: Software Agents. In: Communication of the ACM 37 (7), 1994; S. 48-53.

[Glo86] Glover, F.: Future paths for integer programming and links to artificial intelligence. In: Computers and Operations Research 13 (5), 1986; S. 533-549.

[GMC95] Gosh, A., McLafferty, S., Craig, S.: Multifacility Retail Networks. In: [Dre95]; S. 301-330.

[GMS04] Di Gaspero, L., Mizzaro, S., Schaerf, A.: A multiagent architecture for distributed course timetabling. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2004), 2004; S. 471-473.

[Gol89] Goldberg, D.: Genetic Algorithms in Search, Optimisation and Machine Learning, Addison-Wesley, 1989.

[GP02] Gutin, G., Punnen, A. (Hrsg.): The Traveling Salesman problem and its variations, Kluwer Academic Publishers, 2002.

[Gre85] Grefenstette, J. (Hrsg.): Genetic Algorithms and Their Applications. Proceedings of the First International Conference on Genetic Algorithms. Lawrence Erlbaum, 1985.

[Gre87] Grefenstette, J.: Incorporating problem specific knowledge into genetic algorithms. In: Davis, L. (Hrsg.): Genetic Algorithms and Simulated Annealing, Morgan Kaufmann, 1987; S. 42-60.

[Guc97] Guckert, M.: Anschlussoptimierung in öffentlichen Verkehrsnetzen - Graphentheoretische Grundlagen, objektorientierte Modellierung und Implementierung. Dissertation, Universität Marburg, 1997.

[GV99] Goldberg, D., Voessner, S.: Optimizing Global-Local Search Hybrids. In: Banzhaf, W., Daida, J., Eiben, A., Garzon, M., Hoovar, V., Jakiela, M., Smith, R. (Hrsg.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), Morgan Kaufmann, 1999; S. 220-228.

[GW03] Galanda, M., Weibel, R.: Ein Multiagentensystem zur Generalisierung von Polygonmosaiken in thematischen Karten. In: Koch, A., Mandl, P. (Hrsg.): Multi-Agenten-Systeme in der Geographie. Klagenfurter Geographische Schriften, Heft 23, 2003; S. 139-166.

[Hel00] Helsgaun, K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal of Operational Research 126, 2000; S. 106-130.

[Hes02] Hesse, R.: Agentensysteme in der Geographischen Handelsforschung. Diplomarbeit, Technische Universität München, 2002.

[HGH05] Hesse, W., Guckert, M., Hesse, R.: OptiTakt - A tool for developing and evaluating periodic timetables. In: Proceedings of the first International Seminar on Railway Operations Modelling, RailDelft, 2005.

[Hol75] Holland, J.: Adaption in Natural and Artificial Systems. University of Michigan Press, 1975.

[Hol92] Holland, J.: Adaption in Natural and Artificial Systems. University of Michigan Press, 1992. (Neuauflage von [Hol75])

[Hro03] Hromkovič, J.: Algorithmics for Hard Problems. Springer, 2003.

[IT03] Igel, C., Toussaint, M.: Results on No-Free-Lunch Theorems for Optimization. ArXiv Computer Science e-prints, 2003.

[Jak04] Jakob, W.: Eine neue Methodik zur Erhöhung der Leistungsfähigkeit evolutionärer Algorithmen durch die Integration lokaler Suchverfahren. Wissenschaftliche Berichte Forschungszentrum Karlsruhe 6965, 2004.

[Jan00] Jansen, T.: Theoretische Analyse evolutionärer Algorithmen unter dem Aspekt der Optimierung in diskreten Suchräumen. Dissertation, Universität Dortmund, 2000. http://ls2-www.cs.uni-dortmund.de/~jansen/diss/diss.pdf (Stand: Mai 2007).

[JF95] Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Eshelman, L. (Hrsg.): Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, 1995; S. 184-192.

[Jin05] Jin, Y. (Hrsg.): Knowledge Incorporation in Evolutionary Computation. Springer, 2005.

[Jon95] Jones, T.: Evolutionary Algorithms, Fitness Landscapes and Search. PhD thesis, University of New Mexico, 1995.

[Kau93] Kauffman, S.: The origins of order. Self-organization and selection in evolution. Oxford University Press, 1993.

[KE95] Kennedy, J., Eberhart, R. C.: Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks, 1995; S. 1942-1948.

[KE02] Kolonko, M., Engelhardt-Funke, O.: Mathematische Optimierung in der Praxis. Genetisch optimierte Fahrpläne und Kosten-Nutzen-Analysen für Verkehrsnetze. In: TUContakt, Hochschulzeitschrift der TU Clausthal, Heft 10, 2002; S. 31-34.

[KGV83] Kirkpatrick, S., Gelatt C. D., Vecchi M. P.: Optimization by Simulated Annealing. In: Science, Band 220, Nr. 4598, 1983; S. 671-680.

[KL87] Kauffman, S. A., Levin, S.: Towards a General Theory of Adaptive Walks on Rugged Landscapes. In: Journal of Theoretical Biology 128, 1987, S. 11-45.

[Klu01] Klügl, F.: Multiagentensimulation. Konzepte, Werkzeuge, Anwendung. Addison-Wesley, 2001.

[Koe04] Koehler, G.: Conditions that Obviate the No Free Lunch Theorems for Optimization. In: Forthcoming INFORMS Journal on Computing 19 (2), 2007. http://www.cba.ufl.edu/dis/docs/papers/ConditionsThatObviateTheNoFreeLunch-TheoremsForOptimization.pdf (Stand: Mai 2007).

[Koz92] Koza, J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, 1992.

[KS04] Kräuchi, C., Stöckli, U.: Mehr Zug für die Schweiz. Die Bahn-2000-Story. AS Verlag, 2004.

[KSSSW07] Knopp, S., Sanders, P., Schultes, D., Schulz, F., Wagner, D.: Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In: Workshop on Algorithm Engineering and Experiments (ALENEX), 2007 (to appear). http://i10www.ira.uka.de/schultes/hwy/distTable.pdf (Stand: Mai 2007).

[Law01] Lawler, E.: Combinatorial Optimization. Networks and Matroids. Dover Publications, 2001.

[LD60] Land, A. H., Doig, A. G.: An automatic method of solving discrete programming problems. In: Econometrica 28; S. 497-520.

[Lie04] Liebchen, C.: Symmetry for Periodic Railway Timetables. In: Electronic Notes in Theoretical Computer Science 92, Nr. 1, 2004; S. 34-51.

[Lie05] Liebchen, C.: Fahrplanoptimierung im Personenverkehr - muss es immer ITF sein? In: Eisenbahntechnische Rundschau 11, 2005; S. 689-702.

[LK73] Lin, S., Kernighan, B. W.: An effective heuristic algorithm for the traveling salesman problem. In: Operations Research 21, 1973; S. 498-516.

[LLRS85] Lawler E., Lenstra J., Rinnooy Kan A., Shmoys D. (Hrsg.): The Traveling Salesman Problem. John Wiley, 1985.

[LM04] Liebchen, C., Möhring, R.: The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables and Beyond. To appear in: LNEMS Proceedings of the 9th Conference on Computer-Aided Scheduling of Public Transport 2004. ftp://ftp.math.tu-berlin.de/pub/Preprints/combi/Report-020-2004.pdf (Stand: Mai 2007).

[LMSK63] Little, J. D. C., Murty, K. G., Sweeney, D. W., Karel, C.: An algorithm for the traveling salesman problem. In: Operations Research 21, 1963; S. 972-989.

[LPW04] Liebchen, C., Proksch, M., Wagner, H.: Performance of Algorithms for Periodic Timetable Optimization. To appear in: LNEMS Proceedings of the 9th Conference on Computer-Aided Scheduling of Public Transport 2004. http://fugazi.engr.arizona.edu/caspt/liebchen_proksch_wagner.pdf (Stand: Mai 2007).

[LV90] Liepins G. E., Vose, M. D.: Representational issues in genetic optimization. Journal of Experimental and Theoretical Artificial Intelligence 2, 1990; S. 101-115.

[MAK07] Michiels, W., Aarts, E., Korst, J.: Theoretical Aspects of Local Search. Monographs in Theoretical Computer Science, Springer, 2007.

[May88] Mayr, E.: Toward a new philosophy of biology. Observations of an evolutionist. Belknap Press of Harvard University Press, 1988.

[MC92] Manela, M., Campbell, J.: Harmonic analysis, epistasis and genetic algorithms. In: Manner, R., Manderick, B. (Hrsg.): Proceedings of the 2nd conference on parallel problems solving from nature (PPSN-II), Elsevier, 1992; S. 59-66.

[Mer00] Merz, P.: Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies. Dissertation, Universität Siegen, 2000. http://www.ub.uni-siegen.de/pub/diss/fb12/2001/merz/merz.pdf (Stand: Mai 2007).

[Mey04] Meyer, D.: Modellbasierte Mehrzieloptimierung mit Neuronalen Netzen und Evolutionsstrategien. Dissertation, Technische Universität Ilmenau, 2004. http://deposit.ddb.de/cgi-bin/dokserv?idn=974935875&dok_var=d1&dok_ext=pdf&filename=974935875.pdf (Stand: Mai 2007).

[MF00] Michalewicsz, Z., Fogel, D. B.: How to Solve It: Modern Heuristics. Springer, 2000.

[Mic96] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, 1996.

[Mos89] Moscato, P.: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. C3P Report 826, Caltech Concurrent Computation Program, 1989. http://www.densis.fee.unicamp.br/~moscato/papers/bigone.ps (Stand: Mai 2007).

[Mos99] Moscato, P.: Memetic Algorithms: A short Introduction. In: [CDG99]; S. 219-234.

[MR04] Milano, M., Roli, A.: MAGMA: a multiagent architecture for metaheuristics. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 2004; S. 925-941.

[MWS91] Manderick, B., de Weger, M., Spiessens, P.: The genetic algorithm and the structure of the fitness landscape. In: Belew, R., Booker, L. (Hrsg.): Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, 1991; S. 143-150.

[Nis94] Nissen, V.: Evolutionäre Algorithmen: Darstellung, Beispiele, betriebswirtschaftliche Anwendungen. Deutscher Universitäts-Verlag, 1994.

[Nis97] Nissen, V.: Einführung in Evolutionäre Algorithmen. Optimierung nach dem Vorbild der Evolution. Vieweg, 1997.

[NK98] Naudts, B., Kallel, L.: Some Facts about So Called GA-Hardness Measures. Technical Report 379, Centre de Mathematiques Appliquees, Palaiseau, 1998. ftp://barbes.polytechnique.fr/pub/RI/1998/naudts_kallel_379.mars.ps.gz (Stand: Mai 2007).

[NSV97] Naudts, B., Suys, D., Verschoren, A.: Epistasis as a basic concept in formal landscape analysis. In: Bäck, T. (Hrsg.): Proceedings of the 7th international conference on genetic algorithms, Morgan Kaufmann, 1997; S. 65-72.

[NV99] Naudts, B., Verschoren, A.: Epistasis and Deceptivity. In: Bulletin of the Belgian Mathematical Society 6, 1999; S. 147-154.

[Par99] Parunak, H. V. D.: Industrial and Practical Applications of Distributed AI. In: [Wei99]; S. 377-421.

[Pee03] Peeters, L.: Cyclic Railway Timetable Optimization. PhD Thesis, University Rotterdam, 2003.
https://dspace.ubib.eur.nl/retrieve/569/EPS-2003-022-LIS+9058920429+PEETERS.pdf (Stand: Mai 2007).

[PS78] Papadimitriou, C, Steiglitz, K.: Some Examples of difficult traveling salesman problems. In: Operations Research 26, 1978; S. 434-443.

[PS82] Papadimitriou, C, Steiglitz, K.: Combinatorial Optimization. Algorithms and Complexity, Prentice Hall, 1982.

[QRS98] Quick, R. J., Rayward-Smith, V., Smith, G. D.: Fitness Distance Correlation and Ridge Functions. In: Eiben, A. E., Bäck, T., Schoenauer, M., Schwefel, H. P. (Hrsg.): Parallel Problem Solving from Nature (PPSN) V. Lecture Notes in Computer Science 1498, Springer, 1998; S. 77-86.

[Rec73] Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Fromman-Holzboog, 1973.

[Rec94] Rechenberg, I.: Evolutionsstrategie '94. Frommann Holzboog, 1994. (Neuauflage von [Rec73])

[RHS04] Rauh, J., Hesse, R., Spichale, C.: Einsatz agentenbasierter Mikrosimulation für kleinräumige Bevölkerungsprognosen - am Fallbeispiel von Regensburg. In: GeoBit/GIS 12/2004; S. 27-34.

[RN03] Russell, S. J., Norvig, P.: Artifical Intelligence: A Modern Approach. Prentice Hall, 2nd edition, 2003.

[RW99] Reeves, C., Wright, C.: Genetic algorithms and the design of experiments. In: Davis, L., Vose, M., De Jong, K., Whitley, D. (Hrsg.): Evolutionary Algorithms: The IMA Volumes in Mathematics and its Applications, Band 111, Springer Verlag, 1999; S. 207-226.

[Scha85] Schaffer, J. D.: Multiple objective optimization with vector evaluated genetic algorithms. In: [Gre85]; S. 93-100.

[Schw77] Schwefel, H. P.: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Birkhäuser, 1977.

[SCM04] Seo, D., Choi, S., Moon, B. R.: New Epistasis Measures for Detecting Independently Optimizable Partitions of Variables. In: Genetic and Evolutionary Computation - GECCO 2004, Lecture Notes in Computer Science 3103, Springer, 2004; S. 150-161.

[SGLR00] Stützle, T., Grün, A., Linke, S., Rüttger, M.: A Comparison of Nature Inspired Heuristics on the Traveling Salesman Problem. In: Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN-VI), Lecture Notes in Computer Science 1917, 2000; S. 661-670.

[SH00] Stützle, T., Hoos, H.: MAX-MIN Ant System. Journal of Future Generation Computer Systems 16(8), 2000; S. 889-914.

[SHF94] Schöneburg, E., Heinzmann, F., Feddersen, S.: Genetische Algorithmen und Evolutionsstrategien. Addison-Wesley, 1994.

[SRSF91] Sycara, K., Roth, S., Sadeh, N., Fox, M.: Distributed Constrained Heuristic Search. IEEE Transactions on System, Man, and Cybernetics, Band. 21, 1991; S. 1446-1461.

[Sta02] Stadler, P.: Fitness landscapes. In: Lassig, M. and Valleriani, A. (Hrsg.): Biological Evolution and Statistical Physics. Springer, 2002; S. 187-207.

[SVW01] Schumacher, C., Vose, M. D., Whitley, L. D.: The No Free Lunch and Problem Description Length. In: Spector et al. (Hrsg.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann, 2001; S. 565-570.

[SW99] Sen, S., Weiss, G.: Learning in Multiagent Systems. In: [Wei99]; S. 259-298.

[TSP96] Moscato, P.: TSPBIB Home Page.
http://www.ing.unlp.edu.ar/cetad/mos/ TSPBIB_home.html (Stand: Mai 2007).

[TSP06] National TSP Library, Georgia Institute of Technology.
http://www.tsp.gatech.edu/ (Stand: Mai 2007).

[Vla03] Vlassis, N.: A Concise Introduction to Multiagent Systems and Distributed AI. University of Amsterdam, 2003.
http://www.science.uva.nl/˜vlassis/cimasdai (Stand: Mai 2007).

[Vog95] Voget, S.: Aspekte genetischer Optimierungsalgorithmen: mathematische Modellierung und Einsatz in der Fahrplanerstellung. Dissertation, Universität Hildesheim, 1995.

[Wei90] Weinberger, E.: Correlated and uncorrelated fitness landscapes and how to tell the difference. In: Biological Cybernetics, Band 63, 1990; S. 325-336.

[Wei99] Weiss, G. (Hrsg.): Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence. MIT Press, 1999.

[WJ95] Wooldridge, M.J., Jennings, N. R.: Intelligent Agents: Theory and Practice. In: The Knowledge Engineering Review 10 (2), 1995; S. 115-152.

[WM95] Wolpert, D. H., Macready, W. G.: No Free Lunch Theorems for Search, Technical Report SFI-TR-95-02-010. Santa Fe Institute, Sante Fe, 1995.

[WM97] Wolpert, D. H., Macready, W. G.: No Free Lunch Theorems for Optimization In: IEEE Transactions on Evolutionary Computation, Band 1, 1997; S. 67-82.

[Wri32] Wright, S.: The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution. In: Proceedings of The Sixth International Congress of Genetics, Band 1, 1932; S. 356-366.

[WW03] Wagner, D., Willhalm, T.: Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. In: Di Battista, G., Zwick, U. (Hrsg.): Proceedings of the 11th European Symposium on Algorithms ESA 2003; S. 776-787.

[WW05] Whitley, D., Watson, J.: Complexity Theory and the No Free Lunch Theorem. In: Burke, E., Kendall, G.: Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Kluwer, 2005; S. 317-339.

[WW07] Wagner, D., Willhalm, T.: Speed-Up Techniques for Shortest-Path Computations. In: Thomas, W., Weil, P. (Hrsg.): Symposium on Theoretical Aspects of Computer Science 2007, LNCS 4393; S. 23-36.

[YH00] Yokoo, M., Hirayama, K.: Algorithms for Distributed Constraint Satisfaction: A Review. In: Autonomous Agents and Multi-Agent Systems, Band 3, Nr. 2, 2000; S. 198-212.