Effizienzuntersuchungen in der „Vorschlagen und Vertauschen“-Methode für Zuordnungsprobleme

Diplomarbeit im Fach Informatik

cand. inform. Rainer Herrler

Betreuer

Prof. Dr. Frank Puppe

Dipl. Inform. Ciske Busch

Lehrstuhl für Künstliche Intelligenz

und Angewandte Informatik

Universität Würzburg

Ich erkläre hiermit, die vorliegende Arbeit selbstständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Würzburg, den 14.01.2000

Danksagung

An dieser Stelle möchte ich mich bei denjenigen Personen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben. Zuerst möchte ich den Betreuer dieser Arbeit, Ciske Busch, nennen. Durch seine konstruktive Kritik und Hilfestellung konnte ich schon während meiner Studienarbeit wertvolle Erfahrungen sammeln, die beim Erstellen dieser Arbeit sehr wichtig waren. Ich bedanke mich bei Frank Puppe für viele Ideen während den Besprechungen.

Desweitern bedanke ich mich bei den Mitarbeitern des INKAD-Projektes, besonders Johannes Beck, der mir bei vielen Fragen zu COKE helfen konnte. Auch Frank Forster möchte ich danken für seinen Einsatz, den Nurse-Scheduler noch fertigzustellen. Ich bedanke mich bei Christoph Oechslein, für das Korrekturlesen und weitere Anregungen zu Änderungen. Für das Korrekturlesen in letzter Minute bedanke ich mich bei Jens Jordan. Allen, die mich stets auf meinen cäsarischen Satzbau hingewiesen haben, bin ich zu Dank verpflichtet.

Ohne die Hilfe meiner Schwester und meines Vaters hätte diese Arbeit wohl einige Rechtschreibfehler und Fehler in der Kommasetzung mehr. Ein Dankeschön auch dafür.

Für die hilfsbereite Unterstützung bei meinen Latex-Problemen danke ich Christian Hestermann.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In der vorliegenden Arbeit soll eine Problemlösungsmethode, die sogenannte „Vorschlagen und Vertauschen“-Methode auf ihre Effizienz beim Lösen von Zuordnungsproblemen hin untersucht werden.

Bei Zuordnungsproblemen soll eine Menge von Nachfrageobjekten auf eine Menge von Angebotsobjekten abgebildet werden. Bei dieser Abbildung müssen bestimmte Regeln eingehalten werden (für eine genaue Definition vgl. S. 7). Praktische Beispiele sind die Schulstundenplanung - hier werden Unterrichtsstunden in einen Zeitplan eingeordnet - und die Resourcenbelegungsplanung, bei der Arbeitsaufträge auf Maschinen verteilt werden. Viele dieser Aufgaben sind sehr komplex und sind nur mit sehr viel Erfahrung und unter großem Zeitaufwand von Hand zu lösen. Bisherige Ansätze für Software aus diesem Bereich sind vielfach so gestaltet, daß der Mensch die Aufgabe der Problemlösung übernimmt und vom Computersystem lediglich unterstützt wird.

Der Grund liegt zum einen darin, daß die Probleme häufig so komplex sind, daß sie selbst mit Computerunterstützung nicht mit einfachen Algorithmen (z.B. systematisches Ausprobieren aller Lösungen) in vernünftiger Zeit gelöst werden können, zum anderen, daß das Problem vielfach nicht vollständig spezifiziert werden kann, was einer vollautomatischen Lösungsfindung im Wege steht.

In der geschichtlichen Entwicklung der Zuordnungsprobleme standen zunächst mathematische Lösungsmethoden wie z.B. die SIMPLEX-Methode [Burkard & Derigs 1980] oder der Busacker-Goven-Algorithmus [Busacker & Goven 1961] im Vordergrund. Diese Verfahren schränken jedoch teilweise die Problemstellung sehr ein, die Problemstellungen sind für sie schwer zu formulieren und sie „scheitern jedoch meist an zu großer Komplexität oder zu restriktiven Vorbedingungen der verwendeten Verfahren.“[Hestermann 1997] Der Trend ging deshalb in Richtung der heuristischen Suchverfahren, die auf „einer wesentlich eleganteren Struktur aufbauen“[Poeck 1995] und auch für komplexere Zuordnungsprobleme einsetzbar sind.

Der in dieser Arbeit untersuchte „Vorschlagen und Vertauschen“-Algorithmus [Puppe 1990] (vgl. Seite 14) ist ein solches Suchverfahren. Es wurde bereits in COKE[1], einer universellen Zuordnungs-Shell implementiert und basierend auf dieser existieren einige Anwendungen zu konkreten Zuordnungsproblemen.

Der „Vorschlagen und Vertauschen“-Algorithmus verspricht erfolgreichen Einsatz bei vielen Problemen, in denen ein effektiver Computereinsatz bisher nicht möglich war. Eine Lösungsmethode für Zuordnungsprobleme ist umso besser, je bessere Lösungen sie findet und je schneller sie arbeitet. Eine weitere Verbesserung bzw. mehr Wissen über die Eigenschaften des „Vorschlagen und Vertauschen“-Algorithmus sind daher ein erstrebenswertes Ziel.

1.2 Ziel und Aufbau der Arbeit

Die „Vorschlagen und Vertauschen“-Strategie ist kein bis ins Detail definierter Lösungsalgorithmus, sondern ein Lösungsverfahren, das bei der Implementierung einen großen Spielraum läßt. Die allgemeine Beschreibung läßt sehr viel Freiraum für Variationen. COKE, die bisher einzige Zuordnungs-Shell, die diese Strategie beinhaltet, bietet nur eine kleine Auswahl der vielen möglichen Variationen und kann an einigen Stellen verbessert werden.

Ziel der Arbeit ist es, nach allgemeinen Effizienzuntersuchungen verschiedener Varianten der „Vorschlagen und Vertauschen“-Methode durch die Implementierung ausgewählter Ideen eine Verbesserung bei bestehenden COKE-Anwendungen zu erreichen. COKE soll damit in größerem Rahmen konfigurierbar werden.

Im Theorieteil der Arbeit (Kapitel 3, 4 und 5) werden verschiedene Detailvarianten der „Vorschlagen und Vertauschen“-Methode vorgestellt und bewertet. Um eine möglichst vollständige Betrachtung der Alternativen und Verbesserungen zu erreichen, werden zwei Herangehensweisen angestrebt. Zum einen wird für jeden Abschnitt des V&V überlegt, welche Alternativen es in der Umsetzung dieses Schrittes gibt, zum anderen werden die bekannten Suchparadigmen aus dem Bereich der künstlichen Intelligenz und andere Lösungsstrategien für Zuordnungsprobleme analysiert, um festzustellen, welche Elemente man daraus mit V&V kombinieren könnte.
Die vorliegende Arbeit entstand im Kontext anderer Arbeiten, die sich bereits mit der Effizienz in Lösungsmethoden von Zuordnungsproblemen beschätigten. Diese Arbeiten von [Busch 1997] und [Forster 1999] beschäftigten sich überwiegend mit der Effizienz der graphischen Repräsentation, der Bedienung und den Interaktionsmöglichkeiten. Der Schwerpunkt dieser Arbeit wird dagegen in der Optimierung der Ergebnisse und der effizienten Ausnutzung der Ressourcen Zeit und Speicherplatz liegen.

Die Ergebnisse des Theorieteils bildeten die Grundlage für die Auswahl der Verbesserungen und Erweiterungen, die für COKE im Rahmen dieser Arbeit implementiert wurden (im folgenden *COKE-Extensions* genannt). Das implementierte System wird in Kapitel 7 beschrieben und in Kapitel 8 anhand von zwei bestehenden COKE-Anwendungen evaluiert.

[1] COKE bedeutete COordination with Knowledge-based Expertise. Dieses Akronym hat im Laufe der Entwicklung seine ursprüngliche Bedeutung verloren, wie [Poeck 1995] zur Namensgebung meint.

Kapitel 2

Grundlagen

Dieses Kapitel erklärt zuerst den Begriff *Zuordnungsproblem* und stellt kurz einige Beispiele dieses Problemtyps zur Veranschaulichung vor. Abschnitt 2.2 gibt einen Überblick über die Problemlösungsmethoden für diesen Problemtyp. Die Idee des „Vorschlagen und Vertauschens“ wird in Abschnitt 2.3 erklärt und dort werden auch die Eigenschaften der Methode dargelegt. Die Abschnitte 2.4 bis 2.6 schließlich definieren einige zentrale Begriffe der Themenstellung, die nicht direkt mit „Vorschlagen und Vertauschen“ zusammenhängen, aber in seinem Umfeld häufig in Gebrauch sind und deshalb geklärt werden sollen.

2.1 Zuordnungsprobleme

Verschiedene Probleme des alltäglichen Lebens haben oft scheinbar wenig miteinander zu tun. Bei genauerer Betrachtung und Abstrahierung der Probleme kann man allerdings feststellen, daß sich verschiedene Problemstellungen durchaus in dieselben Problemtypen[1] einordnen lassen und deshalb mit verwandten Problemlösungsmethoden bearbeitet werden können. Im Vordergrund steht natürlich, eine Lösungsmethode für ein gegebenes Problem zu finden. Umso nützlicher ist deshalb eine Methode, mit der sich eine ganze Klasse von Problemen lösen läßt. Abbildung 2.1 zeigt eine Einordnung von Problemtypen in Problemlösungstypen. Im folgenden wird ein solcher Problemtyp, die Zuordnungsprobleme, vorgestellt.

2.1.1 Nomenklatur und Definitionen

Ausgangsmengen eines Zuordnungsproblems sind eine Menge von *Nachfragern* und eine Menge von *Anbietern*. Gesucht ist eine Abbildung der Menge der Nachfrager auf die Menge der Anbieter, wobei verschieden gewichtete *Randbedingungen* berücksichtigt werden sollen. Ich nenne diese Randbedingungen im folgenden auch *Restriktionen*, da sie die Menge der gültigen Abbildung einschränken. Jede Abbildung der Menge der Nachfrager auf die Menge der Anbieter heißt *Zuordnung*. Jeder Nachfrager muß mit einem Anbieter versorgt werden. Anbieter müssen nicht notwendigerweise einem Nachfrager zugeteilt werden, können aber

[1] Die Begriffe Problemtyp und Problemlösungsmethode werden an späterer Stelle genauer definiert. An dieser Stelle genügt die intuitive Vorstellung von diesen Begriffen.

Abbildung 2.1: Einordnung von Problemtypen in die drei Hauptproblemlösungstypen (nach [Puppe 1990]). Der Problemtyp Zuordnung fällt unter den Problemlösungstyp *Konstruktion*, bei dem die Problemlösung aus vorgegebenen primitiven Bausteinen zusammengesetzt wird.

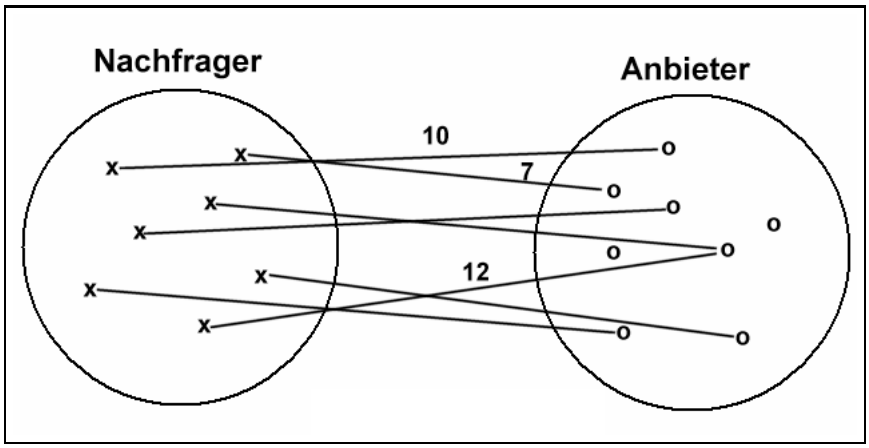

Abbildung 2.2: Zuordnungsprobleme zeichnen sich durch disjunkte Mengen von Anbietern und Nachfragern aus. Gesucht ist eine Abbildung der Menge der Nachfrager auf die Menge der Anbieter. Bei dieser Abbildung (gemeint „Abbildung“ im mathematischen Sinne, nicht die Illustration.) sollen möglichst wenige Restriktionen verletzt werden. Die Zahlen deuten die Verletzungsgewichte bei einzelnen Zuteilungen an. Ziel ist es, die Summe dieser Gewichte zu reduzieren.

auch zugleich mehreren Nachfragern zugeteilt werden. In diesem Falle spricht man von einer *1-n-Zuordnung*. Der Spezialfall, daß jeder Anbieter genau einem Nachfrager zugeteilt wird, heißt *1-1-Zuordnung*. [Forster 1999] unterscheidet zum besseren Verständnis zwischen einer *Zuteilung*, der Abbildung eines einzelnen Nachfragers, und einer *Zuordnung*, der Abbildung aller Nachfrager auf die Anbieter. Darüberhinaus spricht man von einer *partiellen Zuordnung*, wenn man eine eingeschränkte Nachfragermenge betrachtet und bezüglich dieser alle Nachfrager bereits Anbietern zugeteilt sind.

2.1.2 Beispiele für Zuordnungsprobleme

Viel anschaulicher wird diese Definition anhand einiger Beispiele.

- Ressourcenbelegungsplanung

Die Ressourcenbelegungsplanung ist ein Problem aus dem industriellen und betriebswirtschaftlichen Umfeld. Die Aufgabe ist, verschiedene Arbeitsaufträge, die für Fertigungsprozesse benötigt werden, auf Zeitfenster der benötigten Ressourcen (dies kann vorhandenes Personal oder eine geeignete Maschine sein) zu verteilen. Die Fertigungsaufträge bestimmen eine Menge von Operationen, die von der Menge der zur Verfügung stehenden Ressourcen in einem bestimmten Zeitrahmen durchgeführt werden müssen. Das Zuordnungsproblem besteht darin, den Operationen solche Zeitpunkte zuzuordnen, daß die für die Bearbeitung benötigten Ressourcen zur Verfügung stehen. Zu den Randbedingungen zählen somit z.B. Kapazitätsgrenzen von Maschinen, ein Einhalten einer bestimmten Reihenfolge der Operationen und die Berücksichtigung von Lieferterminen. Mit WIZARD wurde an der Universität Würzburg in Zusammenarbeit mit der infor business solutions AG ein auf einer Zuordnungsshell basierendes System zur Ressourcenbelegungsplanung entwickelt. Eine Beschreibung zu WIZARD enthält [Hestermann 1996].

- Personaleinsatzplanung
  Im Pflegebereich und allgemein in Arbeitsbereichen mit Schichtdienst stellt sich das Problem der Personaleinsatzplanung. Bei der Verteilung der Schichten ist auf die Einhaltung gesetzlicher Regelungen, die Wünsche des Personals und soziale Gerechtigkeit zu achten. Soziale Gerechtigkeit bedeutet dabei z.B. gerechte Behandlung bezüglich der Wochenendschichten. Als Zuordnungsproblem betrachtet stellt sich die Personaleinsatzplanung folgenderweise dar: Die Nachfragermenge ist eine Menge von Schichten eines Planungszeitraums, die einer Menge von Arbeitskräften oder Zeitintervallen der Arbeitskräfte zugeordnet werden sollen. Restriktionen bestehen z.B. für die Arbeitszeit pro Woche und die Anzahl der aufeinanderfolgenden Nachtschichten. Ebenfalls zu berücksichtigen sind möglicherweise die Qualifikation des Personals, Urlaubswünsche und persönliche Präferenzen einzelner Arbeitnehmer. In [Herrler 1999] wurde mit PEPSI ein Prototyp zur Dienstplanerstellung bei der Deutschen Bundesbahn vorgestellt. Im Rahmen der Arbeit von [Forster 1999] wurde eine Anwendung zur Einsatzplanung von Pflegepersonal entwickelt.

- Schulstundenplanung
  Am Anfang eines jeden Schuljahres muß aufgrund der veränderten Personal- und Klassensituation erneut ein Schulstundenplan erstellt werden. Ein solcher Plan „legt fest, wann und in welchen Räumen die Unterrichtsveranstaltungen stattfinden sollen. Als Zuordnungsproblem formuliert versucht man den Kursen, die jeweils aus mindestens einem Lehrer, einer Klasse und einem Fach bestehen, geeignete Zeiten des Wochenstundenplans und passende Räume zuzuordnen.“[Busch 1997]. Restriktionen bestehen z.B. über die Verfügbarkeit von Lehrern (es kann kein Lehrer gleichzeitig zwei Unterrichtsstunden halten) und Bindungen an bestimmte Plätze im Zeitplan (Doppelstunden, Werken am Nachmittag). Andere Restriktionen bestehen bezüglich der Raumbelegung, wie „Physikunterricht kann nur im Physikraum gehalten werden“. Mit REST wurde an der Universität Würzburg ein Prototyp zur Schulstundenplanung entwickelt, das erfolgreich an der Hauptschule Heuchelhof in Würzburg getestet wurde. [Poeck 1995]

Gegeben:
- $N := \{n_1, ..., n_n\}$ die Menge der Nachfragerobjekte
- $A := \{a_1, ..., a_a\}$ die Menge der Anbieterobjekte
- $Z := \{\{x_{11}, ..., x_{na}\} \mid \forall_{i,j} x_{ij} \in \{0,1\}$ und $x_{ij} = 1 \Leftrightarrow \exists$ Zuteilung zwischen $n_i \in N$ und $a_j \in A$ $\}$ eine Zuordnung (Menge der Zuteilungen) und
- $K := Z \rightarrow \Re$ eine Kostenfunktion (Summe der Verletzungskosten)

Gesucht:
Suche $z \in Z$ so, daß $K(z)$ minimal ist und jedes $n_i \in N$ genau einem $a_j \in A$ zugeordnet wird.

Abbildung 2.3: Vereinfachte mathematische Definition eines Zuordnungsproblems als Optimierungsproblem (aus [Busch 1997]).

2.1.3 Randbedingungen

Es gibt verschiedene Kategorien von Randbedingungen. Sogenannte *Harte Restriktionen* müssen eingehalten werden, um eine gültige Lösung zu erhalten. *Weiche Restriktionen* sind Präferenzen und dienen als Optimierungskriterium. Im Beispiel der Dienstplanung sind gesetzliche Forderungen und problemimmanente logische Voraussetzungen harte Restriktionen (eine solche wäre: Eine Dienstagsschicht darf nur an einem Dienstag eingeplant werden). Harte Restriktionen werden häufig auch *Konsistenzbedingungen* genannt, da ihre Einhaltung für einen konsistenten (=gültigen) Plan ausschlaggebend ist. Eine Randbedingung, wie der Wunsch nach einer möglichst gleichmäßigen Schichtverteilung oder das Einhalten einer vorgegebenen Ruhezeit zwischen den Schichten, kann eine weiche Restriktionen sein; sie wird zwar gewünscht, es besteht jedoch keine gesetzliche Notwendigkeit für sie. Gesucht ist eine bezüglich der harten Restriktionen gültige und bezüglich der weichen Restriktionen möglichst gute Zuordnung.

Es kann zwischen *einfachen Randbedingungen* und *mehrfachen Randbedingungen* [Busch 1997] unterschieden werden[2]. Einfache Randbedingungen betreffen nur eine Zuteilung wie zum Beispiel die Randbedingung, daß eine Schicht nur dem Wochentag zugeteilt werden darf, für den sie vorgesehen ist. Eine mehrfache Restriktion schränkt dagegen eine Gruppe von Zuteilungen ein wie beispielsweise die Nachtschichtenregelung, die besagt, daß keine vier Nachtschichten (entspricht einer Gruppe von vier Zuteilungen) in Folge liegen dürfen.

Ein weiteres Unterscheidungskriterium der Randbedingungen ist die Einteilung in *lineare Randbedingungen* und *komplexe Randbedingungen*. In der mathematischen Formulierung der Zuordnungsprobleme, in der es um Minimierung der Kostenfunktion geht (vergleiche Definition in Abbildung 2.3), ist ein Zuordnungsproblem genau dann komplex, wenn die Kostenfunktion nicht mehr linear ist, sie also nicht mehr in folgender Form notiert werden kann:

$$k = c_{11} \cdot x_{11} + c_{12} \cdot x_{12} + ... + c_{na} \cdot x_{na}$$

[2] an anderer Stelle werden einfache und mehrfache Randbedingungen auch *lokale* und *globale* oder *dynamische* und *statische* Randbedingungen genannt. Es wird dabei jeweils ein anderes Unterscheidungskriterium gewählt, im Prinzip jedoch entsprechen die Klassen einander.

Die Bewertungsfunktion ist die Summe der Verletzungsbewertungen der einzelnen Restriktionen. Eine Kostenfunktion ist komplex sobald eine ihrer Restriktionen komplex ist. Die mathematische Formalisierung der Zuordnungsprobleme ist sehr unhandlich und flexiblere Repräsentationen für Suchverfahren lassen im allgemeinen keine so einfache Klassifikation in komplexe und lineare Randbedingungen zu. Die meisten der praktisch auftretenden Zuordnungsprobleme und alle in dieser Arbeit als Beispiel aufgeführten Probleme besitzen jedoch komplexe Randbedingungen.

Neu in dieser Arbeit ist die Definition *monotoner Restriktionen*. Solche Restriktionen haben die Eigenschaft, daß eine Verletzung dieser Restriktionen nicht durch die Zuteilung eines weiteren Nachfragers beseitigt oder ihr Verletzungsgewicht reduziert werden kann, wenn die bisherige partielle Zuordnung nicht verändert wird. Bei einer schrittweisen Zuteilung ist der Wert der Bewertungsfunktion also monoton steigend. Restriktionen, die diese Eigenschaft nicht erfüllen, nenne ich *nichtmonoton*.

2.1.4 Kontinuierliche und diskrete Zuordnungsprobleme

In der Definition der Zuordnungsprobleme wurde bezüglich der Nachfrager- und Anbietermenge gefordert, daß die Mengen diskret sind. Ist die Anbietermenge außerdem endlich, spricht man von einem *diskreten Zuordnungsproblem*. Die Schulstundenplanung und die Personaleinsatzplanung sind diskrete Zuordnungsprobleme, da die Anbietermengen (Zeitfenster in einem Wochenplan) festgelegt und endlich sind. Handelt es sich bei der Anbietermenge um eine unendlich große oder nahezu unendlich große Menge, spricht man von einem *kontinuierlichen Zuordnungsproblem*. Die zuvor schon erwähnte Ressourcenbelegungsplanung ist ein solches kontinuierliches Problem, da die Menge der Anbieter (alle Zeitintervalle innerhalb eines Planungszeitraums) unendlich groß ist. Bei geschickter Implementierung kann man solche Zuordnungsprobleme jedoch mit denselben Problemlösungsmethoden lösen wie diskrete Zuordnungsprobleme. In dieser Arbeit wird, sofern es eine Rolle spielt, von diskreten Zuordnungsproblemen ausgegangen.

2.2 Lösungsverfahren für Zuordnungsprobleme

In diesem Abschnitt sollen zunächst die Anforderungen an eine Problemlösemethode definiert werden. Anschließend wird ein kurzer Überblick über bekannte Lösungsmethoden für Zuordnungsprobleme gegeben.

2.2.1 Anforderungen an die Problemlösungmethode

Zuordnungsprobleme haben sehr unterschiedliche Charakteristika und stellen deshalb auch unterschiedliche Anforderungen an die Problemlösemethode. Es gilt also zunächst mögliche Anforderungen zu klären, um die Entscheidung für eine geeignete Problemlösemethode fällen zu können.

Zusammengefaßt nach [Poeck 1995] und [Busch 1997] gibt es folgende Anforderungen an eine Problemlösungsmethode.

- Lösungsgüte
  Viele Zuordnungsprobleme sind aufgrund der Komplexität ihrer Randbedingungen NP-vollständig. Die *exakte* Lösung des Problemes (das bedeutet die Minimierung der Verletzungsbewertungsfunktion) erfordert dann exponentiell viel Zeit. Eine exakte Problemlösemethode kann deshalb aus Zeitgründen nicht praktikabel sein. Häufig ist dann eine Näherungslösung ausreichend. Einfachere Probleme können dagegen auch in kurzer Zeit exakt gelöst werden. Bezogen auf die Anforderungen kann man also zwischen Problemen unterscheiden, die eine exakte Lösung fordern, und Problemen, bei denen eine Approximationslösung ausreichend ist.

- Benötigte Laufzeit
  Eine weitere Anforderung ist das Einhalten einer bestimmten Laufzeit. Jede praktische Problemstellung hat einen zeitlichen Rahmen, in dem sie gelöst werden muß. In der Ressourcenbelegungsplanung geben z.B. die Termine der Aufträge den zeitlichen Rahmen vor.

- Variable Laufzeit abhängig von der gewünschten Exaktheit der Lösung.
  Diese Anforderungen stellt die beiden vorangegangen Anforderungen in Relation. Der Anwender der Problemlösungsmethode will in der Regel möglichst schnell eine akzeptable Lösung, statt einer optimalen Lösung, die nur in wesentlich längerer Zeit gefunden werden kann und möglicherweise nur geringfügig besser ist. Je nach Problemstellung kann es unterschiedlich wichtig sein, bereits in kurzer Zeit eine akzeptable Lösung zu finden.

- Eingriffsmöglichkeiten des Anwenders in den Lösungsprozeß.
  Um die Akzeptanz eines Systems beim Anwender zu fördern, ist Flexibilität nötig. Dazu gehören auch Eingriffsmöglichkeiten des Benutzers. Sie helfen bei der Lösungsfindung, indem der Benutzer als zusätzliche Wissensquelle genutzt wird und ermöglichen, unkompliziert vom Standard abweichende Problemstellungen zu meistern.

- Geeignete Problem- und Lösungsrepräsentation.
  Eine geeignete Repräsentation, die der realen Problembeschreibung möglichst nahe kommt, ist nicht nur für einen erleichterten Wissenserwerb nützlich, sondern auch Voraussetzung für den interaktiven Lösungsprozeß. Sie ermöglicht dem Benutzer ein Verständnis der Vorgänge und gibt damit die Grundlage zum Eingriff.

- Modifikationsmöglichkeiten für die Spezifikation während der Planung.
  In der Praxis tritt häufig auch das Problem der „Unvollständigkeit der Aufgabenstellung“ auf. Bei der Modellierung des Zuordnungsproblems für die Problemlösungsmethode kann nicht jeder eventuell auftretende Fall berücksichtigt werden. Bei der Personaleinsatzplanung kann der Planer einem Mitarbeiter vielleicht doch eine gesetzlich eigentlich nicht zugelassene Schicht

zuteilen, wenn er zuvor mit ihm Rücksprache gehalten hat. Dies erfordert ebenfalls Interaktionsmöglichkeiten.

Die in dieser Arbeit betrachteten Zuordnungsprobleme sind aus der Praxis gegriffen und nichttriviale Probleme mit komplexen Randbedingungen. Sie können typischerweise nicht in vertretbarer Zeit exakt gelöst werden. Die anderen Anforderungen sollen jedoch idealerweise erfüllt werden.

2.2.2 Überblick über die Problemlösungsverfahren für Zuordnungsprobleme

Es soll nun ein kurzer Überblick über die Problemlösemethoden für Zuordnungsprobleme gegeben werden.

Exakte mathematische Verfahren
Betrachtet man die Definition als Gleichungssystem, wie sie sie in 2.3 gegeben ist, so bieten sich mathematische Verfahren zur Lösung an. Dazu gehören Verfahren wie der Busacker-Goven-Algorithmus [Busacker & Goven 1961] und die Ungarische Methode [Burkard & Derigs 1980]. Sie finden die beste Lösung für lineare 1-1-Zuordnungsprobleme in ploynomieller Zeit. Mit Lösungsverfahren für die ganzzahlige lineare Programmierung, z.B. kann man beliebige lineare Zuordnungsprobleme exakt lösen. Hier wird die Berechnungszeit allerdings exponentiell.

Mathematische Näherungsverfahren
Gerade im OR-Bereich werden häufig mathematische oder graphentheoretische Näherungsalgorithmen für ganzzahlige lineare oder nichtlineare Programme verwendet [Grosche & Ziegler 1990]. Es wurden eine Reihe von heuristischen Verfahren entwickelt, die sich in Eröffnungs- und Iterationsverfahren gliedern. Eröffnungsverfahren generieren eine initiale potentiell suboptimale Lösung, während Iterationsverfahren eine gegebene Lösung schrittweise verbessern [Poeck 1994].

Genetische Algorithmen
Genetische Algorithmen [Mitchell 1997] sind ein typisches Paradigma zum Lösen von Optimierungsproblemen. Es soll evolutionäres Verhalten mit einer „Population“ von Lösungen nachgebildet werden. Dazu wird zunächst eine Menge von initialen suboptimalen Lösungen erzeugt, auf die dann im Wechsel die Funktionen Mutation, Crossover und Selektion angewendet werden, bis sich eine genügend gute Lösung entwickelt hat. In [Klügl 1994] wurde ihre Anwendung auf Zuordnungsprobleme untersucht.

Suchverfahren
Im Gegensatz zu den mathematischen Verfahren kann hier eine wesentlich intuitivere Problemrepräsentation verwendet werden. Sie muß nicht in ein lineares Programm oder ein kostenminimales Flußproblem (wie beim Busacker-Goven-Algorithmus) transformiert werden. Die vollständigen Suchverfahren benötigen wie auch die exakten mathematischen Verfahren exponentielle Laufzeit, wenn wir die Menge der betrachteten Zuordnungsprobleme nicht auf lineare Probleme einschränken wollen.

Es bieten sich *heuristische Suchverfahren* an. „Heuristisch“ bedeutet nach [Russell & Norvig 1995], daß Techniken verwendet werden, die die durchschnittliche Rechenzeit zur Problemlösung reduzieren, nicht jedoch notwendigerweise die

worst-case-Rechenzeit. Diese Techniken bestehen meist in der Einbringung und Ausnutzung von zusätzlichem Wissen über das Problem. Bestensuche ist damit im Gegensatz zur Tiefensuche ein „heuristisches“ Suchverfahren, da es bei Verwendung einer guten Bewertungsfunktion im average-case die Zeit zur Lösungsfindung reduziert, im worst-case zum vollständigen Durchlaufen des Suchraumes jedoch dieselbe Zeit benötigt.

2.3 „Vorschlagen und Vertauschen“

In der vorliegenden Arbeit wird die „Vorschlagen und Vertauschen“-Methode untersucht und soll hier beschrieben werden. V&V ist ein heuristisches, approximatives Suchverfahren. Es erzeugt schrittweise eine initiale Lösung und versucht in jedem Schritt, die bis dahin entstandene Teillösung zu optimieren. Nach vollendeter Generierung der initialen Lösung wird in einer weiteren Optimierungsphase versucht, die bis dahin gefundene Lösung zu verbessern. V&V ist in vielen Punkten der menschlichen Vorgehensweise beim Lösen von Zuordnungsproblemen nachempfunden.

2.3.1 Beschreibung des Algorithmus

Der V&V-Algorithmus ist nach [Poeck & Gappa 1993] eine spezielle Ausprägung des folgenden generellen Lösungsschemas für Zuordnungsprobleme:

1. Suche ein nicht zugeordnetes Nachfrager-Objekt.
2. Suche ein nicht zugeordnetes Anbieter-Objekt.
3. Bewerte die partielle Zuordnung.
4. Wenn Anforderungen (Restriktionen) nicht erfüllt sind, versuche eine neue partielle Zuordnung zu finden, so daß wiederum alle zuvor zugeteilten Nachfrager zugeteilt sind und soviele Anforderungen wie möglich erfüllt sind. (Anforderungen die nicht erfüllt werden können, werden möglicherweise in Schritt 6 korrigiert)
5. Wiederhole Schritt 1 bis 4 bis alle Nachfrager zugeteilt sind.
6. Wenn einige Anforderungen noch nicht erfüllt sind, versuche durch Vertauschen von Zuteilungen die Lösung zu optimieren.

Die „Vorschlagen und Vertauschen“-Methode im speziellen zeichnet sich durch die Einbringung von problemspezifischen Wissen in die Schritte 1, 2, und 3 aus. Im folgenden werden die Vorgehensweisen jedes Schrittes erläutert:

> „Die Problemlösung beginnt mit der Auswahl des nächsten zuzuordnenden Nachfrageobjektes aus der vom Benutzer vorgegebenen Startmenge durch den Algorithmus. Diese Auswahl kann zufällig geschehen, aufgrund einer vorbestimmten statischen Reihenfolge oder nach speziellen Auswahlstrategien, z.B. nach der Zahl der Freiheitsgrade.“

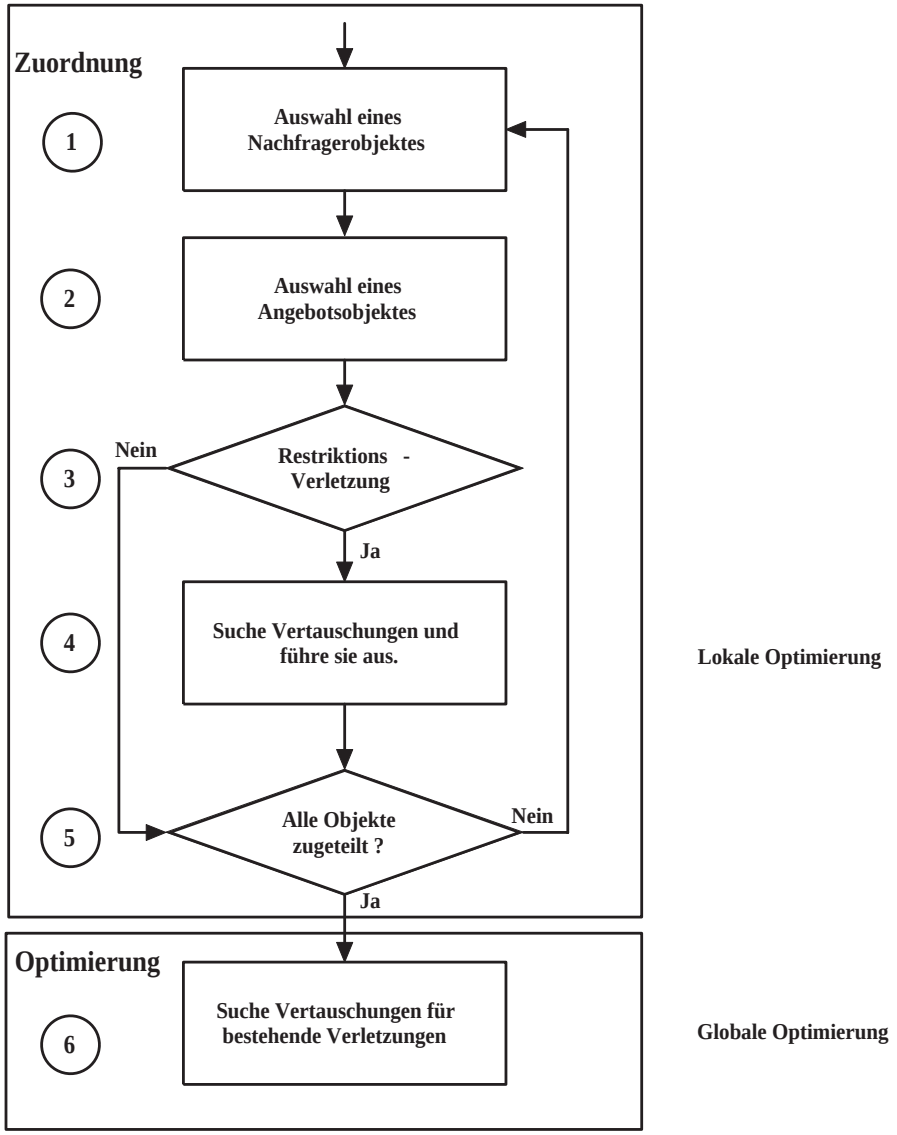

Abbildung 2.4: Illustration aus [Busch 1997] zum „Vorschlagen und Vertauschen"-Algorithmus.

„Im zweiten Schritt wird ein geeignetes, d. h. passendes und freies Angebotsobjekt ausgesucht und zugeordnet. Auch hier kann die Auswahl automatisch oder durch den Benutzer erfolgen. Im Spezialfall kann der Benutzer ein einzelnes Angebotsobjekt gezielt mit einem Element aus der Nachfragermenge belegen lassen.

Für die erfolgte Zuordnung[3] erfolgt in Schritt 3 eine Überprüfung aller zugehörigen Randbedingungen. Wurden Randbedingungen verletzt, so sucht der Algorithmus im vierten Schritt nach Tauschmöglichkeiten. Dazu werden zunächst alle Zuordnungen, bei denen Verletzungen aufgetreten sind, zurückgezogen und dann Alternativmöglichkeiten ausprobiert. Dieser Tauschvorgang wird durch eine zeitliche Schranke Über die Vorgabe der Zeitschranke kann der Benutzer

[3] Busch verwendet den Begriff Zuordnung hier synonym für Zuteilung.

in diesem Schritt die Laufzeit des Algorithmus und die Qualität der Lösung steuern. Falls keine Verbesserung erzielt wurde und wesentliche Randbedingungen nicht berücksichtigt werden konnten, kann der Benutzer später gezielt versuchen, diese aufzulösen."

„Eine gefundene vollständige Zuordnung kann schließlich durch weitere Vertauschungen noch optimiert werden, indem weitere Zeit in die Resolvierung verletzter Randbedingungen investiert wird" [Busch 1997]

Schritt 4 des Algorithmus (siehe Abbildung 2.4), die Suche nach Vertauschungen zur Korrektur der Verletzung, nennt man *lokale Optimierung*, da hier lokal beschränkt verbessert wird, d.h. nicht alle Verletzungen werden korrigiert, sondern die Verletzungen der aktuellen Zuteilung.
Schritt 6 des Algorithmus, die Optimierung nach vollständiger Zuordnung, nennt man im Gegensatz dazu *globale Optimierung*. Hier kann das Ergebnis weiter optimiert werden und es können gegebenenfalls andere Restriktionen berücksichtigt werden.

2.3.2 Eigenschaften des V&V-Verfahrens

Die Effizienz der Methode mißt sich unter anderem auch an der Erfüllung der Anforderungen. Dieser Abschnitt stellt unter Berücksichtigung der in Abschnitt 2.2 gestellten Anforderungen an eine Problemlösungsmethode einige Eigenschaften der V&V-Methode heraus.

- **Anytime-Eigenschaft**
  Ein großer Vorteil der V&V-Methode ist die *Anytime-Eigenschaft*[4]. Die Qualität des Ergebnisses ist abhängig von der dem Lösungsalgorithmus zur Verfügung gestellten Rechenzeit. Dabei müssen jedoch einige Einschränkungen gemacht werden. Ein Anytime-Algorithmus benötigt eine von der Problemstellung abhängige Mindestlösungszeit. Darüberhinaus kann Zeit zur Verbesserung des bis dahin gefundenen Ergebnisses investiert werden. In COKE ist die zur Verfügung gestellte Zeit nicht absolut gegeben (z.B. „Liefere eine Lösung für das Problem in maximal 10 Minuten") sondern in Größenordnungen („Pro Verletzung darf 30 Sekunden optimiert werden"). Das Zeitkontingent für den gesamten Lösungsprozeß ist also abhängig vom Problem.

- **Unterbrechungseigenschaften**
  Auf Grund der „Hillclimbing"-Verwandschaft des V&V's kommt der Algorithmus gut mit Unterbrechungen seitens des Benutzers oder Unterbrechungen nach einer Zeitüberschreitung zurecht. Da die Lösung inkrementell verbessert wird, ist die Qualität einer Zwischenlösung von der schon verwendeten Zuordnungs- und Optimierungszeit abhängig. Unterbricht man ihn während der globalen Optimierung, erhält man bereits ein zufriedenstellendes, eventuell schon gutes Ergebnis. Man kann den Zuordnungsvorgang jederzeit wieder starten, um die Lösung noch weiter zu optimieren. Die Punkte Anytime-Eigenschaft und Unterbrechungseigenschaften sind sich sehr ähnlich. Die Anytime-Eigenschaft macht Aussagen

[4] Eine ausführliche Behandlung des Themas Anytime-Eigenschaft findet sich in Kapitel 5.2.

bezüglich der vor dem Start der Optimierung festgesetzten Rechenzeit. Die Unterbrechungseigenschaft dagegen macht Aussagen über das Verhalten bei nachträglicher Veränderung (Verkürzung) der Rechenzeit.

- **Interaktionsmöglichkeiten**
  Aufgrund der Unterbrechungseigenschaften bieten sich zahlreiche Interaktivitätsmöglichkeiten. So kann der Benutzer die Zuordnung unterbrechen, dann den bestehenden Zustand verändern und den Lösungsvorgang erneut anstoßen. Während einer Unterbrechung findet der Benutzer den aus Sicht des Systems[5] besten Zwischenzustand, der jedoch mit „besserem Wissen" verändert werden kann. In [Busch 1997] werden ausführlich weitere Interaktionsmöglichkeiten dargestellt.

- **Effizient durch Expertenwissen**
  Der große Vorteil von V&V gegenüber reinen Suchverfahren ist, daß anwendungsspezifisches Wissen zur Lösungsfindung verwendet werden kann. Da die Problemlösungsmethode und die Wissensrepräsentation der menschlichen Vorgehensweise und Problembeschreibung nachempfunden sind, können erfolgreiche Strategien von Experten leicht umgesetzt werden. Diese Strategien sind vielfach bewährt und reduzieren die Zeit zum Finden einer Lösung.

- **Beliebige Randbedingungen**
  Die zu beachtenden Randbedingungen können bei V&V beliebig komplex sein. Die Problemlösungsmethode fordert keine Einschränkungen wie z.B. die ausschließliche Verwendung von linearen Randbedingungen. Viele in der Praxis auftretenden Probleme, insbesondere die in dieser Arbeit als Beispiele beschriebenen, enthalten komplexe Randbedingungen. (ausführlich in [Busch 1997]; S. 30)

2.4 Algorithmen und Problemlösungsmethoden

„Vorschlagen und Vertauschen" ist eine *Problemlösungsmethode* für Zuordnungsprobleme [Puppe 1990]. Intuitiv verwendet man für V&V auch die Begriffe Algorithmus oder Strategie. In diesem Abschnitt wird der Begriff der Problemlösungsmethode von semantisch ähnlichen oder verwandten Begriffen abgegrenzt und deren Verwendung in dieser Arbeit erläutert.

Der Duden definiert den Begriff *Algorithmus* folgenderweise:

> „Unter einem *Algorithmus* versteht man eine Verarbeitungsvorschrift, die so präzise formuliert ist, daß sie von einem mechanisch oder elektronisch arbeitenden Gerät durchgeführt werden kann. Aus der Präzision der sprachlichen Darstellung eines Algorithmus muß die Abfolge der einzelnen Verarbeitungsschritte eindeutig hervorgehen. Hierbei sind Wahlmöglichkeiten zugelassen. Nur muß dann genau festliegen, wie die Auswahl einer Möglichkeit erfolgen soll. [...] Kochrezepte, Bastelanleitungen, Partituren, Spielregeln und alltägliche Vorschriften erinnern an Algorithmen; sie sind aber nur selten

[5] Das „System" ist hier die Kombination des Algorithmus und des programmierten Expertenwissens.

exakt ausformuliert und enthalten oft Teile, die vom Ausführenden beliebig interpretiert werden können."[Duden 1988]

Diese Definition ist sehr streng. Im allgemeinen Sprachgebrauch wird Algorithmus oft auch zur Beschreibung von Vorgehensweisen verwendet, die nicht dieser Definition genügen. Häufig ist es wünschenswert, einen Algorithmus möglichst allgemein zu formulieren und keine spezielle Datenstruktur vorauszusetzen. Auch in dieser Arbeit wird der Begriff etwas aufgeweicht verwendet. Ein Algorithmus soll so nahe wie möglich an einer Implementation sein, jedoch nicht zu sehr eine bestimmte Datenstruktur (oder Wissensrepräsentation) vorschreiben.

Im Zusammenhang mit V&V und dem *Problemtyp* Zuordnung werden häufig auch die Begriffe *Problemlösungsmethode* und *Problemlösungsstrategie* verwendet. Diese sind als Bezeichnung für eine abstrakte Beschreibung der Vorgehensweise geeigneter. Dennoch wird eine Problemlösungsmethode auch Algorithmus genannt:

„Als *Problemlösungsmethode* bezeichnen wir einen Algorithmus, der angibt, wie bereichsspezifisches Wissen zur Problemlösung verwendet wird. Im Vergleich dazu ist ein Sortieralgorithmus nur auf einer Ebene parametrisiert, da er nur auf Eingabedaten, aber nicht auf bereichsspezifisches Wissen referiert."[Puppe 1990]

Eine Trennung der Begriffe Algorithmus und Problemlösungsmethode ist schwer durchzuführen. Eine Problemlösungsmethode ist ein Algorithmus, und auch eine spezielle Implementation einer solchen Methode beschreibt einen Algorithmus bzw. geht nach einer Problemlösungsmethode vor. Die Implementation ist viel spezieller und in mehr Details beschrieben als es eine Problemlösungsmethode sein muß. Sowohl die abstrakte als auch die spezielle Vorgangsbeschreibung hat ihre Vorteile. Eine abstrakte Beschreibung ist übersichtlicher und außerdem flexibler gegenüber Veränderungen der Problemstellung. Sie umfaßt in vielen Fällen eine größere Klasse speziellerer Ausformulierungen. Eine speziellere Beschreibung hat dagegen den Vorteil, daß sich die Eigenschaften der mit ihr beschriebenen Methode besser analysieren lassen und der Arbeitsaufwand zur Erstellung eines Programmes geringer ist.

Jetzt kann man wieder einen Bogen zur Themenstellung der Arbeit schließen. Diese Arbeit versucht die Lücke zwischen der abstrakt beschriebenen Problemlösungsmethode und den sehr speziellen Implementationen schließen. Für einige Teilbereiche der Problemlösungsmethode werden verschiedene Implementierungsalternativen genauer spezifiziert werden.

2.5 Effizienz

Der Begriff der *Effizienz* verdient in dieser Arbeit auch eine genauere Betrachtung, da Effizienzuntersuchungen und Effizienzsteigerungen das zentrale Thema sind. Der Duden definiert Effizienz folgendermaßen:

Effizienz[lat] Wirksamkeit, Leistungsfähigkeit (im Verhältnis zu den aufgewandten Mitteln); [bes.] Wirtschaftlichkeit.

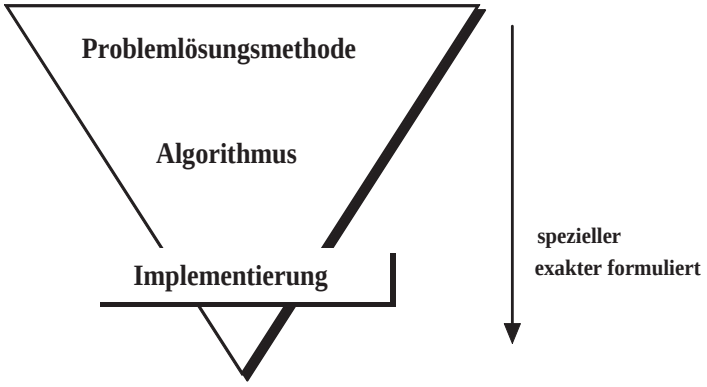

Abbildung 2.5: Der Weg von einer Problemlösungsmethode zu einer Implementation ist fließend. Während Problemlösungsmethoden oft noch sehr abstrakt die Vorgehensweise beschreiben und vom Ausführenden interpretiert werden müssen, sind Algorithmen schon möglichst exakt und bieten die Grundlage für eine Implementation.

Wirksamkeit und Leistungsfähigkeit bedeutet auf Zuordnungsprobleme gesehen eine möglichst hohe Qualität der Lösungen. Die Anforderungen an Lösungen sind in den verschiedenen Anwendungsbereichen sehr unterschiedlich, gemeinsam ist ihnen jedoch die Forderung nach Einhalten der harten Randbedingungen und die Optimierung der weichen Randbedingungen. Eine *Effizienzsteigerung* bedeutet eine bessere Erfüllung dieser Forderungen.

Leistungsfähigkeit muß in Relation zu den aufgewandten Mitteln gesehen werden. Ein Algorithmus, der hinsichtlich der Lösung sehr leistungsfähig ist, d.h. die beste Lösung findet, kann bei realen Problemen unpraktikabel sein, da die nötigen Mittel nicht aufgebracht werden können. Bei der Zuordnung spielen die Mittel

- Speicher,
- Zeit und
- Wissen

eine wesentliche Rolle. Speicher ist der begrenzende Faktor vieler Lösungsverfahren. Zudem ist er in realen Umgebungen immer endlich und kann nicht beliebig ausgebaut werden. Zeit kann dagegen zwar theoretisch beliebig hinzugenommen werden; Wirksamkeit bedeutet jedoch auch, daß man das Ergebnis abwarten kann, bzw. daß das Ergebnis nicht schon veraltet ist. So ist die Zeit zwischen Bekanntwerden der Anforderungen und dem Termin, an dem eine Lösung benötigt wird, üblicherweise auch beschränkt. Ein gutes Beispiel ist die Ressourcenbelegungsplanung, wo sich häufig die Anforderungen an einen Plan ändern und schnell eine neue Lösung benötigt wird [Busch 1997]. Eine weitere Ressource ist das Wissen. Zusätzliches Wissen über das Problem kann die Leistungsfähigkeit eines Systems zur Lösung verbessern. Der Aufwand, der in Wissenserwerb und Evaluation steckt, muß mit der gewonnenen Leistung im Verhältnis gesehen werden.

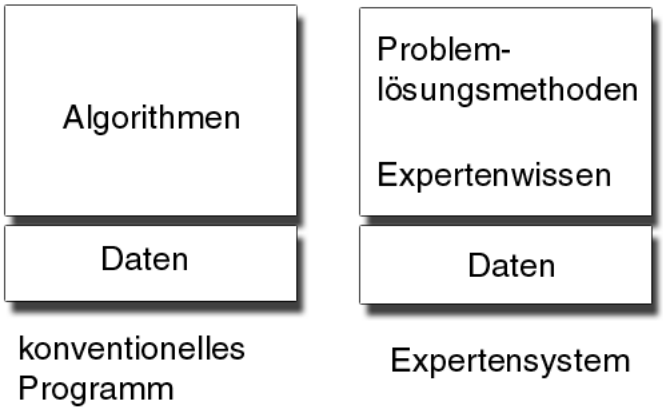

Abbildung 2.6: Der Aufbau von konventionellen Programmen und Expertensystemen im Vergleich. Bild aus [Puppe 1990].

Effizienz bedeutet auch Leistungsfähigkeit in Bezug auf die Mensch-Maschine-Kommunikation der Anwendung. In den Arbeiten von [Busch 1997] und [Forster 1999] wird dieser Aspekt bereits ausführlich behandelt und soll hier nur am Rande erwähnt werden. Die aufgewandten Mittel sind bei diesem Aspekt die Zeit der Arbeitskräfte, Experte und Anwender. Vielfach betreffen Effizienzkriterien bei der Mensch-Maschine-Kommunikation die graphische Benutzeroberfläche, aber es gibt auch algorithmisch relevante Bereiche wie die Möglichkeiten zur Interaktivität oder die Beschaffenheit einer Schnittstelle, die dem Experten die Wissenseingabe ermöglicht.

2.6 Expertensysteme und Shell-Baukästen

Der Programmierteil dieser Arbeit wird als Bestandteil der Expertensystem-Shell COKE implementiert. [Puppe 1988] definiert den Begriff Expertensystem folgenderweise:

> „Expertensysteme sind Programme, mit denen das Spezialwissen und die Schlußfolgerungsfähigkeit von Experten auf eng begrenzten Aufgabengebieten rekonstruiert werden soll. Ihr grundlegendes Organisationsprinzip ist die Trennung von Problemlösungsmethoden und Wissen.“ [Puppe 1988]

„Vorschlagen und Vertauschen“ benutzt zur Lösung konkreter Probleme anwendungsspezifisches Wissen wie die Kenntnis von Restriktionen und Auswahlkriterien für passende Anbieter. Eine Anwendung, die auf V&V basiert, ist deshalb nach obiger Definition ein *wissensbasiertes System* oder *Expertensystem*, da sie sich aus Problemlösungsmethode und Expertenwissen zusammensetzt.

Die Trennung von Methode und Wissen ist gewährleistet, da das Wissen weitgehend in Restriktionen, Vorschlägen und Korrekturen gekapselt ist. Aufgrund dieser Trennung ist „Vorschlagen und Vertauschen“ nicht nur sehr flexibel und

kann an viele Problemstellungen angepaßt werden, sondern sie eignet sich auch sehr gut zur Verwendung in einer *Expertensystem-Shell*, einem Werkzeug zur Erstellung von Expertensystemen.

> Solche Shells sind Programme, „die auf bestimmte Problemlösungstypen spezialisiert sind und geeignete Problemlösungsmethoden, Wissensrepräsentationen und Wissenseingabeformen anbieten.“ [Puppe 1990]

COKE ist ein Beispiel einer Zuordnungsproblem-Shell. Sie ist spezialisiert auf den Problemtyp Zuordnung und verwendet die Problemlösungsmethode „Vorschlagen und Vertauschen“. Die Ergebnisse dieser Arbeit sollen zu einer Verbesserung oder Erweiterung dieser Zuordnungs-Shell beitragen. Genauso sollen die Ergebnisse auf andere Expertensysteme oder Expertensystem-Shells anwendbar sein, die auf „Vorschlagen und Vertauschen“ basieren.

[Puppe 1990] beschreibt die in Expertensystemen verwendeten Problemlösungsmethoden auch als „hochgradig parametrisierte Algorithmen“. Die Auswahl verschiedener Detailvarianten des Algorithmus wie die, die in den folgenden Kapiteln beschrieben werden, kann auch als Parametrisierung des Algorithmus gesehen werden. Desweiteren bedeutet Parametrisierung, daß die Problemlösungsmethode durch die Eingabe des anwendungsspezifischen Wissens an verschiedene Probleme angepaßt werden kann. Das Wissen des Experten über eine Instanz eines Problemtyps kann in einer Shell auf folgende Arten angegeben werden.

- Angabe von Funktionen.
  Für die Eingabe von komplexem heuristischen Wissen ist es notwendig, eine Schnittstelle zum Problemlöser zu schaffen. Solches Wissen sind z.B. - Restriktionen oder Korrekturvorschläge. Aufgrund der Verschiedenheit der Anforderungen ist dazu eine Sprache nötig. COKE ist in LISP [Guy & Steele 1990] programmiert und benutzt hier die Möglichkeiten zur Angabe von LISP-Funktionen während der Laufzeit. Andere Systeme können eine Sprache zum Zugriff auf die Wissensbasisobjekte definieren. Vorstellbar wäre auch eine graphische Sprache [Schiffer 1998]. Programme wie ThingLab zeigen, daß sich visuelle Programmierung bei Einschränkung auf bestimmte Domänen[6] auch für die Eingabe von Constraints (hier: Restriktionen) eignet.

- Schalter
  Viel einfacher als eine Konfigurierung durch die Angabe von Funktionen, allerdings auch weniger flexibel, ist das Auswählen verschiedener Alternativen durch Schalter. Beispiel ist eine Auswahl zwischen verschiedenen implementierten Problemlösern.

- Angabe von numerischen Parametern
  Eine Konfigurierung der Problemlösungsmethode kann auch anhand numerischer Parameter geschehen. Die Angabe von Zeitschranken zur Berechnung einer Lösung oder die Angabe einer Rekursionstiefe für manche Suchalgorithmen sind Beispiele dafür.

[6] Eine solche Problemdomäne kann z.B. die Domäne Produktionsplanung sein.

Kapitel 3

Vergleich von Implementierungsalternativen

Die Suche nach Implementierungsalternativen des V&V-Algorithmus erstreckt sich über die nächsten drei Kapitel; dabei wählt jedes Kapitel einen anderen Ansatzpunkt. In diesem Kapitel werden die einzelnen Schritte des "Vorschlagen und Vertauschen"-Algorithmus aufgeschlüsselt und für jeden Schritt Variationsmöglichkeiten überlegt. Manche dieser Schritte sind sehr eindeutig. In diesem Falle wird anstatt einer Darstellung der Varianten eine Spezifikation des Schrittes angegeben.

Die Abbildung 2.4 auf Seite 15 zeigt den Algorithmus. Die Gliederung dieses Kapitels richtet sich streng nach dem Aufbau des Schaubildes.

3.1 Auswahl eines Nachfrageobjektes

Die Auswahl der Nachfrageobjekte bestimmt die Reihenfolge der Zuteilungen. In einigen Fällen kann es sinnvoll sein, auf diese Reihenfolge Einfluß zu nehmen. So wählen auch Disponenten[1] bei der manuellen Planung „Problemschichten" vorrangig zur Zuteilung aus. Bei der Zugbegleiterplanung waren solche Problemschichten beispielsweise Wochenendschichten und Nachtschichten. Es zeigte sich, daß es eine gute Strategie ist, solche Schichten, für die besondere Regeln existieren, frühzeitig zuzuteilen.

Ganz allgemein kann man feststellen, daß es vorteilhaft ist, den Nachfrager mit den meisten Einschränkungen zuerst zu behandeln. Auf diese Weise können Sackgassen im Suchpfad vermieden werden - oder intuitiv: Man versucht die Nachfrager, für die es aufgrund der vielen Restriktionen nur wenig Anbieter gibt, zuzuteilen, solange es noch freie Anbieter gibt. [Russell & Norvig 1995] bezeichnen dieses Objekt als die „most constrained variable".

Miteinander verwandt, aber nicht notwendigerweise identisch sind die Eigenschaften *wenig Freiheitsgrade* und *stark eingeschränkt*. Wenig Freiheitsgrade für einen Nachfrager bedeutet, daß er nur sehr wenigen Anbietern zugeteilt werden kann, ohne daß Verletzungen entstehen. *Sehr eingeschränkt* bedeutet dagegen, daß für viele der Anbieter Restriktionen bestehen, die eine Zuteilung

[1] Disponent bezeichnet einen Experten, der das Zuordnungsproblem löst.

verhindern. Bei konstant angenommener Anzahl der Anbieter sind die beiden Aussagen gleichbedeutend.

3.1.1 Grundsätzliche Möglichkeiten

Um die Vorgehensweise des menschlichen Disponenten bei der Zuordnung nachzubilden, soll es also möglich sein, mit anwendungsspezifischem Wissen die Reihenfolge der Nachfrager bei der Zuteilung zu bestimmen. Folgende Alternativen sind denkbar und bereits in [Poeck & Puppe 1992] beschrieben:

- **Unsortierte Auswahl der Nachfrager**
  Die Reihenfolge bleibt unbeeinflußt und richtet sich zufällig nach der vorgefunden Datenstruktur.
- **Statisch sortierte Nachfragerliste**
  Direkt beim Start des "Vorschlagen und Vertauschen"-Algorithmus werden die Nachfrager bewertet und daraus eine Reihenfolge generiert, die im gesamten folgenden Zuordnungsprozeß eingehalten wird.
- **Statisch sortierte Nachfragerliste mit dynamischen Modifikationen**
  Die Nachfragerliste wird beim Start statisch sortiert und im späteren Verlauf bei bestimmten Zuteilungen modifiziert. So können bei Knappwerden einer Ressource Nachfrager, die diese Ressource benötigen, bevorzugt zugeteilt werden.
- **Dynamisch sortierte Nachfragerliste**
  Im Gegensatz zur statischen Variante wird davon ausgegangen, daß sich durch eine erfolgte Zuteilung die Voraussetzungen so verändern können, daß die Auswahl des nächstbesten Nachfragers zuvor noch nicht absehbar war. Eine statische Vorsortierung reicht in diesem Fall nicht aus. Benötigt wird eine Bewertungsfunktion, die geeignet ist, nach jeder Zuteilung den besten nachfolgenden Nachfrager zu ermitteln.

In einem Shellbaukasten bedeuten die letzten beiden Varianten, daß der Wissensingenieur Sortier- bzw. Bewertungsfunktionen angeben muß. Mit diesen kann er Heuristiken berücksichtigen, wie z.B., sich Nachfrager mit wenig Freiheitsgraden zuerst zur Zuteilung vorzunehmen. COKE ermöglicht bereits diese vier Varianten bei der Auswahl der Nachfragerobjekte. Soll eine Sortierung vorgenommen werden, ist die Angabe einer weiteren Wissenskomponente, einer statischen oder dynamischen Sortierfunktion nötig.

3.1.2 Generische Ermittlung der Sortierung

Eine Sortierung der Nachfragerliste anwendungsunabhängig möglichst generisch oder mit wenig Aufwand zu erstellen, ist eine nützliche Erweiterung, die dem Experten die explizite Angabe einer Sortierfunktion erspart. Dies ist ein Spezialfall der dynamischen Sortierung.

Eine Möglichkeit dazu zeigt Algorithmus 1. Durch systematisches Vorausberechnen wird der Nachfrager mit den wenigsten Freiheitsgraden ermittelt. In der

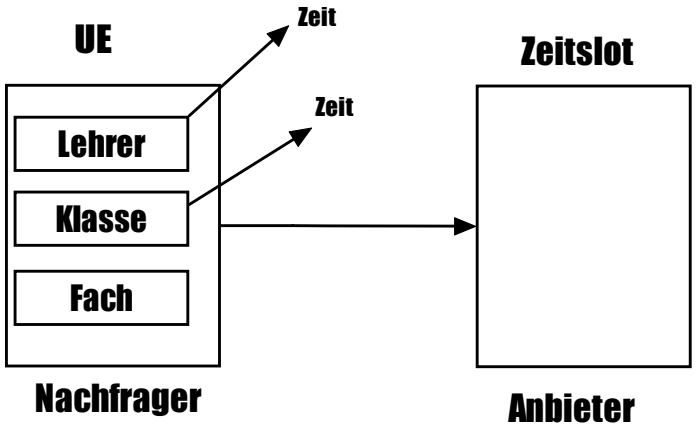

Abbildung 3.1: Darstellung der Objektstruktur bei der Schulstundenplanung. Zugeordnet werden Unterrichtseinheiten (UE) zu Zeitslots im Stundenplan. Eine UE ist ein Kompositum aus den Objekten Lehrer, Klasse und Fach. Die Einzelelemente haben jeweils eigene begrenzte Ressourcen. In diesem Falle ist es die Ressource Zeit, die die Möglichkeiten zur Zuteilung einschränkt.

Interpretation von Freiheitsgrad kann man hier Unterschiede machen: So kann man die Zuteilung eines Nachfragers zu einem Anbieter auch bei einer leichten Verletzung als Freiheitsgrad zulassen, statt wie im vorher beschriebenen Fall nur bei Verletzungsfreiheit. In Schritt 6 des Algorithmus wird diese Unterscheidung getroffen. Die Berechnungskomplexität[2] des Algorithmus 1 liegt in $\mathcal{O}(n \cdot a)$. Da n Nachfrageobjekte ausgewählt werden müssen, liegt der Beitrag der Nachfragerbestimmung zum ganzen V&V-Prozeß in der Komplexität von $\mathcal{O}(n^2 \cdot a)$. Dieser Aufwand ist nicht unerheblich und veranlaßt dazu, mehr wissensbasierte Ansätze vorzuziehen.

Algorithmus 1 Am meisten eingeschränktes Nachfrageobjekt

Input: Nachfragermenge N , Anbietermenge A

1 $minanzahlanbieter = |A| + 1$
2 $besternachfrager = undef$
3 Für alle $n \in N$
4 $\quad anzahlanbieter = 0$
5 $\quad$ Für alle $a \in A$
6 $\quad\quad$ Wenn Zuteilung (n, a) keine [schwere] Verletzung
7 $\quad\quad\quad anzahlanbieter = anzahlanbieter + 1$
8 $\quad$ Wenn $minanzahlanbieter > anzahlanbieter$
9 $\quad\quad minanzahlanbieter = anzahlanbieter$
10 $\quad\quad besternachfrager = n$

Output: Nachfrager mit minimalen Freiheitsgraden: $besternachfrager$

[2]Die Angaben in der $\mathcal{O}$-Notation werden hier unter der Voraussetzung gemacht, daß ein Verletzungstest in $\mathcal{O}(1)$ gemacht werden kann. Diese Voraussetzung ist faktisch nicht zu garantieren, wird hier jedoch vereinfachend gemacht, um Vergleichswerte zu bestimmen.

Bei manchen Problemen kann man weitgehend generisch aus dem Aufbau der Wissensrepräsentation eine Berechnungsheuristik für die Freiheitsgrade von Nachfragern ableiten. Betrachten wir das Beispiel der Schulstundenplanung. Die Nachfrager der Schulstundenplanung, die Unterrichtseinheiten (kurz UE), sind ein Kompositum aus mehreren Objekten (vergleiche Abbildung 3.1). Eine UE wird bestimmt durch einen sie haltenden Lehrer, eine Schulklasse und ein Unterrichtsfach. Die Anbieter sind Zeitslots[3] in einem Stundenplan. Sowohl Lehrer als auch Klassen haben eine Ressource (ihre freien Zeitslots), die durch eine Zuteilung knapp werden kann. Daraus resultieren Einschränkungen, die ohne Vorausberechnung von Zuordnungen nur durch Ermittlung der freien Zeitslots von Lehrer und Klasse bestimmt werden können. Noch genauer wird diese Heuristik, wenn man die Schnittmenge der freien Zeitslots von Lehrern und Klassen betrachtet. Völlig generisch kann eine Sortierung der Nachfrager nach den Freiheitsgraden auf diesem Weg nicht erreicht werden. Gibt man zu jeder Objektklasse eine Funktion an, die errechnet, wie eingeschränkt sich eine Instanz dieser Klasse gerade fühlt, kann man daraus generisch die Sortierung ermitteln. In unserem Beispiel bedeutete dies:

- Das Objekt **Lehrer** fühlt sich eingeschränkt, wenn es wenige freie Zeitslots hat.
- Das Objekt **Klasse** fühlt sich ebenfalls eingeschränkt, wenn es wenige freie Zeitslots hat.
- Das Objekt **Fach** ist nie eingeschränkt.
- Das Objekt **Unterrichtseinheit** ist eingeschränkt, wenn eines der Objekte, aus denen es zusammengesetzt ist, eingeschränkt ist.

Diese heuristische Vorgehensweise ist weniger rechenintenstiv als die völlig generische. Im beschriebenen Beispiel liegt der Aufwand zur Ermittlung des am meisten eingeschränkten Nachfragers nur noch in $\mathcal{O}(n)$. Außerdem muß kein Verletzungstest durchgeführt werden, dessen Komplexität beliebig groß sein kann. Diese Vorgehensweise ist jedoch domänenabhängig, läßt sich außerdem nicht in jeder Problemdomäne anwenden und erfordert die Eingabe eines zusätzlichen Wissenselementes. Der Vorteil im Gegensatz zur direkten Angabe einer Sortierfunktion liegt darin, daß die Formulierung der Heuristik für den Experten einfacher wird. Generell kann die generisch erzeugte Sortierung nicht mehr leisten als die in Abschnitt 3.1.1 beschriebenen Varianten.

3.1.3 Fazit

Welche der oben vorgestellten Varianten der Wissensingenieur wählt, hängt primär davon ab, welche Art von Strategie er formulieren will. Eine dynamische Sortierung der Nachfragerliste erfordert mehr Rechenzeit als eine statische. [Poeck & Puppe 1992] untersuchten die Varianten an verschiedenen Problemen und stellten fest:

[3] Zeitslots sind festgelegte Zeitintervalle. Sie entsprechen einer festgelegten Stundeneinteilung, wie man sie aus der Stundenplanung kennt

„However, for complex assignment problems like COMPLEX TIME TABLE it is much better to invest the computing time in the early steps, while it is very difficult or even impossible to repair a bad solution afterwards.“

Eine generische Ermittlung einer sinnvollen Sortierung orientiert an der Heuristik der *most constrained variable* ist sehr rechenintensiv, man kann jedoch auch hier einen mehr wissensbasierten Ansatz wählen. Dieser kann jedoch auch nicht mehr leisten, als durch eine dynamische Sortierfunktion möglich wäre.

3.2 Auswahl des Anbieterobjektes

Nachdem ein Nachfrageobjekt ausgesucht wurde, muß ein für die Zuteilung möglichst geeignetes Angebotsobjekt gefunden werden, d.h. z.B. ein Tag in einem Dienstplan für eine Schicht oder eine Maschine für einen Arbeitsauftrag. Da es für gewöhnlich zu aufwendig wäre, alle Anbieter zu testen, fließt hier problemspezifisches Wissen in Form von *Vorschlägen* ein, das der Wissensingenieur mit Hilfe von Vorschlagsfunktionen formulieren kann. Die Möglichkeit zur Eingabe mehrerer Vorschlagsfunktionen, die einzeln aktiviert und deaktiviert werden können, erleichtert es, wahlweise verschiedene Lösungsstrategien zu verwenden. Im Falle mehrerer gleichzeitig aktiver Vorschläge beinhaltet der Gesamtvorschlag die Vereinigung der Rückgabewerte.

3.2.1 Interpretation der Vorschläge

Ob eine Lösung des betrachteten Zuordnungsproblems gefunden wird, hängt wesentlich von der Qualität der Vorschläge ab. Wenn eine zur Lösung notwendige Zuteilung nicht durch passende Vorschläge ermöglicht wird, wird sie durch V&V auch nicht gefunden. Im Idealfall enthalten die durch die Vorschlagsfunktionen gegebenen Anbieter auch den für eine Zuteilung optimalen Anbieter. Das Expertenwissen kann jedoch auch als Vorschlag in dem Sinne verstanden werden, daß es durchaus auch andere Anbieter gibt, die eine bessere Zuteilung als die vorgeschlagenen Anbieter ergeben. Man kann also folgende beiden Fälle unterscheiden:

- Vorschläge enthalten alle sinnvollen Anbieter
  In diesem Fall enthalten die Vorschläge alle Anbieter, außer denen, von den zweifelsfrei bekannt ist, daß sie keine gültige Lösung ergeben können. Im Beispiel von PEPSI, der Anwendung zur Dienstplanerstellung für Zugbegleiter, werden zu einer Montagsschicht alle im Planungszeitraum vorkommenden Montage als Vorschlag gegeben. Weitere Anbieter müssen nicht betrachtet werden, da sie zu keiner gültigen Lösung führen können bzw. eine Konsistenzverletzung erzeugen. Eine Montagsschicht kann nicht an einem Dienstag eingeplant werden. Diese Verletzung kann auch nicht durch Korrekturen an anderen Schichten beseitigt werden.

- Vorschläge enthalten nur einen Teil der sinnvollen Anbieter.
  Liefert die Vorschlagsfunktion „nur Vorschläge“, aber es kann darüberhinaus noch andere sinnvolle Anbieter geben, empfiehlt sich eine andere Vorgehensweise. Dann sollte der Algorithmus weitere Anbieter testen, wenn

sich keine zufriedenstellende Zuteilung ergibt. In dem Beispiel des letzten Punktes hieße das, daß nicht alle Montage sondern einige aus bestimmten Gründen vorgezogene Montage vorgeschlagen werden. Wenn sich aus diesen keine Zuteilung ohne Konsistenzverletzung ergibt, müssen dennoch andere Anbieter in Betracht gezogen werden.

3.2.2 Auswahlkriterium für den Besten unter den Vorschlägen

Das Ziel ist, eine bezüglich der Bewertungsfunktion möglichst gute Lösung zu finden, d.h. möglichst wenige Verletzungen zu erzeugen. Das Hauptkriterium für die Auswahl eines Vorschlages bleibt also auch eine möglichst niedrige Verletzungsbewertung. Sind unter den vorgeschlagenen Anbietern mehrere mit gleicher Verletzungsbewertung, kann es nützlich sein, vorausschauend die Einschränkung des Suchraumes durch die Zuteilung dieses Anbieters abzuschätzen. Unter den bezüglich der Bewertungsfunktion gleichwertigen Anbietern ist der zu wählen, der die Setzmöglichkeiten (bzw. Freiheitsgrade) für die nachfolgenden Zuordnungen möglichst wenig einschränkt. Zusammenfassend kann man also folgende zwei Auswahlkriterien festhalten:

- Primäres Auswahlkriterium: Restriktionsbewertung
- Sekundäres Auswahlkriterium: Erhaltung möglichst vieler Freiheitsgrade

Das primäre Auswahlkriterium hat unmittelbare Auswirkungen auf die Bewertungsfunktion. Die Wirkung des sekundären Kriteriums ist erst in indirekter Folge über die Anzahl der Freiheitsgrade für spätere Zuteilungen gegeben. Mehr Freiheitsgrade lassen mehr Raum (im Suchraum) für die Optimierung und haben damit Einfluß auf die Bewertungsfunktion.

Das sekundäre Auswahlkriterium über das primäre zu stellen, scheint im allgemeinen wenig sinnvoll, da das Ziel die Optimierung bezüglich des Verletzungsgrades ist. In der Anfangsphase der Zuordnungen (also während der ersten Zuteilungen) mag es jedoch auch sinnvoll sein, den Erhalt vieler Freiheitsgrade einer geringfügig schwereren Verletzung vorzuziehen. In diesem Fall muß man jedoch auch darauf achten, daß dieser Vorzug nicht durch die lokale Optimierung wieder revidiert wird. Das Ziel dieser ist die Reduzierung der Restriktionsbewertung ohne Berücksichtigung der Freiheitsgrade.

3.2.3 Ermittlung des am wenigsten einschränkenden Anbieters

Die Berechnung eines Vorschlages, der die nachfolgenden Zuteilungen möglichst wenig einschränkt, ist sehr aufwendig. Hier kann man wieder problemspezifisches Wissen einfließen lassen, um die Freiheitsgrade abzuschätzen. In einem Shell-Baukasten-System kann man dem Experten folgende Möglichkeiten zur Ermittlung des „am wenigsten einschränkenden“ Anbieters geben.

- Wissensbasiert
  Der Experte kann selbst eine Funktion, die das Ausmaß der Freiheitsgrade bewertet, angeben. Dies ist zweifelsohne die flexibelste Methode, stellt den

Personal \ Arbeitstage											
				F		F	F				
			F	F	N						
	F		F	F	F		F	F	F		
		F		F	F	F	F	F	F	F	
		F		F	F			F	F		
		N	N	N	N						

Abbildung 3.2: Ausschnitt aus einem Dienstplan für Pflegepersonal. Viele Zuordnungsprobleme lassen sich in Tabellenform darstellen. Ein eingetragenes `N` bedeutet eine Nachtschicht für einen Mittarbeiter an einem Arbeitstag. Ein `F` bedeutet eine Frühschicht. Häufig ist es eine gute Heuristik anfangs so *kompakt* wie möglich zu planen, um die zukünftigen Auswahlmöglichkeiten nicht einzuschränken. "Kompakt" bezieht sich hier auf die Nachbarschaften in der Tabellendarstellung.

Wissensingenieur allerdings vor das Problem, eine geeignete Schätzfunktion zu finden. Hier ist wieder zusätzliches Wissen über die Problemdomäne nötig. Abbildung 3.2 zeigt ein Beispiel. Hier werden für die weitere Planung Freiheitsgrade erhalten, indem versucht wird, möglichst *kompakt* zu planen, solange es keine Verletzungen gibt. Kompakt zu planen bedeutet möglichst keine einzelnen leeren Tabellenfelder zwischen eingetragenen Schichten zu lassen. Diese Technik ist der menschlichen Vorgehensweise nachempfunden. Sie wirkt wenig einschränkend, da mehr freier Raum im Dienstplan erhalten bleibt. Da Verletzungen meist zwischen benachbarten Zuteilungen auftreten, ist viel Freiraum positiv für spätere Zuteilungen. Natürlich soll aufgrund der Kompaktheit keine Verletzung in Kauf genommen werden. Die Verletzungsfreiheit bleibt also das primäre Auswahlkriterium für einen Anbieter. Elegant kann man solche Heuristiken in Form einer sehr leichten Verletzung angeben. Auf diese Weise wird auch bei der lokalen Optimierung auf die Erhaltung der Freiheitsgrade geachtet. Da das Gewicht der „echten" Verletzungen größer ist, bleibt das sekundäre Auswahlkriterium auch wirklich sekundär.

- Generisch
  Ohne zusätzliches Domänenwissen berechnet der Algorithmus 2 den Anbieter, der den Freiheitsgrad für spätere Zuteilungen am wenigsten einschränkt. Systematisch werden die Konsequenzen nach dem Setzen eines jeden Anbieters vorausberechnet. Der Algorithmus besitzt die Laufzeitkomplexität $\mathcal{O}(a^2 \cdot n)$ und wird im Laufe der kompletten Zuordnung n mal aufgerufen, sein Gesamtbeitrag zur Laufzeit liegt also in $\mathcal{O}((a \cdot n)^2)$. Insgesamt ist dieser Aufwand als nicht unerheblich zu bewerten und man wird besser auf den wissensbasierten Ansatz zurückgreifen.

Algorithmus 2 Bestimmung des am wenigsten einschränkenden Anbieters

Input: Menge nicht zugeteilter Nachfrager N, zuzuteilender Nachfrager m

1 $maxfreiheitsgrade = 0$
2 $ausgesuchteranbieter = undef$
3 Für alle $a1 \in (Vorschlge\ zu\ m)$
4 Teile $(m, a1)$ zu.
5 $freiheitsgrade = 0$
6 Für alle $n \in N$
7 Für alle $a \in (Vorschlge\ zu\ n)$
8 Wenn Zuteilung (n, a) keine [schwere] Verletzung
9 $freiheitsgrade = freiheitsgrade + 1$
10 Wenn $freiheitsgrade > maxfreiheitsgrade$
11 $maxfreiheitsgrade = freiheitsgrade$
12 $ausgesuchteranbieter = a1$

Output: Am wenigsten eingeschränktes Objekt: $ausgesuchteranbieter$

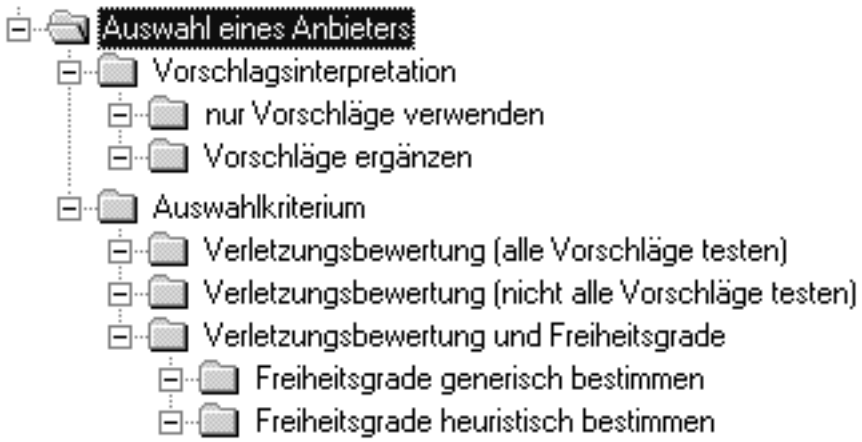

Abbildung 3.3: Übersicht über die Implementationsvariationen bei der Auswahl.

3.2.4 Vergleich mit Auswahl eines Nachfragers

Die Auswahl des Nachfragers wirkt auf die Reihenfolge der Zuteilungen, die Auswahl eines Anbieters bestimmt eine vorläufige Zuteilung. In beiden Fällen lohnt es sich, auf die Freiheitsgrade zu achten, jedoch in unterschiedlicher Weise. Beim Nachfrager ist es vorteilhaft, darauf zu achten, den Nachfrager zu wählen, der am meisten eingeschränkt ist, beim Anbieter dagegen wählt man den Anbieter, der die Folgezuordnungen möglichst wenig einschränkt. Ziel ist es jeweils, sich möglichst geschickt im Suchraum an die Lösung heranzutasten. [Russell & Norvig 1995] nennen diese Heuristiken für Constraint-Satisfaction-Probleme die *most-constrained-variable*-Heuristik bzw. *least-constraining-value*-Heuristik.

3.2.5 Fazit

Abbildung 3.3 zeigt eine Übersicht der Variationsmöglichkeiten bei der Anbieterauswahl. Wie schon bei der Auswahl der Nachfragerobjekte kann man auch bei der Auswahl eines Anbieterobjektes unterschiedlich viel Aufwand betreiben. Gemeinsam mit Varianten bei der Nachfragerauswahl wurden in [Poeck & Puppe 1992] auch die Auswirkungen von unterschiedlichem Berechnungsaufwand bei der Anbieterauswahl untersucht. Je besser die Auswahl in diesen frühen Schritten ist, desto mehr Zeit kann man bei späteren Korrekturen sparen. Bei einfachen Problemen können die Verletzungen auch später noch durch Korrekturen ausgebessert werden, bei komplexeren Problemen jedoch macht sich der Aufwand der Rechenzeit und des Wissenserwerbs bezahlt.

In PEPSI wird die Auswahl wenig einschränkender Anbieter über sehr leicht bewertete Restriktionen gesteuert. Hier verschmilzt das sekundäre Bewertungskriterium mit dem primären. Die Erhaltung der Freiheitsgrade wird Teil der Zielfunktion.

3.3 Vorschlag für einen Restriktionsverletzungstest

Dieser Abschnitt behandelt Schritt 3 des V&V-Algorithmus, das Testen der Restriktionen (vergleiche Abbildung 2.4, Seite 15). Dieser Schritt ist weitgehend klar und es bieten sich keine Alternativen an. Es folgt deshalb statt einer Beschreibung verschiedener Varianten eine Beschreibung der Aufgaben und ein Vorschlag für eine effiziente Implementation eines Restriktionsverletzungstestes.

Durch die Auswahl eines Nachfragers und eines Anbieters in den beiden ersten Schritten des V&V-Algorithmus wurde eine vorläufige Zuteilung definiert. Nun wird getestet, ob aufgrund dieser Zuteilung Restriktionsverletzungen entstanden sind. Dazu werden die Restriktionen der Wissensbasis befragt, ob diese Zuteilung an einer Verletzung beteiligt ist. Wie dies geschieht, ist von der Wissensrepräsentation abhängig, in der Funktionalität gibt es jedoch wenig Variationsmöglichkeiten. Es interessiert dabei nur, ob durch die letzte Zuteilung Restriktionen verletzt wurden. Das Ausmaß einer Verletzung und ob die Verletzung schon vor der Zuteilung bestand, ist zu diesem Zeitpunkt noch irrelevant, da es nur um die Entscheidung geht, ob eine lokale Optimierung (Vertauschen, Änderung der Teilzuordnung) lohnend ist. Liegt keine Verletzung vor, so kann Vertauschen keine lokale Verbesserung der Teillösung bewirken, anderenfalls schon.

Neben der Funktionalität ist es sinnvoll, sich auch über die Effektivität des Restriktionsverletzungstestes Gedanken zu machen. Die Laufzeit eines solchen Testes kann reduziert werden, wenn man nur relevante Restriktionen testet, also Restriktionen, die Objekte betreffen, die durch die Zuteilung verändert wurden. In COKE weiß dazu z.B. jedes Objekt, welche Restriktionen bei einer Zuteilung desselben betroffen sein können. Der Wissensingenieur kann Restriktionen sowohl für ganze Objektklassen als auch für einzelne Objektinstanzen als gültig erklären. So kann man eine Restriktion, die testet ob eine ausreichende Zahl freier Wochenenden besteht, nur für die Nachfragerobjekte Samstags- und Sonntagsschichten geltend machen.

3.4 Lokale Optimierung

Schritt für Schritt wird mit den bisher beschriebenen Algorithmusschritten eine Lösung konstruiert. Zu jedem Zeitpunkt existiert eine Teillösung bzw. partielle Zuordnung. Da zu einer wirklichen Lösung des Zuordnungsproblems alle Nachfrager zugeteilt werden müssen, ist dies noch keine gültige Problemlösung, sie kann jedoch bereits bewertet werden.

Die hier getesteten Restriktionen sollten dafür monoton sein. Nichtmonotone Restriktionen machen in der Regel erst nach vollständiger Zuordnung Sinn. Das soll durch folgendes Beispiel veranschaulicht werden: Bei der Schichtplanung für Pflegepersonal gibt es beispielsweise eine Restriktion, die einzeln auftretende Nachtschichten schlecht bewertet. Diese ist nichtmonoton, da die Verletzung ausgelöst von einer einzelnen Nachtschicht durch die spätere Zuteilung einer anderen behoben werden kann. Bei einer partiellen Zuordnung wäre diese Bewertung jedoch verfrüht, da eine Verletzung durch spätere Zuteilungen noch verbessert werden kann.

Das Nichteinhalten dieser Monotonieeigenschaft bei der lokalen Optimierung hat nicht unbedingt eine Verschlechterung der lokalen Optimierung zufolge. Es wird nur unnötige Zeit zum Korrigieren dieser Restriktion verwendet. Um die Qualität der lokalen Optimierung zu erhalten, ist es jedoch wichtig, darauf zu achten, daß das Verletzungsgewicht nichtmonotoner Restriktionen kleiner ist als das von Restriktionen, deren Verletzungen nicht durch weitere Zuteilungen aufgehoben werden können. Eine solche Bewertung würde eine Verletzung korrigieren, die später beseitigt werden kann, zu Lasten einer Verletzung, die nicht mehr beseitigt werden kann.

Da eine Teilzuordnung also bereits bewertet werden kann, kann man sie auch optimieren. Man nennt diesen Vorgang *lokale Optimierung* (Vergleiche Punkt 4 im Algorithmus; Abbildung 2.4 auf Seite 15). Hier wird versucht, die durch die letzte Zuteilung entstandenen Verletzungen mit Vertauschungen aufzulösen. Dies entspricht einer Suche im Suchraum der partiellen Zuordnungen.

Die Suche in diesem Raum kann auf verschiedene Weise realisiert werden. Dieses Kapitel ist so aufgebaut, daß zunächst die Grundlagen der Suche in Zuordnungsproblemen untersucht werden. Anhand dieser werden verschiedene Varianten zur Umsetzung herausgearbeitet. Der umfangreichere Punkt der Untersuchung von verschiedenen Suchstrategien wird in einem separaten Kapitel (Kapitel 4) behandelt. Unabhängig vom verwendeten Suchalgorithmus gibt es jedoch für die Suche folgende gemeinsame Grundlagen:

- Den Suchraum,
- die Nachbarschaften und
- die Bewertungsfunktion.

Diese Grundlagen sind Thema der nächsten Abschnitte.

3.4.1 Suchraum bei der lokalen Optimierung

Der *Suchraum* bei der lokalen Optimierung ist die Menge aller Teilzuordnungen, bei denen dieselben Nachfrager Anbietern zugeteilt sind, die auch zu Beginn der

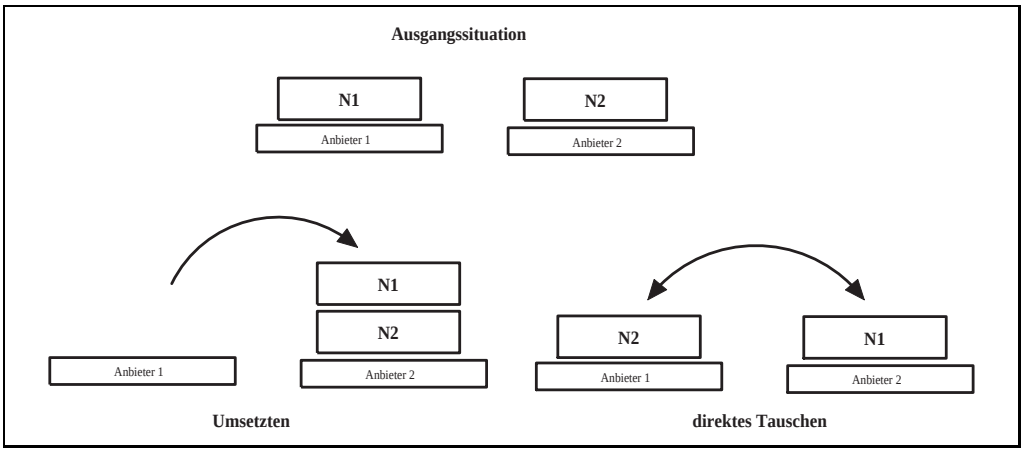

Abbildung 3.4: Mögliche Veränderungen an einer Zuordnung. Die Veränderungen definieren benachbarte Zustände im Suchraum.

Suche zugeteilt waren. Bei den ersten Zuteilungen ist der Suchraum noch sehr klein. Je mehr Zuteilungen bereits gemacht sind, desto größer wird am Ende auch der Suchraum. Sind bereits m von n Nachfragern zugeteilt, und es gibt a Anbieter, so besitzt der Lösungsraum die Mächtigkeit a^m.

Durch Vorschläge, die unsinnige Zuteilungen ausschließen, wird dieser Suchraum zwar reduziert, er bleibt jedoch exponentiell groß. Probleme der realen Anwendungswelt erreichen oft solche Größenordnungen, daß uninformierte[4] Suche wie auch vollständige Suche[5] keine Lösung in vertretbarer Zeit findet.
Folgendes Beispiel hilft einen Eindruck von der Größenordnung zu bekommen: In der Dienstplanung für Zugbegleiter hat man für einen Dienstplan im Schnitt 40 Nachfrager und 12 mögliche Anbieter pro Nachfrager. Das bedeutet, daß der Suchraum die Mächtigkeit 12^{40} hat.

3.4.2 Nachbarschaften bei der lokalen Optimierung

Suchalgorithmen benötigen Transformationen, die *Nachbarschaften* im Suchraum definieren. Im V&V werden diese Transformationen auch „Vertauschungen“ genannt. Im folgenden werden die Nachbarschaften klassifiziert nach der *Veränderungsart*, der verwendeten *Wissenskomponente* und dem *Einsatz der Wissenskomponente*.

Die Veränderungsarten
Jede Veränderung einer partiellen Zuordnung kann als Kombination mehrerer *elementarer Veränderungen* betrachtet werde. Diese sind das *Zurückziehen* einer Zuteilung und das *Setzen* eines zuvor zurückgezogenen Nachfragers[6].

Nachbarzustände können jedoch nicht nur aus diesen elementaren Veränderungen erzeugt werden, sondern auch durch zusammengesetzte Veränderungen.

[4] Eine Suchstrategie heißt informiert, wenn Bewertungswissen zur Suche verwendet wird. Beispiele sind Best-First-Search oder Hillclimbing. Tiefensuche ist uninformiert. Ausführlicher wird dies in [Russell & Norvig 1995] oder später in Kapitel 4.2.2 behandelt.

[5] Eine Suche ist vollständig, wenn sie eine Lösung findet, sofern es eine gibt, d.h. die optimale Teilzuordnung findet.

[6] Die Bedingung, daß er zuvor zurückgezogen wurde, ist wichtig, da sonst der Lösungsraum nicht eingehalten wird.

Direktes Tauschen:

- + Ungültige Zwischenzustände bei der 1-1-Zuordnung werden vermieden.
- – Auf Vorschläge für den Verdrängten wird keine Rücksicht genommen. Es kann beim Tausch eine Zuteilung stattfinden, die nicht vorgeschlagen werden würde.
- – Ist ungeeignet für 1-n-Zuordnung, da gültige Zwischenzustände mit Zuordnungen von mehreren Nachfragern zu einem Anbieter vermieden werden.

Umsetzen:

- + Umsetzen ist mächtiger, eine Vertauschung kann durch zwei Umsetzungen erledigt werden.
- – Eventuell verhindert eine schlechte Zwischenbewertung eine effektive Suche.
- + Ist geeignet für 1-n-Zuordnung.
- – Der Suchraum ist größer, die Anzahl der Nachbarzustände pro Zustand ist jedoch gleich.

Abbildung 3.5: Vorzüge (gekennzeichet mit +) und Nachteile (gekennzeichnet mit –) des direkten Tauschens im Gegensatz zum Umsetzen. Da es sich bei den meisten Zuordnungsproblemen um eine 1-n-Zuordnung handelt, wird in der Regel das Umsetzen besser geeignet sein.

Eine *Umsetzung* ist die Kombination aus Zurückziehen einer Zuteilung und erneutem Setzen dieses Nachfragers. Umsetzen will man Nachfrager verletzter Zuteilungen oder Nachfrager, die bei einer vorgenommenen Zuteilung „stören“. Aus dem Umsetzen entwickelt sich auch das *direkte Tauschen*. Dabei wird der störende Nachfrager an den alten Platz des verdrängenden Nachfragers gesetzt. Dies entspricht einem Platztausch zweier Objekte. Abbildung 3.4 veranschaulicht den Unterschied zwischen Umsetzen und direktem Tauschen. Das Umsetzen ist mächtiger, da sich ein direkter Tausch durch zwei Umsetzungen erreichen läßt. Bei einigen Problemdomänen, insbesondere solchen mit 1-1-Zuordnungen, kann direktes Tauschen vorteilhafter sein. In Abbildung 3.5 werden die Vor- und Nachteile der beiden Veränderungsarten gegenübergestellt.

Generell kann man auch zwischen *einstufigen* und *mehrstufigen* Veränderungen unterscheiden. Eine Veränderung heißt einstufig, wenn die Zuteilung genau eines Nachfragers verändert wurde, andernfalls mehrstufig. Nimmt man zur Menge der Nachbarzustände auch solche, die über mehrstufige Veränderungen erreicht werden, so hat dies verschiedene Wirkungen. Auf der einen Seite wird der Verzweigungsfaktor des Suchbaumes vergrößert - dies ist wenig erstrebenswert, denn es erhöht sowohl den Zeit- als auch den Speicheraufwand der Suche -, auf der anderen Seite können Suchverfahren ohne Backtracking so lokale Minima überwinden. Die Unterscheidung einstufig/mehrstufig wird bei der späteren Evaluierung noch interessant sein. Wenn die Nachbarschaften nur einstufige Veränderungen beinhalten, ist es Aufgabe der Suche, mehrstufige Verletzungskorrekturen zu finden. Wie gut dies gelingt, ist Kriterium zur Bewertung.

Wissenskomponente	angegeben durch	angegeben pro	abhängig von
Initiale Feinde	Startkorrekturen	Restriktion	• Problem • Zuteilung • Verletzung
Störenfriede	Störenfried-funktion	Zuordnungs-problem	• Problem • Zuteilung

Tabelle 3.1: Benennung und Struktur der Wissenselemente zur Korrektur in COKE.

Verwendete Wissenskomponente
Die allgemeinste Methode, die Nachbarschaften ohne Verwendung weiteren Zusatzwissens zu definieren, ist, je einem Nachfrager alle möglichen Anbieter zuzuteilen. Diese Struktur der Nachbarschaften ermöglicht eine vollständige Suche im Suchraum, d.h. von jedem Zustand aus kann jeder andere, insbesondere der optimale, erreicht werden. Zu einem Zustand gibt es $m \cdot a$ Nachbarzustände. Das sind alle möglichen einstufigen Veränderungen der bestehenden partiellen Zuordnung. Im zweiten Suchschritt erreicht man auf diese Weise auch alle möglichen zweistufigen Veränderungen usw. Diese Menge an Zuständen zu testen, ist sehr zeitraubend. Es werden dabei außerdem auch Nachfrager neu zugeordnet, die mit vorliegenden Verletzungen nichts zu tun haben. Ohne Zusatzwissen sind diese nicht bekannt.

Besser ist daher, eine Wissenskomponente zu verwenden, die gezielt angibt, welche Nachfrager eine Zuteilung stören. Diese *Störenfriede* werden dann im Folgeschritt zurückgezogen und erneut zugeteilt. Im Gegensatz zur allgemeinen Vorgehensweise werden so Nachbarzustände, die keine Verbesserung darstellen, von vornherein ausgeschlossen und gezielt Nachbarzustände erzeugt, die eine Verletzung auflösen können.

Eine Verfeinerung dieses Konzeptes ist es, die Störenfriede nicht allein abhängig von der Zuteilung zu bestimmen, sondern abhängig von der Zuteilung und der dadurch erzeugten Verletzung. Der Wissensingenieur kann so genauer seine Lösungsstrategien beim Auftreten einer Verletzung nachbilden. Um dieses Konzept vom vorherigen zu unterscheiden, werden diese Störenfriede in dieser Arbeit *verletzungsabhängige Störenfriede* genannt.

In COKE werden beide Konzepte verwendet. Im ersten Suchschritt werden die Störenfriede verletzungsabhängig bestimmt. Sie werden dort *Initiale Feinde* genannt. In allen folgenden werden sie unabhängig von der Verletzung, allein in Abhängigkeit von der Zuteilung ermittelt. Die Benennung der verschiedenen Wissenselemente ist etwas unterschiedlich zu der hier verwendeten. Tabelle 3.1 stellt die Struktur des Korrekturwissens in COKE dar.

Einsatz der Wissenskomponente
Es wurde bereits angedeutet, daß die Störenfriede einer Zuteilung zurückgezogen werden. Es entsteht ein Zwischenzustand mit zurückgezogenen Nachfragern. Diese werden bei der weiteren Suche gesetzt. Dabei können wiederum Verletzungen entstehen, zu denen Störenfriede ermittelt werden können. Dieses „Schneeballsystem“ kommt zur Ruhe, wenn entweder keine Verletzung mehr besteht oder eine Zeitschranke überschritten wird. Durch den Einsatz des Wissenselementes der Störenfriede ist der Suchraum nicht mehr so stark vernetzt, die Suche

einfache Störenfriede:
$M_1 = \{N_1, N_2, N_3, N_4, N_5, ..., N_i\}$
erweiterte Störenfriede:
$M_2 = \{\{N_1\}, \{N_2\}, \{N_3\}, \{N_4\}, \{N_5\}, ..., \{N_i\}\}$
$M_3 = \{\{N_1, N_2, N_5\}, \{N_1\}, \{N_2\}\}$

Abbildung 3.6: M_1 ist ein Beispiel für eine einfache Störenfriedmenge, M_2, M_3 sind Beispiele für die erweiterte Störenfriedwissensrepräsentation. M_1 und M_2 bewirken dieselben Korrekturen. Bei der Korrektur mit M_3 werden zunächst N_1, N_2 und N_5 zurückgezogen, um den Konflikt zu beheben. Erst danach werden die Nachfrager wieder zugeteilt.

wird jedoch dadurch gelenkt und somit effizienter. Von den Eigenschaften der Korrektur- bzw. Störenfriedfunktionen hängt es ab, ob die Nachbarschaftsstruktur weiterhin vollständig ist und wie schnell eine Lösung gefunden werden kann.

Zum Zurückziehen der Störenfriede werden hier drei Varianten vorgestellt:

- Störenfriede werden einzeln zurückgezogen.
  Es wird jeweils nur ein Störenfried zurückgezogen. Dieser wird im nächsten Suchschritt zunächst wieder gesetzt, bevor neue Störenfriede ermittelt werden. Jeder der erzeugten Zustände hat also maximal einen zurückgezogenen Nachfrager.

- Störenfriede werden alle gemeinsam zurückgezogen.
  Nach einer Verletzung werden alle Störenfriede gleichzeitig zurückgezogen. Bevor erneut zurückgezogen wird, wird zuerst je Suchschritt ein Nachfrager wieder an den Platz zugeteilt, an dem er die beste Bewertung hat. Erst wenn wieder alle zurückgezogenen Nachfrager zugeteilt sind, ist ein erneutes Zurückziehen möglich.

- Störenfriede werden in Gruppen zurückgezogen.
  Die bisher definierte Wissensrepräsentation bietet nur das einzelne oder komplette Zurückziehen der Störenfriede an. Mit einer Erweiterung der Wissensrepräsentation könnte man z.B. ermöglichen, einzelne Nachfrager oder Gruppen von Nachfragern als Störenfriede anzugeben. So können die beiden oben angegebenen Konzepte gemischt werden. Innerhalb einer Gruppe kann man noch eine Reihenfolge zur Neuzuteilung festlegen.

Das einzelne Zurückziehen der Störenfriede erzeugt schneller vollständige Lösungszustände. Beim Zurückziehen aller sind dagegen viele Suchschritte nötig, bis wieder alle zurückgezogenen Nachfrager gesetzt sind und damit ein bewertbarer Zustand erzeugt ist. Es ist daher nur bei kleinen Störenfriedmengen praktikabel. Eine Lösung könnte der dritte Ansatz bieten, da hier gezielt kleine Verursachermengen von der Wissenskomponente identifiziert werden können. Abbildung 3.6 zeigt Beispiele für die verschiedenen möglichen Störenfriedwissensrepräsentationen. Man erkennt auch, daß das Konzept der Störenfriedgruppen mächtiger ist. Die Menge M_1 in der einfachen Repräsentation ist äquivalent zur Menge M_2 in der erweiterten Repräsentation.

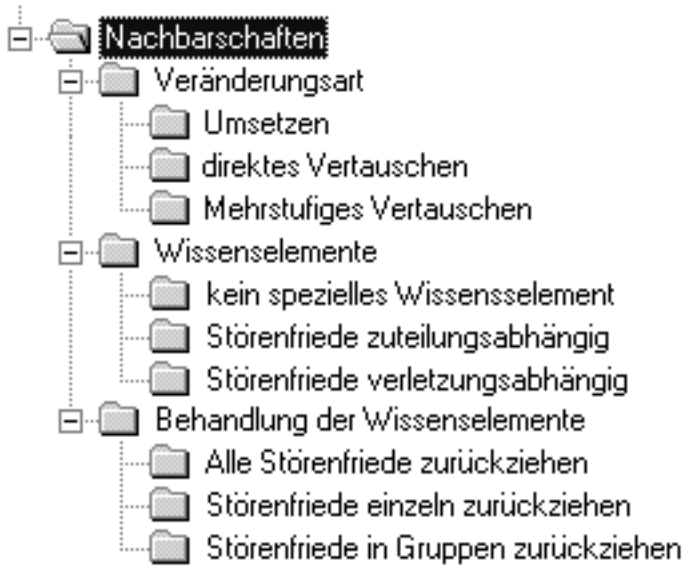

Abbildung 3.7: Ein Überblick über die Varianten für die Nachbarschaftsermittlung bei der lokalen Suche.

In COKE werden „initiale Feinde“ wie Störenfriede bisher einzeln zurückgezogen. Die meisten Verletzungen können auch auf diese einfachere Weise aufgelöst werden. Im Rahmen des Praxisteils dieser Arbeit wird noch eine Variante implementiert werden, bei der alle Störenfriede gemeinsam zurückgezogen werden. Diese wird in Abschnitt 8.2.6 evaluiert. Die Variante, die Störenfriede in Gruppen zurückzuziehen, wurde nicht implementiert, da eine sinnvolle Evaluation dieser Variante einen zusätzlichen Wissenserwerb in den zu untersuchenden Domänen erforden würde. Man müßte sinnvolle Gruppen von störenden Nachfragern zu den jeweiligen Verletzungen definieren. Der Wissenserwerb ist erfahrungsgemäß eine zeitaufwendige Arbeit, die viel Experimentieren erfordert.

Die Hierarchie in Abbildung 3.7 faßt die eben vorgestellten Varianten zur Nachbarschaftsermittlung zusammen. Die Art und Weise, wie die Nachbarzustände ermittelt werden, beeinflußt wesentlich den Erfolg oder Mißerfolg einer Suche. Das beste Suchverfahren versagt, wenn der Zielzustand vom Ausgangszustand nicht erreichbar ist. Auch die Qualität der Wissenskomponente spielt hier eine große Rolle.

3.4.3 Bewertungsfunktion bei der lokalen Optimierung

Informierte Suchstrategien benötigen neben den Nachbarschaften auch Bewertungsfunktionen, die die Suche im strukturierten Suchraum leiten. Aufgrund dieser Bewertungsfunktion wird entschieden, welche Lösungsschritte (Vertauschungen, Umsetzungen) besonders vielversprechend sind, um schnell eine möglichst gute Lösung zu erreichen. Hat man kein weiteres Wissen über Verletzungen, die bei noch ausstehenden Zuteilungen entstehen werden, ist die Bewertung der aktuell verletzten Restriktionen das Hauptkriterium der Bewertungsfunktion. Es gibt nicht notwendigerweise einen absolut bestimmten Zielzustand. Ziel der Suche ist der Zustand mit der minimalen Verletzungsbewertung.

Restschätzer
Viele informierte Suchstrategien verwenden *Restschätzer* als zusätzliche heuristische Information. Der Restschätzer ist Teil der Bewertungsfunktion und gibt in unserem Fall die *Restbewertung* - eine Schätzung der noch zu erwartenden Verletzungen bzw. deren Schwere bis zum Zielzustand - an.

In der lokalen Optimierung ist also für alle Zustände ohne zurückgezogene Nachfrager die Restbewertung gleich 0, da diese Zustände potentielle Lösungszustände sind. Die Angabe einer positiven Restbewertung würde die Suche nach dem Zustand mit dem Minimum der Bewertungsfunktion inkonsistent machen. Für Zustände mit zurückgezogenen Nachfragern kann, da es sich hierbei nicht um mögliche Lösungszustände handelt, eine Schätzung der noch entstehenden Verletzungen angegeben werden. In COKE wird ein solcher Restschätzer verwendet, dies allerdings ohne Schnittstelle zum Wissenserwerb. Stattdessen wird vereinfachend ein Standardverletzungsgewicht einer bestimmten Höhe für einen zurückgezogenen Nachfrager angenommen.

Von dem eben beschriebenen Restschätzer zu unterscheiden ist ein Restschätzer, der eine Bewertung der zu erwartenden Verletzungen von der partiellen Zuordnung zur vollständigen Zuordnung angibt. In den bisher implementierten COKE-Anwendungen gibt es keine Beispiele für eine solche Schätzungsfunktion, es ist jedoch denkbar, daß das Expertenwissen in anderen Problemdomänen eine solche umfaßt. Eine entsprechend formulierte Restriktion kann die Aufgabe eines Restschätzers nachbilden, eine separate Implementation bringt jedoch den Vorteil einer klareren Unterscheidung. Zum einen wird es erleichtert, diese verschiedenen Wissenselemente gesondert zu behandeln, zum anderen ist die Trennung intuitiver und trägt somit zum Verständnis der Benutzer bei. Intern kann der Restschätzer wie eine Restriktion behandelt werden, die, solange noch Zuteilungen ausstehen, als Verletzungsbewertung eine Abschätzung der noch zu erwartenden Verletzungen liefert. Diese Restriktion darf jedoch nicht mehr bei der globalen Optimierung verwendet werden.

Zeiteffizienz
Die Situationsbewertung ist ein wesentlicher Teil der Suche und nimmt viel Zeit in Anspruch. Auch wenn nur wenige Vertauschungen nötig sind, werden oft sehr viele Situationen bewertet, bis eine genügend gute gefunden ist. Man muß deshalb auch darauf achten, die Bewertung zeiteffektiv zu errechnen. Zwei Punkte sollte man dabei beachten:

- Aus der Menge der dem System bekannten Restriktionen müssen nur die mit solcher Objekt-Attribut-Relation getestet werden, die auch durch die letzte Zuteilung betroffen worden sind. Eine Restriktion, die die Arbeitszeit des Personals betrifft, muß nicht bei der Zuteilung von Aufträgen zu Maschinen beachtet werden, wenn dieser Zuordnungsprozeß vorrangig ist und vor der Personalplanung stattfindet.

- Da sich bei einem Vertauschen bzw. Umsetzen nicht die ganze Situation verändert, sondern nur ein oder zwei Zuteilungen, muß man keine gänzlich neue Bewertung mit einem Restriktionstest über alle Nachfrageobjekte durchführen, sondern nur eine Veränderungsbewertung. Es wird also nicht nach den Restriktionsverletzungen der aktuellen Teilzuordnung gefragt, sondern nach den Verletzungen, die nach dem Zurücknehmen einer

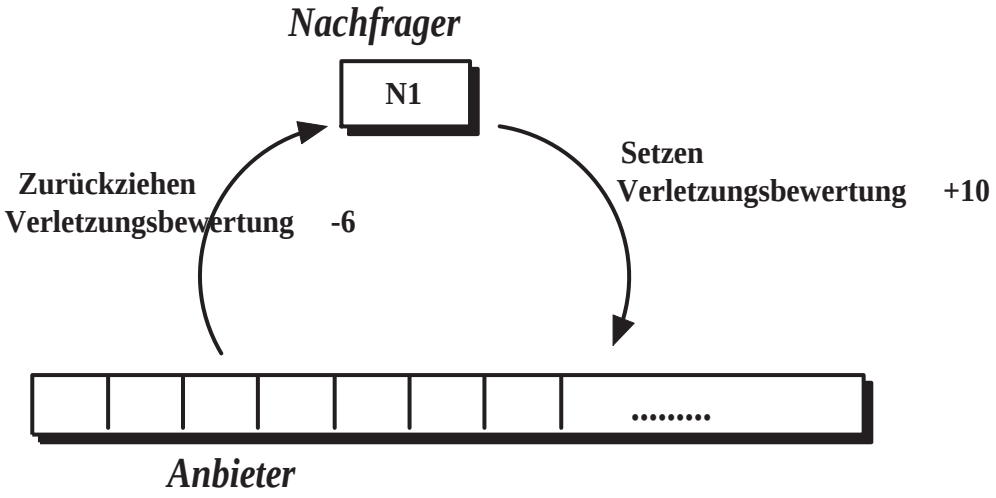

Abbildung 3.8: Beim Zurückziehen eines Nachfragers von einem Anbieter verändert sich die Verletzungsbewertung der Restriktionen, die diese Zuteilung betreffen. Bei einer Neuzuteilung tritt erneut eine Verletzung auf. Zur Beurteilung interessiert nur die Veränderung der Gesamtbewertung durch diesen Umsetzvorgang.

Zuteilung nicht mehr gelten, und den Verletzungen, die nach einer Zuteilung neu entstehen. Abbildung 3.8 zeigt ein Beispiel.

Eine komplette Situationsbewertung für jeden bei der Suche erzeugten Zustand durchzuführen ist sehr aufwendig, bei vielen Nachfrageobjekten sogar nahezu unmöglich. Die Methode, die Veränderungsbewertung statt einer exakten Situationsbewertung zu ermitteln, hat den Nachteil, daß Verletzungsänderungen in „benachbarten“[7] Zuteilungen nicht berücksichtigt werden. Je nach Formulierung der Restriktionen kann dies zu einem Problem führen. Deutlich wird dies am folgenden Beispiel. Eine Schicht an einem Wochentag verletzt eine Restriktion, wenn sowohl am Tag davor als auch am Tag danach eine Schicht zugeordnet ist. Wird die Schicht vom Tag danach umgesetzt, verändert sich die Bewertung unserer mittleren Schicht zum Positiven, dies wird jedoch nicht in der Veränderungsbewertung der Schicht danach sichtbar. Eine Lösung ist, die Restriktion auf folgende Weise zu formulieren: Eine Schicht verletzt die Restriktion, wenn sie Teil eines Dreierblocks von Schichten an aufeinanderfolgenden Tagen ist. So wird die Restriktionsverletzung in allen beteiligten Nachfrageobjekten angezeigt.

Zwei Modelle zur Veränderungsbewertung
Die grundsätzliche Idee der Veränderungsbewertung wurde schon erläutert. Es gibt zwei naheliegende Varianten der Veränderungsbewertung, die im folgenden anhand des Beispiels in Abbildung 3.9 beschrieben werden:

- Veränderung am Störenfried.
  Die Veränderungsbewertung setzt sich zusammen aus einer Bewertungsverringerung in Höhe der ursprünglichen Verletzung des Störenfrieds ($N2$

[7] In vielen Problemdomänen kann man Distanzen auf der Anbietermenge definieren. Restriktionsverletzungen gelten häufig für eine Gruppe von Zuteilungen auf benachbarten Anbietern.

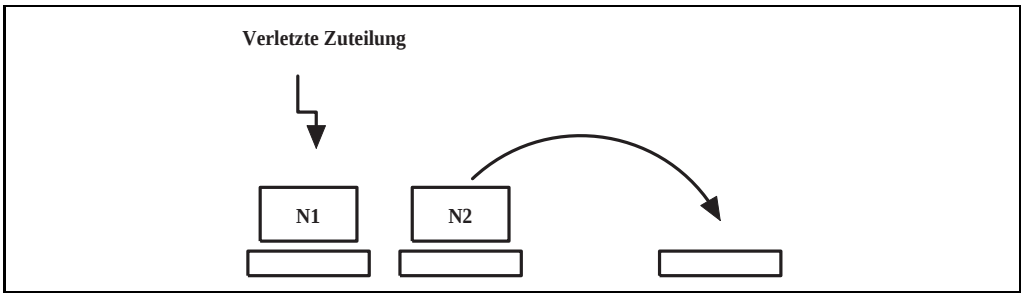

Abbildung 3.9: Das Bild beschreibt folgende Situation: Die Zuteilung des Nachfragers $N1$ ist verletzt. $N2$ wird als Störenfried angegeben und wird deshalb umgesetzt. Das Zurückziehen von $N2$ hebt die Verletzungen an $N2$ auf, beeinflußt jedoch auch die Verletzungsbewertung an $N1$.

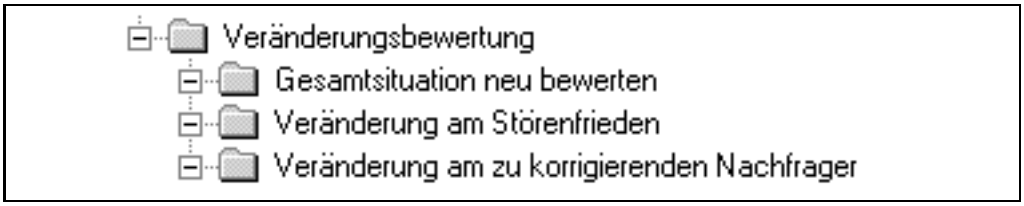

Abbildung 3.10: Varianten bei der Veränderungsbewertung in der lokalen Suche.

im Beispiel) und einer Bewertungserhöhung um die erzeugten Verletzungen durch die erneute Zuteilung. Ob die ursprüngliche Verletzung, deren Korrektur das Ziel war, beseitigt oder verringert wurde, spiegelt sich in dieser Veränderungsbewertung nicht wieder.

- Veränderung am zu korrigierenden Nachfrager.
  Alternativ kann man beim Zurückziehen des Nachfragers $N2$ als Bewertungsverbesserung auch die Differenz zwischen der neuen Bewertung von $N1$ und der alten Bewertung von $N2$ verwenden. So wird zielgerichteter eine Korrektur der Ausgangsverletzung angestrebt.

Fazit

Abbildung 3.10 zeigt die Varianten bei der Ermittlung der Verletzungsbewertung. Eine komplette Verletzungsbewertung während der Suche ist aus Effizienzgründen in der Praxis nicht machbar. Die anderen beiden Varianten ermitteln deshalb lediglich Verletzungsänderungen an Zuteilungen beteiligter Nachfrager. Der Vorteil der effizienten Verletzungsbewertung wird damit erkauft, daß keine quantitativ korrekte Verletzungsbewertung garantiert werden kann und strengere Forderungen an die Formulierung der Restriktionen gestellt werden. So muß z.B. eine Verletzung in allen verursachenden Zuteilungen angezeigt werden. Solche Forderungen müssen dem Programmierer der Restriktionen bekannt sein.

COKE bewertet die Änderung am Initiator einer Korrektur, also am zu korrigierenden Nachfrager. Im Praxisteil der vorliegenden Arbeit wurde als Option die Veränderungsbewerung am Störenfrieden implementiert. Zu erwarten ist, daß die Vorgehensweise, die Veränderungsbewertung am zu korrigierenden Nachfrager zu ermitteln, effektiver ist, da sie nicht irgendeine Verletzungsverbesserung

bewertet, sondern die, die ursprünglich zur Korrektur vorgesehen war. Dies ergibt sich auch aus der später durchgeführten Evaluation (siehe Abschnitt 8.2.3).

3.4.4 Fazit

Die letzten Abschnitte behandelten Grundlagen der Suche (Suchraum, Nachbarschaften, Bewertungsfunktion) und leiteten daraus einige Implementierungsvarianten her. Die Suchstrategien selbst sind so vielfältig, daß ihnen da nächste Kapitel (Kapitel 4) gewidmet wird.

Ein wichtiger Aspekt der Suche soll noch erwähnt werden. Die Suche in „Vorschlagen und Vertauschen“ soll zeitbeschränkt sein. Ziel ist es nicht, unter allen Umständen die beste Lösung zu finden, sondern in einer dem Benutzer vertretbaren Zeit eine möglichst gute Lösung zu finden. Auch die lokale Suche ist deshalb zeitbeschränkt, verschiedene Konzepte dazu werden in Abschnitt 5.2 beschrieben.

3.5 Spezifikation der Zuordnungsschleife

Zum Abschluß der lokalen Optimierung folgt ein Test, ob nach der letzten Zuteilung auch alle Nachfrager zugeteilt sind. Dieser Algorithmusschritt bietet wenig Stoff für Effizienzbetrachtungen. Üblicherweise wird man die nicht zugeteilten Nachfrager in einer Liste[8] speichern. Solange diese Liste Objekte enthält, fährt man mit dem Zuteilungsprozeß fort, ansonsten bricht man ab oder geht zur globalen Optimierung über.

3.6 Globale Optimierung

Ist die Zuordnungsphase beendet, existiert schon eine „vollständige“ Lösung in dem Sinne, daß alle Nachfrager einem Anbieter zugeteilt sind. Es können jedoch immer noch Verletzungen bzw. sogar Konsistenzverletzungen auftreten. Es gibt also zwei Gründe für eine anschließende Optimierung:

- Eine weitere Verbesserung der Lösung:

  Eine weitere, eventuell längere Suche kann Verletzungen auflösen, die bei der lokalen Optimierung nicht beseitigt werden konnten.

- Unterschiedliche Behandlung der Restriktionen:

  Manche Restriktionen ergeben erst Sinn, wenn alle Nachfrager zugeteilt sind. Andere Restriktionen haben nur strategischen Nutzen und sind kein Qualitätsmaß für die Lösung (Beispiele folgen in Abschnitt 3.6.3) und werden nur bei der lokalen Optimierung benötigt.

Die Optimierung der Zuordnung nach Zuteilung aller Nachfrager nennt man *Globale Optimierung*. Sie ist in vielem der lokalen Optimierung ähnlich. Daher orientiert sich der Aufbau dieses Kapitels am Aufbau des Kapitels 3.4. Die

[8] In einer Liste oder einer anderen sortierbaren Datenstruktur, auf die von jedem Teil des Algorithmus zugegriffen werden kann.

einzelnen Abschnitte behandeln die Grundlagen der Suche bei der globalen Optimierung und stellen insbesondere die Unterschiede zur lokalen Optimierung heraus.

3.6.1 Suchraum bei der globalen Optimierung

Der Suchraum bei der globalen Optimierung ist der Raum aller möglichen *vollständigen* Zuordnungen. Im Gegensatz dazu umfaßte die lokale Optimierung partielle Zuordnungen. Bei der letzten Zuteilung stimmt der Suchraum der lokalen Optimierung mit dem Suchraum der globalen Optimierung überein. Die Suche hat jedoch ein anderes Ziel. Ziel der lokalen Optimierung war die Korrektur der aktuellen Zuteilung, Ziel der globalen Optimierung ist die Korrektur aller noch bestehender Verletzungen. Der Suchraum erreicht bei der globalen Optimierung die maximale Größe.

3.6.2 Nachbarschaften bei der globalen Optimierung

In Abschnitt 3.4.2 wurden bereits Veränderungsarten, Wissenselemente und Möglichkeiten zur Bestimmung von Nachfragerzuständen definiert. All dies gilt auch für die globale Optimierung. Da jedoch mehr Verletzungen korrigiert werden sollen, benötigt man ein Verfahren, das darüber hinausgeht.

Das linke Teilbild der Abbildung 3.11 illustriert den Ansatz der lokalen Optimierung. Ausgehend von der aktuell verletzten Zuteilung werden zunächst die Verletzungen ermittelt. Zu jeder dieser Verletzungen werden Störenfriede ermittelt, zurückgezogen und Alternativvorschläge getestet. So ergibt sich die Menge der Nachfolgezustände bei der lokalen Optimierung.

Bei der globalen Optimierung gibt es keine aktuell verletzte Zuteilung mehr, sondern alle verletzten Zuteilungen sind zu optimieren. In der Illustration (Abbildung 3.11) wird dies durch das Zusammenfassen aller Verletzungen angedeutet. Dies vergrößert die Anzahl der im ersten Suchschritt erzeugten Nachfolgezustände sehr und kann die Suche damit ineffizient machen.

Deshalb ist es vorteilhaft, hier eine andere Vorgehensweise zu wählen. Als Alternative bietet sich an, die globale Optimierung als Reihung von mehreren lokalen Optimierungen zu realisieren. Zuteilung für Zuteilung wird eine erneute lokale Optimierung mit den Randbedingungen der globalen Optimierung durchgeführt. Dieses inkrementelle Verbessern entspricht einer Hillclimbing-Suche. Die Nachfolgezustände der Hillclimbing-Suche sind Zustände mit ein- oder mehrstufigen Veränderungen, die das Ergebnis der lokalen Suche sind. Genau genommen hat man damit eine Suche (die lokale Suche) in eine andere (das globale Hillclimbing) eingebettet. Durch das Hillclimbing muß man nur eine eingeschränkte Menge an Verletzungen betrachten. Dies bewirkt, daß die Menge der Nachfolgezustände im ersten Suchschritt und damit die Breite des Suchbaums in den Folgeschritten reduziert wird.

Es gibt verschiedene Möglichkeiten, wie sehr man die Menge der betrachteten Verletzungen einschränkt. Man kann zum einen jede Verletzung einzeln korrigieren, aber auch Gruppen von Verletzungen, die zusammenhängen, gemeinsam verbessern. Sinnvoll ist es z.B., alle Verletzungen einer Zuteilung zusammenzufassen und gleichzeitig zu korrigieren, denn bei der Korrektur einer Verletzung

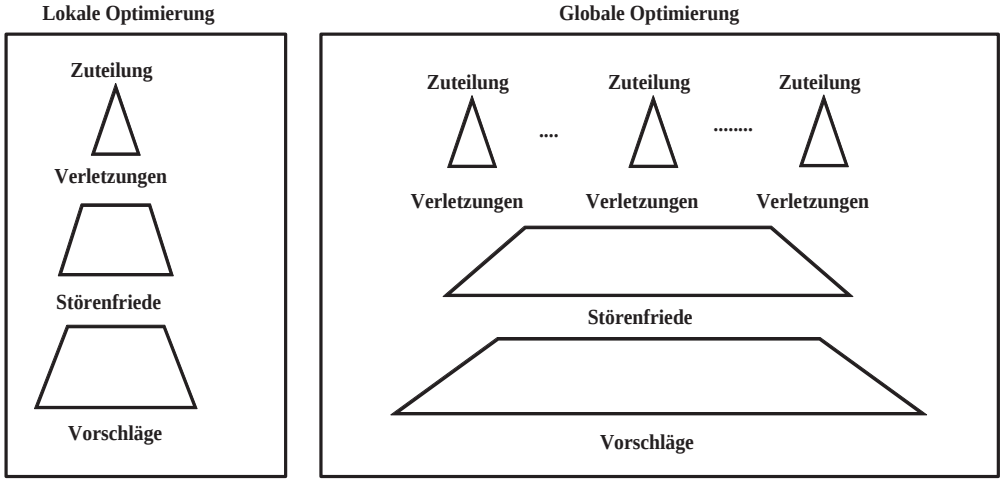

Abbildung 3.11: Die Grafik illustriert den Unterschied des Aufbaus der Menge der Nachfolgerzustände bei der lokalen und globalen Optimierung. Bei der lokalen Optimierung werden nur die Verletzungen zu einer Zuteilung als Ausgangsmenge betrachtet. Zu jeder Verletzung gibt es eine Menge von Störenfrieden und zu jedem Störenfried eine Menge an vorgeschlagenen Anbieterobjekten. Bei der globalen Optimierung sollen die Verletzungen zu allen Zuteilungen optimiert werden.

muß man alle lokalen Veränderungen der Zuteilung bewerten. Dazu gehören auch die anderen Verletzungen der Zuteilung. Die folgende Aufzählung faßt noch einmal die eben vorgestellten Varianten zusammen:

- **Variante 1:**
  Die Verletzungen *aller Zuteilungen* werden gleichzeitig berücksichtigt. Die Nachfolgezustände im ersten Suchschritt sind die Korrekturen zu all diesen Verletzungen. Ein Suchverfahren kann direkt angewendet werden; es ist nicht nötig, das Ergebnis inkrementell zu verbessern.

- **Variante 2:**
  Die Verletzungen *einer Zuteilung* werden gleichzeitig korrigiert. Es wird zunächst die Zuteilung mit der höchsten Verletzungssumme ermittelt. Ausgehend von dieser wird äquivalent zur lokalen Optimierung eine Veränderung gesucht, die die Verletzungen möglichst beseitigt. Solange Zeit zur Verfügung steht, wird dieser Vorgang wiederholt.

- **Variante 3:**
  Jede *einzelne Verletzung* wird für sich korrigiert. Dazu wird die größte Verletzung bezüglich der gesamten Zuordnung ermittelt. Ausgehend von dieser wird nun versucht zu optimieren. Es können hier auch gleichzeitig noch eventuelle andere Verletzungen derselben Zuteilung mitkorrigiert werden. Dieser Ansatz geht in Richtung der Variante 2. Die beiden Varianten unterscheiden sich dann nur noch in der Auswahl der zu korrigierenden Zuteilung.

In COKE wird die Variante 3 zur globalen Optimierung eingesetzt. Damit hat

es gute Unterbrechungseigenschaften, da die Varianten 3 und 2 zunächst zielgerichtet vorgehen und vorrangig schwere Verletzungen (oder stark verletzte Zuteilungen) verbessert werden. Die Lösung ist zu jedem Unterbrechungszeitpunkt entsprechend optimal.

Die Frage, ob eine Suche nach Variante 1 effektiv ist, ist offen. Die starke Verzweigung des Suchbaumes kann die Laufzeit und den Speicherbedarf in die Höhe treiben. In einem Zuordnungsproblem der Ressourcenbelegungsplanung sind z.B. Nachfragermengen der Mächtigkeit bis zu 3500 realistisch [Busch 1997]. Entsprechend viele Zuteilungen können auch bei der globalen Optimierung verletzt sein. Es ist also zu erwarten, daß eine gleichzeitige Optimierung nicht praktikabel ist und deshalb eine inkrementelle Verbesserung Zuteilung für Zuteilung notwendig ist.

3.6.3 Bewertungsfunktion bei der globalen Optimierung

Bezüglich der Restschätzer und der Zeiteffizienz gelten bei der Bewertungsermittlung in der globalen Optimierung dieselben Dinge - insbesondere bieten sich dieselben Varianten - wie bei der lokalen Optimierung. Ein Unterschied liegt in der Verwendung und Bewertung einiger Restriktionen. So gibt es Restriktionen, die in gleicher Weise gültig sind, andere sollen dagegen nicht in jedem Fall verwendet werden.

Typische Restriktionen, die nur für die globale Optimierung gelten, sind solche, die bei Teilzuordnungen verletzt sind, jedoch durch weitere Zuteilungen aufgehoben werden können (nichtmonotone Restriktionen). Als Beispiel kann eine Ruhenrestriktion in der Bahnpersonalplanung genannt werden. Diese Restriktion fordert eine gewisse Anzahl von Ruhephasen im Planungszeitraum. Am Anfang einer Zuordnung stehen diese jedoch noch nicht fest.

Bei der lokalen Optimierung dagegen gelten auch Restriktionen, die kein Qualitätsmaß für die endgültige Lösung bedeuten, sondern als strategisches Wissen zu Beginn der Zuordnung benötigt werden. Ebenfalls aus der Bahnpersonalplanung stammt folgendes Beispiel: Eine Restriktion, die bestimmte Tage eines definierten Ruheschemas möglichst von Schichten freihält, dient dazu, schon frühzeitig auf eine ausreichende Zahl von Ruhen und Wochenendruhen hinzusteuern. Ob in der Lösung letztendlich das Ruheschema eingehalten wird, ist irrelevant; wichtig ist nur, daß die vorgeschriebene Anzahl von Ruhen eingehalten wird (siehe dazu auch [Herrler 1999]). Diese Beispiele zeigen, daß es sowohl Restriktionen gibt, die nur bei der lokalen Optimierung gelten sollten, als auch solche, die nur bei der globalen Optimierung nützlich sind.

Manche Restriktionen gelten zwar in beiden Fällen, sind aber unterschiedlich wichtig für die lokale und globale Optimierung. In einer Problemdomäne, in der es das Ziel ist, eine Maschine möglichst gut mit Aufträgen auszulasten, ist eine Auslastungsrestriktion bei der Zuordnung noch vergleichsweise unwichtig, da sie durch spätere Zuteilungen noch ausgelastet werden kann. Es kann zunächst wichtiger sein, die Aufträge auf Maschinen zu verteilen, die sie am besten abarbeiten können. Bei der globalen Optimierung tritt wieder das Auslastungsziel in den Vordergrund.

Eine Berücksichtigung dieser unterschiedlichen Bewertung und Verwendung von Restriktionen wird in COKE z.B. mit Hilfe von Zuordnungsprofilen realisiert.

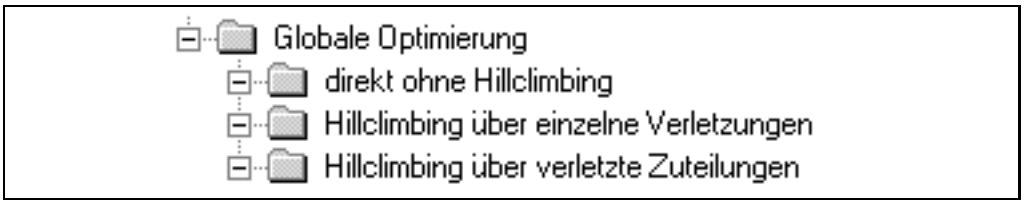

Abbildung 3.12: Varianten bei der globalen Suche.

Verschiedene Profile für globale und lokale Optimierung beinhalten, welche Restriktionen aktiviert sind und welches Verletzungsgewicht sie haben. Die Profile haben noch einen weiteren nützlichen Effekt. Mit ihnen kann man benutzerabhängig verschiedene Gewichtungsschemata verwenden.

3.6.4 Fazit

Die Alternativen bei der globalen Optimierung werden in Abbildung 3.12 zusammengefaßt. Das direkte Optimieren ohne Hillclimbing wurde zwar als wenig aussichtsreich bewertet, wird jedoch der Vollständigkeit halber mit aufgenommen. Die beiden anderen Varianten verwenden eine Hillclimbingsuche und versuchen, das globale Optimierungsziel mit kleinen lokalen Optimierungsschritten zu erreichen.

Man kann die globale Optimierung also sehr leicht als eine lokale Optimierung aller verletzten Zuteilungen implementieren. Beachten muß man dabei, daß bei vielen Zuordnungsproblemen während der globalen Optimierung andere Restriktionen zu berücksichtigen sind als während der Zuordnung.

Kapitel 4

Suchalgorithmen

Bisher wurde vom Algorithmusskelett ausgegangen und jeder Einzelschritt genau daraufhin untersucht, auf welche Weise er sich spezifizieren läßt oder welche der Varianten sich anbieten. Wenn möglich, wurde eine Abschätzung der Effizienz gegeben, an einigen Stellen jedoch kann nur ein Versuch zeigen, welche der Varianten die geeignetste ist. Hier werden nun Problemlösungsalgorithmen für das Problem der Suche betrachtet und es wird analysiert, welche Algorithmen sich mit V&V kombinieren lassen und welche Ideen entsprechend angepaßt auch im V&V-Algorithmus einen Effizienzgewinn bringen können.

Zuordnungsprobleme sind Optimierungsprobleme, die mit Suche gelöst werden können. Der Suchraum ist exponentiell groß. Viele der klassischen Suchalgorithmen sind für sich alleine schon auf Zuordnungsprobleme anwendbar, benötigen jedoch auf Grund des großen Suchraumes zuviel Zeit, um zu einem Ergebnis zu kommen. V&V dagegen ist ein reparaturbasiertes Verfahren, das eine Lösung in einer Mischung aus schrittweiser Konstruktion und Suche nach Verbesserungen erzeugt [Poeck 1995]. Die Suche ist also auch ein wesentlicher Teil des V&V und verdient daher eine ausführliche Betrachtung.

In diesem Kapitel werden einige in der Literatur bekannte Problemlöseparadigmen für Suchprobleme zusammengefaßt. Es wird analysiert, welche von ihnen man zur Optimierung des „Vorschlagen & Vertauschen“-Algorithmus verwenden kann bzw. ob dies schon getan ist.

4.1 Allgemeines für die Suche

Vor einer Betrachtung der Suchalgorithmen ist eine genauere Kenntnis der zugrundeliegenden Suchraumstrukturen notwendig, denn die Beschaffenheit des Suchraumes ist essentiell für die Anwendbarkeit und Wirkung der Suchalgorithmen. Recht ausführlich wurde in Kapitel 3.4.1 bereits der Suchraum bei der lokalen Optimierung beschrieben. Wie jedoch festgestellt wurde, ist die „Vorschlagen und Vertauschen“-Methode eine komplexe Problemlösungsmethode, die sich selbst aus mehreren Suchvorgängen zusammensetzt. Diese sind:

- inkrementelle Suche (Nachfrager für Nachfrager)
- lokale Suche (in jedem der inkrementellen Schritte)

- globale Suche (zur weiteren Optimierung nach der Zuordnung)

Die Suchverfahren werden dabei nicht nur in Serie ausgeführt, sondern sind miteinander verwoben. Die lokale Suche ist eingebettet in die inkrementelle Suche und auch die globale Suche kann als Kombination mehrerer Suchverfahren realisiert werden.

4.1.1 Suchraumtypen

Aus der Analyse der einzelnen Suchvorgänge ergaben sich verschiedene Suchraumstrukturen für die Suche nach Zuordnungen, die im folgenden vorgestellt werden. Danach werden die Suchvorgänge in V&V nach diesen Suchraumtypen klassifiziert.

Der Suchraum kann als gerichteter Graph dargestellt werden oder ausgehend von einem Startzustand als Suchbaum [Nilsson 1980]. Die Knoten des Graphen repräsentieren jeweils einen Zustand, eine gerichtete Verbindung zwischen zwei Knoten symbolisiert eine Transformation, durch die eine *Nachbarschaft*, bzw. der Übergang von einem Zustand zu einem *Nachfolgerzustand* definiert wird. Ein *Lösungszustand* ist ein Zustand im Suchraum, der als Ergebnis der Suche in Frage kommt.

Unterschieden anhand der Transformationen, der möglichen Startzustände und einem Vergleich zwischen der Menge der Zustände und der Lösungszustände ergaben sich folgende drei Haupttypen:

- Konstruktionstyp
- Tauschtyp
- Störenfriedtyp

Diese Typen sind kein allgemeines Klassifikationsschema für Suchräume, sondern wurde in dieser Arbeit speziell für die Anwendung auf Zuordnungsprobleme entworfen.

Der Konstruktionstyp
Abbildung 4.1 zeigt den Konstruktionstyp eines Suchraumes. Startzustand der Suche ist der Zustand, in dem kein Nachfrager zugeteilt ist. Ausgehend von diesem Startzustand werden auf einem Pfad des Lösungsbaumes je abwechselnd ein Nachfrager und ein ihm zuzuteilender Anbieter gewählt. Die Zustandsbeschreibung enthält also eine partielle Zuordnung und gegebenfalls einen zur Zuteilung vorgemerkten Nachfrager. Die Blätter des Suchbaumes bzw. die Enden der Suchpfade repräsentieren die Lösungszustände, das sind all die Zustände, in denen alle Nachfrager zugeteilt sind. Die Menge der Lösungszustände ist also eine echte Teilmenge der Menge der Zustände. Die möglichen Transfomationen eines Zustands in einen anderen sind das Wählen eines Nachfragers (im Startzustand und jedem A_{index}-Zustand) und das Setzen des ausgewählten Nachfragers (in jedem N_{index}-Zustand). Da die Anzahl der Nachfrager beschränkt ist, ist auch die Länge der Suchpfade beschränkt und sogar konstant. Der Suchraum enthält also keine Zyklen. Dennoch können Knoten, die auf verschiedenen Suchpfaden erreicht werden, dieselben Zuordnungen repräsentieren. So ergibt beispielsweise

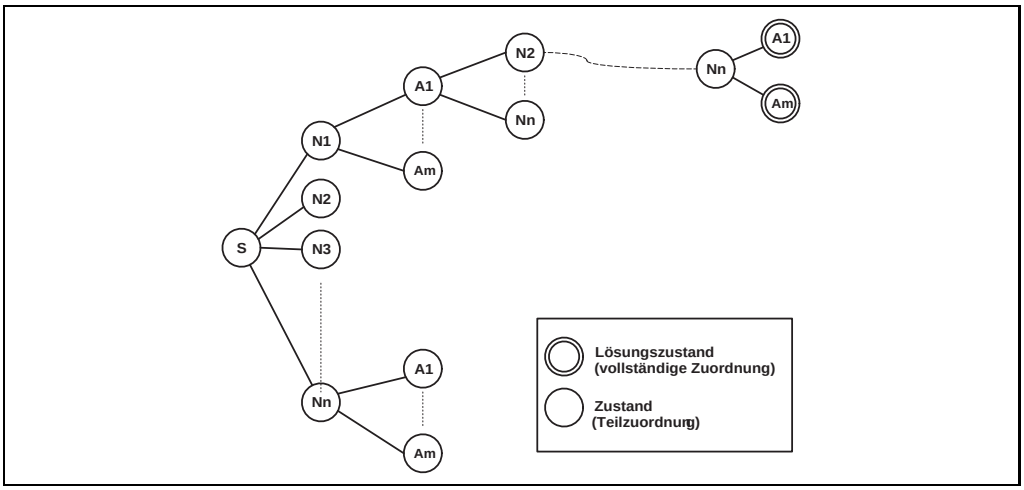

Abbildung 4.1: Beschreibung des Suchraumes für ein „konstruierendes“ Suchverfahren. Die Blätter des Suchbaumes repräsentieren die gültigen Lösungszustände, die durch die auf dem Pfad gewählten Einzelzuteilungen beschrieben werden.

der Suchpfad ($S \Rightarrow N1 \Rightarrow A1 \Rightarrow N2 \Rightarrow A2$) dieselbe partielle Zuteilung wie der Suchpfad ($S \Rightarrow N2 \Rightarrow A2 \Rightarrow N1 \Rightarrow A1$). Bei n Nachfragern gibt es zu einem Lösungszustand $n!$ äquivalente Lösungspfade.

Der Tauschtyp
Beim Tauschtyp (in Abbildung 4.2) ist der Startzustand eine vollständige Zuordnung aller betrachteter Nachfrager. Ausgehend von dieser werden andere Zustände über Veränderungen der aktuellen Zuordnung erreicht. Die möglichen Transformationen sind die in Abschnitt 3.4.2 beschriebenen Veränderungen: Umsetzen und direktes Vertauschen, aber auch mehrstufige Veränderungen, soweit alle zurückgezogenen Nachfrager wieder gesetzt werden. Die Menge der Zustände des Suchraumes ist die Menge der vollständigen Zuordnungen. Jeder Zustand ist also auch Lösungszustand. Für gewöhnlich ist der Suchraum nicht zyklenfrei, deshalb ist die Länge eines Suchpfades auch nicht beschränkt. Ein Zyklus kann z.B. dadurch entstehen, daß eine Veränderung durch eine inverse Veränderung wieder zurückgenommen wird.

Der Störenfriedtyp
Der Störenfriedtyp ist eine Erweiterung des Tauschtyps. Er wurde bereits als typisches Modell bei der lokalen Optimierung vorgestellt (vergleiche Abschnitt 3.4.1) und wird hier noch einmal nach denselben Kriterien wie die beiden vorausgegangenen Suchraumtypen untersucht. Beim Störenfriedtyp (Abbildung 4.3) ist der Startzustand wiederum eine vollständige Zuordnung aller betrachteten Nachfrager. Im Gegensatz zum Tauschtyp (Abbildung 4.2) sind beim Störenfriedtyp auch Transformationen erlaubt, die Nachfrager zurückziehen. Die Menge der Zustände des Suchraumes ist die Menge der vollständigen Zuordnungen (in der Abbildung Z-Zustände) vereinigt mit der Menge aller partiellen Zuordnungen (P-Zustände). Eine Zustandsbeschreibung enthält eine Zuordnung und eine Liste der zurückgezogenen Störenfriede. Nicht jeder Zustand des Suchraumes ist ein gültiger Lösungszustand. Der Suchraum ist wiederum nicht zyklenfrei.

Mit dem Konzept des Störenfried-Suchraumes können verglichen mit dem reinen Tauschtyp besser mehrstufige Veränderungen gesucht werden. Schlechte Zwi-

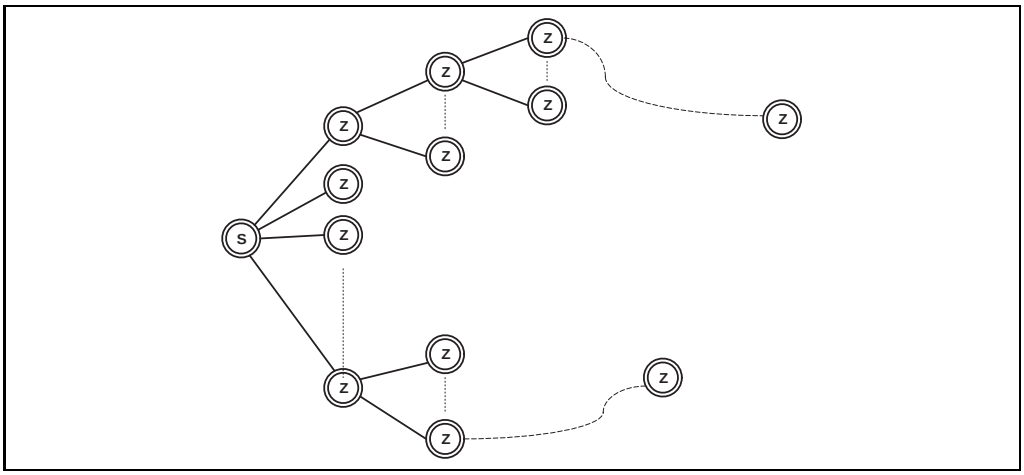

Abbildung 4.2: Eine Darstellung eines zum Baum aufgefächerten Suchraumes vom Tauschtyp für Zuordnungsprobleme.

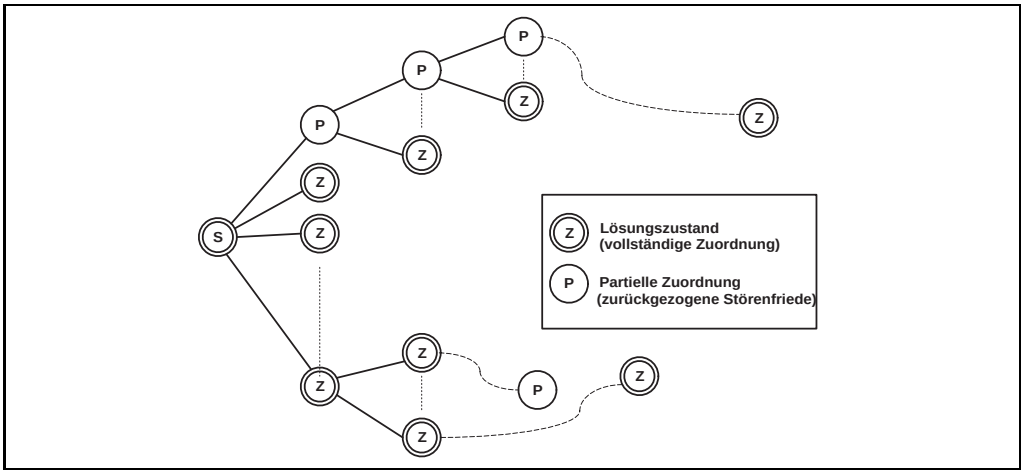

Abbildung 4.3: Eine Darstellung eines zum Baum aufgefächerten Suchraum vom Störenfriedtyp für Zuordnungsprobleme. Die mit P beschrifteten Knoten repräsentieren partielle Zuordnungen, bei denen Störenfriede zurückgezogen sind.

schenbewertungen, wie sie beim reinen Tauschtyp entstehen, können vermieden werden, da Störenfriede, die eine schlechte Zwischenbewertung verursachen würden, rekursiv verdrängt werden. Es ist abhängig von der Qualität des Wissens (Formulierung der Störenfriede), wie gut dieser Mechanismus funktioniert und ob wirklich schlechte Zwischenbewertungen vermieden werden. Das Beispiel in Abbildung 4.4 zeigt die Auflösung eines Konfliktes durch eine mehrstufige Veränderung. Vollständige Zwischenzustände (komplette Zuordnungen) sind sehr schlecht bewertet, die partiellen Zuordnungen mit zurückgezogenen Störenfrieden haben jedoch eine geringe Verletzungsbewertung.

4.1.2 Suchraumcharakteristika im V&V

Es gilt nun, die drei Suchvorgänge aus „Vorschlagen und Vertauschen“ nach den eben beschriebenen Suchraumtypen zu klassifizieren.

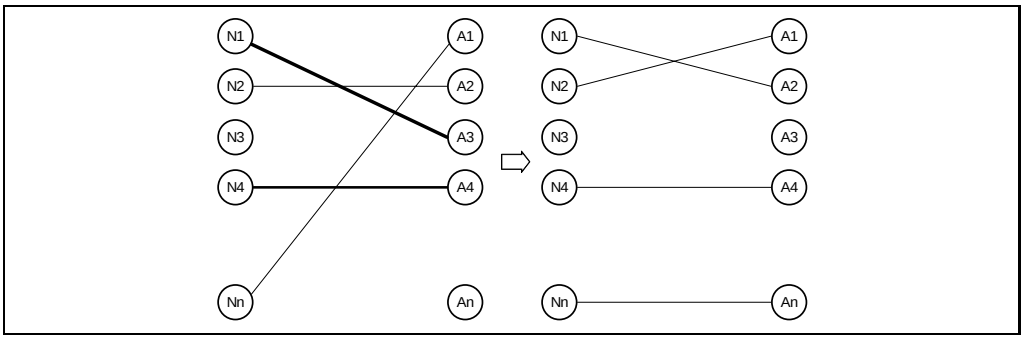

Abbildung 4.4: In der linken Graphik sieht man, daß die gleichzeitige Zuordnung von N1 zu A3 und N4 zu A4, durch die dicken Linien angedeutet, zu einer Bedingungsverletzung führt. Es wird daher zunächst die Zuordnung von N1 zurückgezogen. Das beste mögliche Angebotsobjekt ist das Objekt A2, welches aber bereits von N2 benutzt wird. Daher müssen wir nun ein neues Angebotsobjekt zu N2 suchen. Die beste Möglichkeit ist A1, welches aber bereits von Nn benutzt wird. Das beste neue Angebotsobjekt zu Nn ist An, welches - glücklicherweise - nicht belegt ist. Die Verbesserung terminiert daher. Die verbesserte Zuordnung befindet sich im rechten Teil der Graphik.[Poeck 1995]

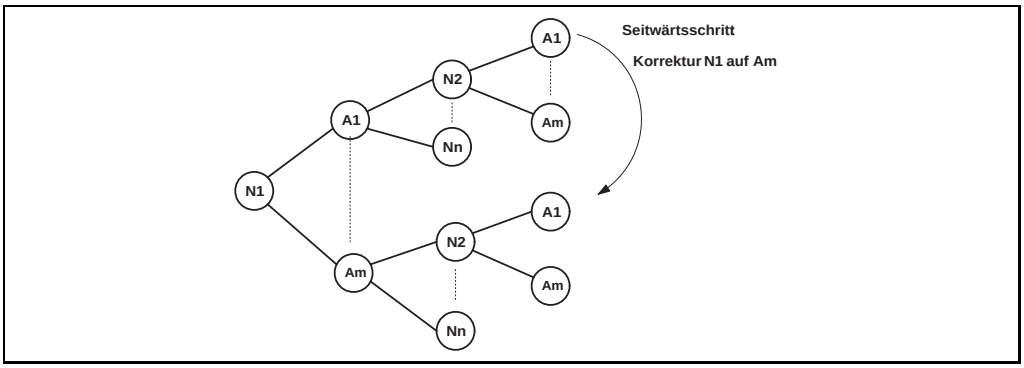

Abbildung 4.5: Beispiel für *Seitwärtsschritte* bei der inkrementellen Suche. Eine Verletzung, die bei der Zuteilung von N2 auf A1 entsteht, wird dadurch korrigiert, daß der Störenfried N1 von Anbieter A1 auf Anbieter Am umgesetzt wird. Dies entspricht einem Seitwärtsschritt im Baum.

Inkrementelle Suche

Die schrittweise Zuteilung eines weiteren Nachfragers im Basisalgorithmus entspricht einer Suche in einem Suchraum der Struktur des Konstruktionstyps. Schritt für Schritt wird ein Nachfrager gewählt und einem Anbieter zugeordnet. Die Länge eines Pfades im Suchbaum ist durch die Anzahl der Nachfrager beschränkt. Durch Korrekturen der lokalen Suche können auch früher zugeteilte Nachfrager umgeordnet werden. Dies entspricht *Seitwärtsschritten* bzw. *-sprüngen* im Suchbaum. Bei der Suche geht man also im Suchbaum niemals wieder nach oben, sondern wechselt durch Korrekturen nur den Ast im Suchbaum [Poeck 1995]. Abbildung 4.5 veranschaulicht einen Seitwärtsschritt.

Lokale Suche

Die Aufgabe der lokalen Suche ist, einen Seitwärtsschritt zu einem besser bewerteten Ast zu finden. Ausgehend von einer partiellen Zuordnung wird eine neue gesucht. Der Suchraumtyp *Konstruktion* paßt daher nicht auf die lokale Suche,

jedoch der *Tauschtyp* und der *Störenfriedtyp*. Bei der allgemeinen Beschreibung dieser Typen wurde gefordert, daß der Ausgangs- und Zielzustand eine vollständige Zuordnung ist. Wir können dieser Forderung damit entsprechen, indem wir die partielle Zuordnung vorübergehend als vollständige Zuordnung eingeschränkt auf die zugeteilten Nachfrager betrachten. COKE verwendet zur lokalen Suche den Störenfriedtyp.

Globale Suche
In Abschnitt 3.6.2 wurde schon festgestellt, daß die globale Suche sich sowohl aus einem Hillclimbing über lokale Verbesserungen zusammensetzen, als auch eine alleinstehende Suche nach einer globalen Verbesserung sein kann. Das Hillclimben von einer vollständigen Zuordnung zur nächsten entspricht einer Suche im Raum des Tauschtyps. Der beste Nachbarzustand wird durch eine lokale Optimierung ermittelt. Da keine Nachfrager zurückgezogen bleiben, kann der Optimierungsprozeß in jedem der Hillclimbingschritte unterbrochen werden, so bleibt die Unterbrechungseigenschaft erhalten.

4.1.3 Wiederholte Zustände und Zyklen

Der Zusammenhang zwischen Suchraumtypen und wiederholten Zuständen wurde bereits angedeutet. Ein Suchbaum kann unter Umständen sehr viel größer als der dazugehörige Suchraum-Graph sein, manchmal sogar unendlich groß. Das bedeutet, daß Suchalgorithmen, wenn keine Vorkehrungen dagegen getroffen wurden, Zustände mehrfach erforschen. Dies macht die Suche ineffizienter. Im Baum des Konstruktionstyps kann jeder Zustand auf genau einem Pfad erreicht werden, bei den beiden anderen Typen sind nicht nur wiederholte Zustände, sondern sogar Zyklen möglich. Für einige Suchverfahren bedeutet dies nur einen Effektivitätsverlust, für andere dagegen kann dies sogar fatal sein, denn sie können in dieser Situation nicht mehr terminieren.

[Russell & Norvig 1995] nennen drei Möglichkeiten, Zyklen entgegenzuwirken.

- Do not return to the state you just came from. Have the expand function[1] (or the operator set) refuse to generate any successor that is the same state as the node's parent.

- Do not create paths with cycles in them. Have the expand function (or the operator set) refuse to generate any successor of a node that is the same as any of the node's ancestors.

- Do not generate any state that was ever generated before. This requires every state that is generated to be kept in memory, resulting in a space complexity of $\mathcal{O}(b^d)$, potentially. It is better to think of this as $\mathcal{O}(s)$, where s is the number of states in the entire state space.

Diese drei Möglichkeiten nehmen in ihrer Reihenfolge an Effektivität und Berechnungsaufwand zu. Die zweite Möglichkeit verhindert bereits Zyklen und damit einen unendlichen Suchbaum, bei der dritten Möglichkeit wird mehrfaches Erforschen von Zuständen gänzlich unterbunden. Die hohe Speicherkomplexität wird allerdings in vielen Fällen einen solchen Test unmöglich machen.

[1] Die „expand function“ ist die Funktion, die die Nachfolgezustände erzeugt.

Die zweite Variante (Erkennung von Zyklen) kann man mit $\mathcal{O}(d)$ Speicheraufwand realisieren, wobei d die Suchtiefe im Baum ist. Der Zeitaufwand zum Wiederherstellen der Vorgängerzustände und zum Vergleich der Zustände ist jedoch nicht unbedeutend.

Ein zeiteffektiver aber nicht optimaler Zyklustest

Aufgrund des eben angesprochenen Problems wird ein zeiteffektiver Zyklustest vorgestellt und weiterentwickelt. Er wird zwar *korrekt* sein, das bedeutet er verhindert alle Zyklen, aber nicht *optimal*, d.h. er verhindert mehr Nachfolgezustände als er müßte. Ich lege als Suchraummodell den *Tauschtyp* zugrunde, denn wie bereits festgestellt wurde, benötigt der *Konstruktionstyp* keinen Zyklustest. Aus dem vorgestellten Zyklustest wird sich sehr einfach ein Zyklustest für den *Störenfriedtyp* herleiten lassen.

- Entwicklungsschritt 1 : einfacher Zyklustest
  Gepeichert werden auf einem Pfad alle Veränderungen als Liste von Tupeln $(Nachfrager, neuerAnbieter)$. Der Zyklustest besteht in der Überprüfung, ob zu einer geplanten Veränderung (N_v, A_v) der Nachfrager schon an einer früheren Veränderung beteiligt war. Ist dies so, wird die geplante Veränderung nicht als Nachfolgezustand akzeptiert. Der Zyklustest ist korrekt, da ein Zustand nicht Nachfolgezustand seiner selbst werden kann, ohne einen zuvor umgesetzten Nachfrager wieder auf den ursprünglichen Anbieter zu setzen. Allerdings werden auch viele andere Nachfolgezustände verhindert, die keinen Zyklus bilden. Die Zuordnungsshell COKE verwendet einen Zyklustest dieser Art.

- Entwicklungsschritt 2 : suboptimaler Zyklus
  Die soeben beschriebene Vorgehensweise kann man sehr einfach erweitern, um den Ausschluß von Nachfolgezuständen, die keinen Zyklus bilden, zu verringern. Dazu speichert man auf einem Pfad die Veränderungen als 3-Tupel $(Nachfrager, alterAnbieter, neuerAnbieter)$. Der Zyklustest besteht darin, zu testen, ob zu einer geplanten Veränderung (N_v, A_v) der Nachfrager schon bei einer früheren Veränderung von diesem Platz wegbewegt wurde. Ist dies so, würde möglicherweise wieder ein Vorgängerzustand hergestellt, also wird die geplante Veränderung nicht akzeptiert. Der Zyklustest ist wiederum korrekt, jedoch werden immer noch Nachfolgezustände ausgeschlossen, die keinen Zyklus erzeugen. Abbildung 4.6 zeigt ein Beispiel. Die Hoffnung ist, daß durch gut formulierte Korrekturfunktionen und Vorschläge dies ausgeglichen werden kann und auch so die nötigen Zustände erreicht werden. Die Ergebnisse, die mit COKE und seinem effizienten aber nicht optimalen Zyklenerkenner erzielt werden, sprechen dafür. Dennoch ist es ein zu erstrebendes Ziel, möglichst wenige Nachfolgezustände fälschlich als Zyklus zu erkennen, um die Reaktion des Systems auf Korrekturen und Vorschläge möglichst durchschaubar zu machen.

- Entwicklungsschritt 3 : effizienter Zyklustest
  Je mehr Berechnungsaufwand man in Kauf nimmt, desto besser kann man den Zyklustest machen. Das dritte Modell eines zeiteffektiven Zyklustestes versucht, das vorige in diese Richtung zu verbessern, ohne jedoch den Berechnungsaufwand eines optimalen Zyklustestes zu erreichen. Mit der expliziten Wiederherstellung der letzten k Zustände kann man Zyklen der

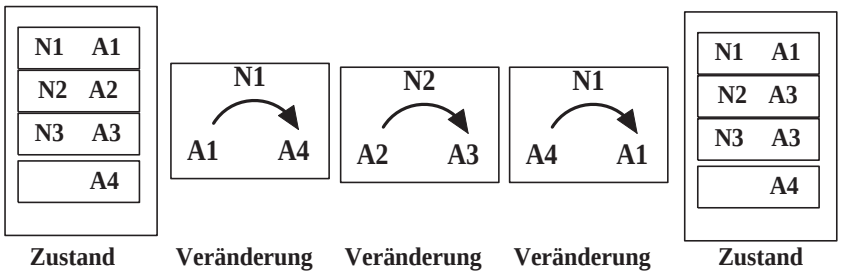

Abbildung 4.6: Das Beispiel zeigt die Fallstricke für zeiteffektive Zyklenerkenner. Da bei der letzten Änderung ein Nachfrager wieder einem früher schon von ihm besetzten Anbieter zugeordnet wird, erlaubt die Nachfolgergenerierungsfunktion eine Zuordnung vorsichtshalber nicht, da es sich um einen Vorgängerzustand handeln könnte. Der Zustand nach der dritten Veränderung wird also nicht angenommen.

Länge k oder kleiner korrekt und optimal erkennen, man bleibt aber, da k konstant ist, bei einer Berechnungskomplexität von $\mathcal{O}(n)$ (wobei n die Anzahl der Nachfrager). Anstatt die letzten k Zustände wiederherzustellen, kann man sie auch in einer Liste speichern. Eine solche Liste nennt man TABU-Liste [Russell & Norvig 1995]. Zur Vermeidung von Zyklen größerer Länge verwendet man wieder den suboptimalen Test. Dabei kann man diesen noch so variieren, daß eine Zuordnung eines bestimmten Nachfragers auf einen bestimmten Anbieter maximal l mal auf dem Pfad erfolgen darf. Zyklen werden also maximal l mal durchlaufen.

- Entwicklungsschritt 4 : speicheraufwendiger Zyklustest
  Das vierte Modell eines Zyklustestes geht einen anderen Weg. Anstatt jeden Zustand wiederherzustellen und ihn mit dem aktuellen zu vergleichen, werden auf einem Pfad alle Zustände in einer geschickten Repräsentation gespeichert und mit dem aktuellen verglichen.

  Da eine Speicherung aller Zustände sehr viel Speicher verbrauchen kann, ist es zunächst notwendig, eine möglichst kompakte Darstellung für die Beschreibung eines Zustandes zu finden. Es gibt a^n verschiedene Zuordnungen und damit Zustände. Wenn man diese Zustände numeriert, benötigt die kürzeste Darstellung $log_2(a^n) = n \cdot log_2(a)$ Bits. Die Zeit für einen Vergleich zweier Zustände liegt damit auch in $\mathcal{O}(n \cdot log_2(a))$.

  Werden die Zustände auf einem Pfad in einem Baum gespeichert, befinden sich in diesem Baum maximal a^n Zustände, bevor ein Zyklus gefunden wird. Deshalb kann nach maximal $log_2(a^n)$ Vergleichen festgestellt werden, ob ein Zustand schon im Baum ist. Die Zeit eines Zyklustestes ist deshalb in $\mathcal{O}((n * log_2 a)^2$ (Zahl der Vergleiche * Vergleichszeit).

  In der Praxis kann die Wahrscheinlichkeit für sehr lange Zyklen gering sein. Dies ist abhängig von der Suchraumstruktur und kann deshalb auch schwer für den Einzelfall nachgewiesen werden. Es ist jedoch für viele Suchverfahren und Wissensrepräsentationen zu erwarten, daß der Worst-Case eines maximalen Zyklus nicht eintritt. In sehr stark vernetzen Suchräumen ist die Lösung auf fast allen Pfaden erreichbar und in schwach vernetzten Suchräumen ist die Gefahr, daß ein sehr großer Zyklus auftritt, nicht sehr groß. Es kann daher effizienter sein, die schon besuchten Zustände in ei-

Berechnungskomplexität	
Vergleich von 2 Zuständen	$\mathcal{O}(n)$
Effektiver Zyklustest Entwicklungsschritt 1	$\mathcal{O}(n)$
Effektiver Zyklustest Entwicklungsschritt 2	$\mathcal{O}(n * a)$
Effektiver Zyklustest Entwicklungsschritt 3	$\mathcal{O}(n + n * a)$
Effektiver Zyklustest Entwicklungsschritt 4	$\mathcal{O}((n * log_2 a)^2)$
Optimaler Zyklustest	$\mathcal{O}(n \cdot a^n)$

Tabelle 4.1: Die Berechnungskomplexitäten der verschiedenen Zyklustests. Es ist n die Anzahl der Nachfrager, a die Anzahl der Anbieter.

ner Hash-Tabelle zu speichern. Bei einer Hash-Tabelle mit n Schlüsseln werden im Worst-Case $\frac{a^n}{n}$ Zustände gespeichert. Die Suche nach einem Zustand liegt also in $\mathcal{O}(\frac{a^n}{n} \cdot n \cdot log(a)) = \mathcal{O}(a^n \cdot log(a))$. Die Average-Case-Berechnungszeit wird jedoch weit darunter liegen.

Ein Problem ist die Speicherkomplexität dieses Modells mit $\mathcal{O}(a^n \cdot n \cdot log_2 a)$. Gegebenenfalls muß man sich damit zufrieden zeigen, die Rekursionstiefe der Suche zu beschränken, obwohl man dabei die Vollständigkeit der Suche verliert. Genau genommen jedoch ist die Vollständigkeit der Suche bei realen Rechnern nie zu garantieren, sondern immer durch den vorhandenen Speicher beschränkt.

Offen bleibt, ob ein so aufwendiger Zyklustest wirklich bedeutende Vorteile bringt und ob die investierte Rechenzeit sich in einer Verbesserung der Ergebnisse niederschlägt.

Fazit

Wiederholte Zustände sind weniger fatal als Zyklen. Wenn sie nicht schon durch die Struktur des Suchraumes verhindert werden, wird es aufgrund der Speicherkomplexität eines entsprechenden Testes nicht praktikabel sein, wiederholte Zustände zu unterbinden. Es bleiben also noch die Fragen: „Wann brauche ich einen Zyklustest?“ und „Wieviel Aufwand stecke ich in den Zyklustest?“. Ob ein Zyklustest erforderlich ist, hängt von der Art des Suchraumes und der verwendeten Suchstrategie ab. Welchen Zyklustest man wählt, ist eine Frage des Tradeoffs zwischen der Rechenzeit und Genauigkeit. Die Vollständigkeit der Nachfolger, also die Menge der Nachfolger, wie sie von den Korrekturen vorgesehen ist, und die Minimierung der Rechenzeit sind dabei die widerstrebenden Ziele. Desweiteren kann man mit Zeitinvestition beim Zyklustest Rechenzeit beim Abarbeiten doppelt besuchter Zustände während der Suche sparen. Hier gilt es, das beste Kosten-Nutzen Verhältnis zu finden. Der Zyklenerkenner in COKE arbeitet ähnlich dem des Modells des Entwicklungsschrittes 1. Der Zyklenerkenner des Entwicklungsschrittes 2 bringt bei wenig Aufwand sicher eine Verbesserung. Entwicklungsschritt 3 ist zugegebenermaßen sehr konstruiert und es ist nicht hervorsehbar, ob sich eine Verbesserung bei realen Problemen daraus ergeben wird. Er zeigt jedoch deutlich den Tradeoff zwischen Exaktheit und Rechenzeit. Interessant wäre schließlich noch, zu untersuchen, ob ein optimaler und korrekter Zyklustest praktikabel ist und welche Vorteile sich daraus ergeben.

Im Praxisteil dieser Arbeit wurde ein Zyklenerkenner des Entwicklungsschrittes 2 umgesetzt. Er wird in Kapitel 8.2.4 mit einem Zyklenkenner des Typs Entwicklungsschritt 1 verglichen. Da sich dabei der geringe Nutzen der ausführlicheren Suche gezeigt hat, wurde auf eine Implementierung der aufwendigeren Zyklustests verzichtet.

4.1.4 Zusammenfassung

Es wurden nun einige Grundlagen geschaffen, die bei bei der Untersuchung der Suche berücksichtigt werden sollen. Zunächst wurden drei Suchraumtypen definiert, da sich die die Suchverfahren unterschiedliche Vor- und Nachteile in den jeweiligen Suchraumtypen haben. Anschließend wurden in V&V drei Vorgänge der Suche in V&V identifiziert und nach den zuvor definierten Suchraumtypen klassifiziert. Als letztes Thema dieses Abschnittes wurde das Zyklenproblem in bestimmten Suchräumen erläutert und Gegenmaßnahmen vorgestellt.

Im den folgenden Abschnitten werden die Suchverfahren Tiefensuche, Hillclimbing, Simulated Annealing und Best First Search in Ihrer Anwendung auf V&V untersucht.

4.2 Tiefensuche

Die Tiefensuche ist ein sehr bekanntes Suchparadigma. Sie zählt bei [Russell & Norvig 1995] zu den uninformierten Suchverfahren. Expandiert wird jeweils der tiefste innere Knoten, bis ein Blatt (ein Zustand ohne Nachfolgezustände) erreicht wird. Dieser wird bewertet und, falls es kein sicherer Zielzustand[2] ist, geht die Suche im Baum zurück (zum tiefsten inneren Knoten). Eine Bewertung der inneren Knoten findet bei einfacher Tiefensuche üblicherweise nicht statt. Denn Aufgabe der Bewertungsfunktion ist nicht, die Suche zu steuern, sondern die Blätter zu bewerten. Wenn innere Knoten auch Lösungszustände sein können (wie bei Tauschtyp und beim Störenfriedtyp), ist es notwendig, auch diese zu bewerten, dennoch verwendet Tiefensuche diese Bewertung nicht zur Auswahl des am frühesten zu expandierenden Nachfolgezustands.

4.2.1 Naive Tiefensuche

Zuordnungsprobleme kann man algorithmisch sehr einfach mit einer reinen Tiefensuche lösen. Dieser Abschnitt macht einen kleinen Exkurs von „Vorschlagen und Vertauschen“. Gezeigt werden soll, daß die Suchstrategie allein nicht der Garant für eine gute Lösung ist, sondern daß es auf den Einsatz der Suchstrategien ankommt. Später (Abschnitt 4.2.2) werden dann geeignete Einsatzstellen in V&V gezeigt.

Der naive Ansatz besteht darin, Tiefensuche auf einen Suchraum der Struktur des Konstruktionstyps (vergleiche Abbildung 4.1) anzuwenden. Bewertet wird jeder besuchte Blattzustand, bis entweder eine Zuordnung ohne Verletzungen gefunden oder der Suchraum erschöpft ist. Für den Fall, daß es keine Zuordnung ohne Verletzung gibt, wird bei der Suche die jeweils beste Zuordnung

[2] „sicher“ bedeutet hier, daß keine Verletzung mehr besteht.

gemerkt. Aufgrund der Größe des Suchraumes ist mit dieser Problemlösungsmethode sogar die Lösung von sehr leichten Zuordnungsproblemen nicht mehr praktikabel. Bei n Nachfragern und durchschnittlich p Vorschlägen pro Nachfrager hat ein Suchbaum des Konstruktionstyps p^n Blätter bzw. Lösungen, die bewertet werden.

4.2.2 Tiefensuche in V&V

Geschickter ist es, die Tiefensuche nicht direkt anzuwenden, sondern z.B. innerhalb eines verbesserungsbasierten Verfahrens wie dem „Vorschlagen und Vertauschen"-Algorithmus. Dort gibt es, wie in Kapitel 4.1 beschrieben, drei Suchvorgänge, die inkrementelle Suche, die lokale Suche und die globale Suche.

Eine Tiefensuche bei der inkrementellen Suche entspricht in etwa dem beschriebenen naiven Ansatz. Im Unterschied zur naiven Suche können durch die Korrekturen der lokalen Optimierung im Idealfall[3] mehr Pfade zu Lösungszuständen führen. Allerdings verliert die Tiefensuche dadurch auch die Vollständigkeit, d.h. bei ungeschickt formulierten Korrekturen kann es dazu kommen, daß ein Zielzustand nicht erreicht wird, da er durch eine Korrektur verhindert wird.

Bei der lokalen Optimierung ist das Ausgangsziel der Suche soweit eingeschränkt (Verletzungsbeseitigung einer Zuteilung statt Beseitigung aller Verletzungen), daß Tiefensuche für die Suche nach lokalen Verbesserungen prinzipiell geeignet ist. Probleme, die sich dabei dennoch ergeben, und die Vorteile werden im nächsten Abschnitt (Abschnitt 4.2.3) beschrieben.

Bei der globalen Suche bieten sich, wie in Kapitel 3.6.2 festgestellt, Suchstrategien an, die ein inkrementelles Verbessern ermöglichen, wie z.B. Hillclimbing. Tiefensuche kann hier innerhalb des Hillclimbing eingesetzt werden. Aus Komplexitätsgründen ist eine direkt angewandte Tiefensuche wenig erfolgversprechend.

4.2.3 Probleme und Vorteile der Tiefensuche

Vorteil der Tiefensuche ist der vergleichsweise geringe Speicherbedarf während der Suche. Bei einem Verzweigungsfaktor der Größe b und einer Suchtiefe von m benötigt man Speicherplatz für nur $b \cdot m$ Knoten[4] [Russell & Norvig 1995]. Die Zeitkomplexität ist dennoch sogar im average-case exponentiell.

Beim Tauschtyp und beim Störenfriedtyp kann der Suchraum Zyklen enthalten. Das bedeutet, daß der aufgefaltete Suchbaum der lokalen Optimierung unendlich tief werden kann. Eine reine Tiefensuche terminiert nie, wenn sie in einen Zyklus läuft. Notwendig ist deshalb eine Strategie, um dies zu verhindern. Zum einen ist es möglich, den Suchpfad explizit auf Zyklen zu testen (siehe Kapitel 4.1.3). Eine andere Möglichkeit besteht darin, die Tiefensuche zu beschränken. Das kann einerseits eine zeitliche Schranke sein, als auch eine Beschränkung der Suchtiefe. Es können also folgende Techniken einzeln oder in Kombination zur Vermeidung von Zyklen eingesetzt werden:

[3] Das hängt von der Güte der lokalen Optimierung ab.
[4] Zum Vergleich: Breitensuche benötigt n^m Knoten.

- Zyklustest
  Ein Zyklustest kann verschieden aufwendig durchgeführt werden. Zeiteffiziente Tests unterbrechen die Tiefensuche auf einem Pfad bereits, wenn ein Verdacht auf einen Zyklus besteht, die Annahme aber nicht unbedingt gesichert ist. (vergleiche Kapitel 4.1.3)

- Beschränkung der Tiefensuche
  Nach einer Beschränkung sind zwar noch Zyklen möglich; da die Suche jedoch in der Tiefe beschränkt ist, wird er nur endlich oft durchlaufen und die Suche terminiert. Der offensichtliche Nachteil dieser Methode ist, daß die Erforschung des Suchraumes begrenzt wird.

Ein weiteres Problem bleibt die Größe des Suchraumes und die Reihenfolge der Knoten-Expandierung. Wenn die Korrektur einer lokalen Verletzung mit wenigen Tauschschritten möglich ist, kann es trotzdem sein, daß Tiefensuche zunächst in einem sehr tiefen Pfad (dieser entspricht vielen Vertauschungen) sucht. Eine mögliche Gegenmaßnahme ist hier wiederum die Beschränkung der Tiefensuche oder eine Iterative-Deepening-Suche, wie sie in Abschnitt 4.2.4 beschrieben wird.

Den Suchbaum verkleinern kann man durch Abschneiden von Ästen, ähnlich wie dies bei Spielbäumen gemacht wird (siehe Alpha-Beta-Pruning in [Russell & Norvig 1995]). Voraussetzung für das Abschneiden von Suchästen ist die Annahme, daß weitere Zuteilungen das Verletzungsgewicht nur erhöhen und nicht verringern. Für diese Annahme dürfen bei der lokalen Optimierung nur Restriktionen aktiv sein, die diese Forderung erfüllen. Wenn dies gilt, kann man Äste an Knoten mit partiellen Zuordnungen, deren Verletzungsgewicht höher als das eines zuvor gefundenen Lösungszustandes ist, abschneiden. Das Zuteilen der zurückgezogenen Störenfriede könnte keine Verbesserung des Zustandes ergeben. Ein Expandieren dieses Zustandes ist also nicht erforderlich.

4.2.4 Beschränkte Tiefensuche

In einigen Fällen ist die Tiefensuche von vornherein beschränkt. Dies ist z.B. bei einer konstruktiven Vorgehensweise wie beim *Konstruktionstyp* der Fall. Die Tiefe eines Baums ist hier durch die Anzahl der zuzuteilenden Nachfrager festgelegt. Auch bei den anderen hier definierten Suchbaumtypen ist es sinnvoll, die Tiefensuche zu beschränken, um Zyklen zu vermeiden und die Laufzeit abschätzen zu können. Eine Grenze für die Suche ist hier nicht so einfach zu bestimmen. Im günstigsten Falle ist die Grenze möglichst klein und es werden doch alle Zustände erreicht. [Russell & Norvig 1995] definieren den *Durchmesser eines Suchraumes* als die minimale Pfadlänge, die notwendig ist, um von einem beliebigen Zustand aus jeden anderen erreichen zu können. Dieser Wert ist abhängig von der Vernetzung des Suchraumes, die durch Vorschläge und Korrekturen bestimmt ist. Man kann also keine allgemeine Aussage über den Durchmesser des Suchraumes treffen. Die Korrekturen können sogar so beschaffen sein, daß bestimmte Zustände gar nicht erreicht werden können. Der stark wissensbasierte Ansatz der V&V-Problemlösungsmethode steht hier also in Konkurrenz zur Möglichkeit, eine sichere Grenze für die Tiefensuche angeben zu können.

Auch ohne eine sichere Grenze für die beschränkte Tiefensuche kann man dennoch eine vollständige Tiefensuche erreichen, indem man die Suchtiefe schritt-

Problempunkt	Maßnahme
Großer Suchraum	• Abschneiden von Suchästen • Beschränkung der Tiefensuche • Iterative Deepening
Zyklen	• Zyklustest • Beschränkung der Tiefensuche
Suchreihenfolge	• Iterative Deepening

Tabelle 4.2: Die Tabelle zeigt die Probleme bei der Anwendung der Tiefensuche zur lokalen Optimierung und Möglichkeiten zur Gegenmaßnahme.

weise erhöht. Diese *Iterative Deepening*-Suche braucht nur doppelt soviel Zeit wie eine Breitensuche, benötigt dabei aber wesentlich weniger Speicherplatz [Russell & Norvig 1995].

4.2.5 Fazit

Das große Manko der Tiefensuche ist, daß sie ein uninformiertes Suchverfahren ist und sie die ja schon zur Verfügung stehende Bewertungsheuristik nicht verwendet. Ein Vorteil gegenüber informierten Suchverfahren (z.B. Bestensuche, siehe 4.5) könnte darin liegen, daß die Tiefensuche weniger Speicher zur Suche benötigt, und damit größere Bereiche des Suchraumes durchsuchen kann. Es kann jedoch durchaus sein, daß die dazu benötigte Zeit den Vorteil wieder zunichte macht.

Im Praxisteil dieser Arbeit wird Tiefensuche als Option zu Hillclimbing und Bestensuche implementiert, um festzustellen, ob sich die theoretischen Überlegungen bestätigen. Die Tiefensuche wird in Abschnitt 8.2.8 evaluiert.

Möglicher Einsatzbereich der Tiefensuche in V&V ist die Phase der lokalen Optimierung. Problematisch für die Tiefensuche ist der große Suchraum, das mögliche Auftreten von Zyklen und die ungünstige Suchreihenfolge, die sich bei der Tiefensuche ergibt. Teilweise kann man diesen Problemen entgegenwirken, z.B. mit der Verwendung eines Zyklustestes, durch eine Beschränkung der Tiefensuche oder durch das Abschneiden von Suchästen, in denen keine Lösung liegen kann. Tabelle 4.2 zeigt noch einmal einen Überblick über die Maßnahmen, die die Tiefensuche effizienter machen können. Für den Praxisteil dieser Arbeit wurde die Tiefensuche als Option in COKE implementiert und wird in Abschnitt 8.2.8 evaluiert.

4.3 Hillclimbing

Die grundlegende Idee der Hillclimbing-Suche ist, mit einem Zustand des Suchraumes zu beginnen und von dort aus iterativ bessere Nachbarzustände zu suchen, bis ein Zielzustand erreicht ist. Im Unterschied zur Tiefensuche kennt dieses Suchverfahren nur den jeweiligen aktuellen Zustand, und nicht den bisherigen Suchpfad. Das Algorithmusbild 3 auf Seite 62 zeigt den Algorithmus. Ob ein Nachbarzustand „besser“ ist, entscheidet eine heuristische Funktion, bei der Zuordnung die Verletzungsbewertungsfunktion. Da es dabei um eine Minimierung der Bewertung geht, statt das Maximum der heuristischen Funktion zu „erklimmen“, könnte man für „Hillclimbing“ auch genauer „gradient descent“[Russell & Norvig 1995] oder „valley descending“[Rich & Knight 1991] sagen.

[Rich & Knight 1991] unterscheiden zwischen *absoluten Lösungen* und *relativen Lösungen*. Besitzt ein Problem absolute Lösungen, kann allein durch die Untersuchung des aktuellen Zustandes erkannt werden, ob es sich um einen Lösungszustand handelt. Ein Problem besitzt relative Lösungen, wenn es sich, wie bei den Zuordnungsproblemen, um ein Optimierungsproblem handelt. Lösungszustände sind solche, die besser als alle anderen Zustände sind, oder approximativ, die besser sind als die meisten der anderen Zustände sind und eine bestimmte Mindestgüte besitzen (Keine Konsistenzverletzungen / Schwere Verletzungen). Hillclimbing bei Zuordnungsproblemen terminiert also, wenn von einem Zustand aus kein besserer Nachbarzustand mehr erreicht werden kann.

Reines Hillclimbing läuft aus diesem Grund leicht in Gefahr, in *lokalen Maxima* zu terminieren - also Zuständen, die nicht der optimalen Lösung (dem globalen Maximum) entsprechen und bei denen in unmittelbarer Nachbarschaft kein besserer Zustand erreicht werden kann. Ein weiteres Problem sind *Plateaus*, Nachbarzustände gleicher Bewertung, die keinen höher bewerteten Zustand in ihrer Umgebung besitzen. Da das Hillclimbing kein Gedächtnis für schon besuchte Zustände hat, kann bei Plateaus das Zyklusproblem auftreten[5].

4.3.1 Reines Hillclimbing

Wie schon bei der Tiefensuche wird auch hier zunächst die reine Anwendung der Suchstrategie (ohne V&V) auf Zuordnungsprobleme untersucht.

[Poeck 1995] stellt reines Hillclimbing in einem Suchraum der Struktur des Konstruktionstyps vor. Er stellt fest:

> „Der offensichtliche Nachteil dieses Verfahrens ist es, daß eine Lösung erzeugt werden kann, bei der Konsistenzbedingungen verletzt sind.“

Ein solches Hillclimbing im Konstruktionssuchraum entspricht einem schrittweisen Setzen der Nachfrager. Einmal gemachte Fehler (Zuteilungen die aus späterer Sicht doch nicht so gut waren), können nicht mehr verbessert werden. Es ergibt sich ein gewissermaßen ein „Vorschlagen und Vertauschen“ ohne Vertauschen-Phase. Die Ergebnisse einer solchen Strategie sind nur bei sehr einfachen Problemen überhaupt von Nutzen.

[5] Das Zyklusproblem tritt nur auf, wenn auch gleich gut bewertete Nachbarzustände besucht werden

Nach [Russell & Norvig 1995] können Iterative-Improvement-Algorithmen[6] gerade für eine Suche im Tauschsuchraum oder im Störenfriedsuchraum von Nutzen sein.

> „We saw .. that several well-known problems (for example, 8-queens and VLSI layout) have the property, that the state description itself contains all the information needed for a solution. The path by which the solution is reached is irrelevant. In such cases, **iterative improvement** algorithms often provide the most practical approach“

Beim Tauschtyp und Störenfriedtyp gelten genau diese Bedingungen. Die Zustandsbeschreibung enthält eine partielle oder vollständige Zuordnung, im Gegensatz zum Konstruktionstyp, bei dem der Pfad im Suchraum die Zuordnung definiert. Das erwähnte 8-Damen-Problem kann übrigens auch als Zuordnungsproblem definiert werden (siehe [Klügl 1994]).

Probleme hat Hillclimbing, wenn Verletzungen mehrstufige Korrekturen benötigen und sich nicht durch einfaches Umsetzen lösen lassen. In diesem Fall versperrt eine schlechte Zwischenbewertung den Pfad zur Lösung. Man bleibt in einem lokalen Minimum stecken.

4.3.2 Hillclimbing im V&V

Dieser Abschnitt soll zeigen, wie man Hillclimbing in der „Vorschlagen und Vertauschen“-Methode verwenden kann. Zum einen kann die Grundstruktur der Methode als Hillclimbing interpretiert werden und zum anderen kann bei der lokalen oder globalen Suche Hillclimbing zur Suche nach Verbesserungen verwendet werden.

Hillclimbing bei der inkrementellen Suche
Die V&V-Grundstruktur bzw. die inkrementelle Suche Nachfrager für Nachfrager ist eine Suche im Konstruktionssuchraum erweitert durch die Möglichkeit zu Seitwärtsschritten (siehe Abschnitt 4.1.2). Algorithmus 5 zeigt eine Hillclimbing-Variante für den Konstruktionssuchraum. Um aus diesem Algorithmus einen „Vorschlagen und Vertauschen“-Algorithmus zu machen, muß man an die Nachfolgerbestimmung in Zeile 4 nur noch eine lokale Optimierung der Zuordnung anhängen.

In Abschitt 4.3.1 wurde die schlechte Eigenschaft des Hillclimbing beim Konstruktionstyp festgestellt: Eine einmal vorgenommene Zuteilung kann nicht nachträglich verbessert werden. Durch den Vertauschen-Schritt, bzw. die lokale Optimierung wird dieser Nachteil in V&V aufgehoben. Das bewirkt gewissermaßen eine Verbesserung der Struktur des Suchraumes. Mehr Pfade können zur Lösung führen, da auch „ungeschickte“, für das Optimierungsziel wiedersprüchliche Zuteilungen wieder korrigiert werden können.

Hillclimbing bei der lokalen Suche
Hillclimbing bei der lokalen Suche ist vergleichbar mit reinem Hillclimben ohne

[6] Hillclimbing und Simulated Annealing sind Iterative Improvement Algorithmen [Russell & Norvig 1995].

„Vorschlagen und Vertauschen“. Es handelt sich um eine Suche in einem Suchraum des Typs Störenfried- oder Konstruktionssuchraum. In vielen Fällen, gerade wenn Verletzungsauflösungen mehrstufige Veränderungen erfordern, kann das Problem der lokalen Maxima diese Vorgehensweise inpraktikabel machen. [Russell & Norvig 1995] nennen als Gegenmaßnahme zum Problem der lokalen Maxima die Technik der *random restarts*. Restarts mit anderen zufälligen Ausgangszuständen sind jedoch bei der lokalen Optimierung nicht möglich, da der Ausgangszustand für die Optimierung schon festgelegt ist. Der Erfolg des Hillclimbings hängt also wesentlich von der Güte einer Bewertungsfunktion und der Definition der Nachbarschaften ab. Nachbarschaftsdefinitionen, die selbst schon mehrstufige Veränderungen erzeugen, sind hier besser als die alleinige Verwendung einstufiger Veränderungen.

Hillclimbing bei der globalen Suche
In Kapitel 3.6.2 wurde bereits festgestellt, daß sich Hillclimbing-Varianten sehr für den Einsatz bei der globalen Optimierung eignen. Sie versuchen, die Gesamtsituation nicht direkt zu verbessern, sondern Schritt für Schritt Verletzungen oder Gruppen von Verletzungen beseitigen. Algorithmus 4 zeigt einen angepaßten Hillclimbingalgorithmus dafür. Solange Verletzungen bestehen, wird versucht, eine ausgewählte Verletzung (üblicherweise die schwerste) zu beseitigen. Die Suche nach einer Verletzungskorrektur kann auf beliebige Weise erfolgen. Ein dazu eingebettetes anderes Suchverfahren definiert die Nachbarzustände für die globale Hillclimbing-Suche. Wie schon bei der inkrementellen Suche relativiert die Kombination mit einem anderen Suchverfahren das Problem der lokalen Maxima.

Algorithmus 3 Basis-Hillclimbing-Algorithmus. Hier in der Variante Steepest Ascent Hillclimbing.

Input: Startzustand S

1 $aktuell = S$
2 $ergebnis = undef$
3 Solange $ergebnis$ undefiniert
4 $\quad naechster =$Nachfolgezustand mit kleinster Verletzungsbewertung
5 $\quad$ Wenn Verletzungsbewertung($naechster$)>Verletzungsbewertung($aktuell$)
6 $\quad\quad ergebnis = aktuell$
7 $\quad aktuell = naechster$

Output: Lösungszustand: $ergebnis$

4.3.3 Fazit

[Russell & Norvig 1995] stellen allgemein fest:

> „Thus it is still true that hill-climbing can be very inefficient in large, rough problem space. But it is often useful when combined with other methods that get it started in the general neighborhood.“

Auch speziell auf Zuordnungsprobleme ist diese Aussage treffend. Aufgrund des Problems der lokalen Minima ist Hillclimbing für sich allein keine geeignete

Algorithmus 4 Hillclimbing über die Verletzungen. Die Optimierungsfunktion bestimmt den besten „Nachfolger“. Dies kann z.B. auch über eine eingebettete Bestensuche geschehen.

Input: Startzustand (Zuordnung oder Teilzuordnung) S

1 $aktuell = S$
2 $ergebnis = undef$
3 Solange Verletzungen($aktuell$) und Zeitlimit nicht erreicht
4 $\quad naechster =$ optimiere (aktuell)
5 $\quad$ Wenn Verletzungsbewertung($naechster$)>Verletzungsbewertung($aktuell$)
6 $\quad\quad ergebnis = aktuell$
7 $\quad aktuell = naechster$

Output: Lösungszustand: $ergebnis$

Algorithmus 5 Hillclimbing im Suchraum *Konstruktionstyp*. Eine Hillclimbing-Suche angepaßt für die Suchraumstruktur des Typs *Konstruktionstyp*. „Vorschlagen und Vertauschen“ erweitert diese Hillclimbingsuche durch Optimierungen/Korrekturen in jedem Schritt und eine anschließende globale Optimierungsphase.

Input: Startzustand S (kein Nachfrager ist zugeteilt)

1 $aktuell = S$
2 $ergebnis = undef$
3 Solange $ergebnis$ undefiniert
4 $\quad naechster =$ bester Nachfolger von $aktuell$
5 $\quad$ Wenn alle Nachfrager zugeteilt
6 $\quad\quad ergebnis = naechster$
7 $\quad aktuell = naechster$

Output: Lösungszustand: $ergebnis$

Suchstrategie zur Lösung von Zuordnungsproblemen oder zur Suche nach lokalen Verbesserungen.

In Kombination mit einem anderen Suchverfahren wird es in V&V sehr effektiv. Hillclimbing ist fester Bestandteil des Algorithmus. Die Grundstruktur entspricht einer Hillclimbing-Suche und auch bei der globalen Optimierung bietet sich Hillclimbing an. In beiden Fällen ist jedoch ein eingebettetes Suchverfahren zur lokalen Suche nach Verbesserungen notwendig.

Die Funktion der lokalen Optimierung kann man sich auf folgende zwei Weisen vorstellen:

- Die Struktur des Suchraums wird verbessert. Durch die lokale Optimierung werden die Nachbarschaften für das Hillclimbing erweitert. Es führen mehr Pfade zur Lösung (Das ist der überwiegende Aspekt beim Konstruktionstyp bzw. der Algorithmus-Grundstruktur).

- Lokale Minima werden durch die lokale Suche nach Verbesserungen überwunden. Das eingebettete Suchverfahren betrachtet nur eine eingeschränkte Problemstellung, ist hier aber fähig, lokale Minima zu überwinden(überwiegender Aspekt beim Zustandstyp bzw. der globalen Optimierung).

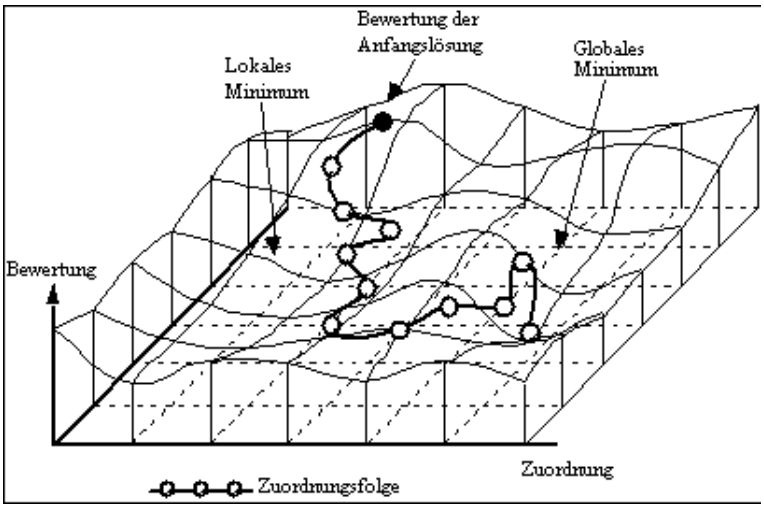

Abbildung 4.7: Schematische Darstellung einer Suche mit Simulated Annealing. Die Suche landet zuerst in einem lokalen Maximum, findet dann aber, da auch Verschlechterungen akzeptiert werden, dennoch zum globalen Maximum. (Quelle des Bildes: [Klügl 1994])

Im Praxisteil dieser Arbeit wurde Hillclimbing als alternatives Suchverfahren bei der lokalen Suche implementiert. In Abschnitt 8.2.8 wird es im Vergleich zu Tiefensuche und Bestensuche evaluiert.

4.4 Simulated Annealing

Simulated Annealing, zu deutsch Simuliertes Ausglühen, ist eine Erweiterung des Hillclimbing-Algorithmus. Es versucht, das Problem der lokalen Maxima dadurch zu umgehen, daß bei der Suche mit einer bestimmten Wahrscheinlichkeit auch eine schlechtere Bewertung akzeptiert wird. Diese Wahrscheinlichkeit sinkt im Laufe der Suche und das Verhalten gleicht sich dann dem reinen Hillclimbing an. Ausführlicher ist dies in [Russell & Norvig 1995] beschrieben.

4.4.1 Reines Simulated Annealing

Der Einsatz von Simulated Annealing bei der Lösung von Zuordnungsproblemen wurde bereits von [Klügl 1994] untersucht. Dabei wurde mit „Vorschlagen und Vertauschen“ oder auf zufälligem Wege eine Lösung erzeugt. Die globale Optimierung wurde dann mit Simulated Annealing und verschiedenen Mutationsstrategien[7] durchgeführt.

Die untersuchte Suche entspricht einem reinen Simulated Annealing, denn V&V wurde hier nur zur Generierung eines Ausgangszustandes für die Simulated-Annealing-Suche verwendet.

Bei relativ einfachen Zuordnungsproblemen kam Simulated Annealing zu sehr guten Ergebnissen. Zur Anwendung auf komplexere Probleme meint [Klügl 1994]:

[7] Strategien zur Generierung und Auswahl eines Nachfolgezustands (siehe Abschnitt 3.4.1)

„Es zeigte sich, daß bei einem so komplexen Zuordnungsproblem auch bei massivem Einsatz von bewertungsgesteuerten Mutationsstrategien während einer Optimierung eine Zuordnung erreicht werden kann, von der aus keine weiteren Verbesserungen mehr möglich sind.“

„Bei diesen Problemen haben wir die Grenze der Leistung des angepaßten Simulierten Ausglühens erreicht.“

Die Autorin weist darauf hin, daß dennoch gute Ergebnisse für den Einsatz in der Vorschlagen-Phase des „Vorschlagen und Vertauschen“-Algorithmus zu erwarten sind.

Auch die *Iterative Repair-Methode*, die für Zuordnungsprobleme eingesetzt werden kann (siehe [Poeck 1995]), benutzt Simuliertes Ausglühen in einer verbesserungsbasierten Suche. Dabei wird zunächst eine Ausgangslösung erzeugt und dann werden wiederholt einstufige Reperaturen durchgeführt.

4.4.2 Simulated Annealing in V&V

Aufgrund der Ähnlichkeit zu Hillclimbing sind die Einsatzmöglichkeiten für Simulated Annealing entsprechend dieselben wie beim Hillclimbing. Es werden deshalb nur die Unterschiede erläutert:

Simulated Annealing bei der inkrementellen Suche
Simulated Annealing bei der inkrementellen Suche der V&V-Grundstruktur einzusetzen, ist wenig sinnvoll. Einen anderen als den besten Nachfolgezustand auszusuchen, widerstrebt der Strategie, die Lösung schrittweise zu erzeugen und dabei Verletzungen möglichst von Beginn an zu vermeiden.

Das ist intuitiv klar, soll aber auch ausführlich begründet werden: Der Vorteil von Simulated Annealing liegt in der Möglichkeit zur Überwindung lokaler Minima in frühen Suchphasen. Bei der Lösungskonstruktion gibt es keine lokalen Minima, man nähert sich dem Lösungszustand immer, die Nachfolgezustände unterscheiden sich nur in einem mehr oder weniger steilem Anstieg. Es bleibt also in diesem Bereich einfaches Hillclimbing die erfolgversprechenste Methode.

Simulated Annealing bei der lokalen Suche
Wie Hillclimbing kann auch Simulated Annealing in „reiner“ Form bei der lokalen Optimierung eingesetzt werden. Für Hillclimbing zeigt die Evaluation (Abschnitt 8.2.8), daß es sich nicht sehr gut für die lokale Optimierung eignet. Ob Simulated Annealing hier erfolgreicher ist als Hillclimbing, bleibt eine schwer abzuschätzende Frage. Aufgrund der Eigenschaften des Simulated Annealing kann man erwarten, daß man durchschnittlich bessere Ergebnisse bekommt, jedoch mehr Zeit zur Lösungsfindung benötigt wird.
Vorstellbar ist, daß sich, ähnlich wie bei der reinen Anwendung des Simulated Annealing auf Zuordnungsprobleme [Klügl 1994], eine starke Abhängigkeit von der Komplexität der Probleme zeigt.

Simulated Annealing bei der globalen Suche
Die Grenzen für den Einsatz von **reinem** Simulated Annealing Suchverfahren bei der globalen Optimierung hat [Klügl 1994] bereits gezeigt.

Es bleibt zu untersuchen, wie - analog zum Hillclimbing - auch Simulated Annealing in Kombination mit lokaler Optimierung eingesetzt werden kann. In der

gegenwärtigen Implementation wird Hillclimbing über eine eingebettete Suche, die eine Korrektur ermittelt, durchgeführt. Verwendet man Simulated Annealing an dieser Stelle, muß man folgende Dinge beachten:

- Die lokale Optimierung wählt den Nachfolgezustand nicht zufallsgesteuert sondern die beste Korrektur aus. Dies erfordert eine kleine Abwandlung des Simulated Annealing. Üblicherweise wählt man einen Nachfolgezustand aus und akzeptiert ihn, wenn er besser als der aktuelle ist, und mit einer bestimmten Wahrscheinlichkeit, auch wenn er schlechter ist.

  Die lokale Optimierung erzeugt keinen Nachfolgezustand, der schlechter als der gegenwärtige ist. Die vorgeschlagene Änderung ist deshalb, die Akzeptierungswahrscheinlichkeit *vor* der Nachfolgergenerierung zu berechnen. Würde man einen schlechten Nachfolgezustand nicht akzeptieren, ermittelt man ihn mit der lokalen Optimierung, anderenfalls erzeugt man auf einem alternativen Weg einen zufälligen Nachfolgezustand. Man benötigt also zusätzlich noch eine Strategie, wie man zufällig Nachbarn erzeugen kann.

- Beachten muß man weiter, daß das Simulated Annealing nicht lokal zu Verschlechterung (bei den früh behandelten Zuteilungen) und lediglich an anderer Stelle (bei den später behandelten Zuteilungen) zu Verbesserungen führt. Insbesondere darf man Verletzungen, die durch das Zulassen einer Verschlechterung entstanden, nicht vergessen, sondern muß sie später bei niedriger Annealingtemperatur korrigieren.

4.4.3 Fazit

Simulated Annealing kann an einigen Stellen (lokale und globale Optimierung) als Alternative zum einfachen Hillclimbing verwendet werden. COKE verwendet sowohl im bestehenden System als auch bei den COKE-Extensions lediglich Hillclimbing.
Es muß noch untersucht werden, ob Simulated Annealing in den beschriebenen Varianten Vorteile bringt. Gerade die Alternative, Simulated Annealing mit einem eingebettetem Suchverfahren zur globalen Optimierung einzusetzen, eröffnet erfolgversprechende Perspektiven. In dieser Arbeit wurde aus Gründen des Umfangs auf eine Implementation dieser Variante verzichtet.

4.5 Best First Search

Wie Hillclimbing verwendet die Bestensuche, auch genannt Best First Search, heuristisches Wissen in Form einer Bewertungsfunktion und setzt es für eine zielstrebigere Suche ein. Die Bestensuche merkt sich nicht nur den aktuellen Zustand, sondern eine Liste aller gefundenen, aber noch nicht expandierten Zustände. Dadurch ist sie zwar vollständig[8], dies bezahlt sie allerdings mit dem Preis, daß sie im schlimmsten Fall exponentiellen Speicherplatz benötigt.

[8] Sie findet eine Lösung, sofern sie im Suchraum über die Nachbarschaftsdefinitionen erreichbar ist.

Im Vergleich zur Tiefensuche geht die Bestensuche „zielstrebiger“ vor; sie besucht den vermeintlich besten der Nachfolger zuerst. Im Gegensatz zur Tiefensuche benutzt sie also heuristisches Wissen zur Suche. In der Literatur finden sich viele Varianten und Bezeichnungen im Zusammenhang mit der Bestensuche: A, A*, Greedy-Search, Uniform-Cost-Search. Im folgenden werden diese Begriffe zunächst voneinander abgegrenzt.

4.5.1 Einführung und Abgrenzung

Die Bestensuche, auch A-Suche genannt, expandiert immer den Zustandsknoten im schon erforschten Suchbaum, von dem sie glaubt, daß er einem Zielzustand am nächsten ist. Schon besuchte, jedoch nicht expandierte Zustände werden zum späteren Vergleich in einer Liste gespeichert. Zum Bestimmen des besten Knotens benötigt die Bestensuche eine Schätzfunktion. Die in Algorithmus 6 zeigt eine Variante der Bestensuche, deren Zielzustand ein Minimum der Bewertungsfunktion ist.

Algorithmus 6 Best-First-Search mit Zeitbeschränkung

Input: Startzustand S Bewertungsfunktion f

1 *offen* $= S$
2 $ergebnis = S$
3 Solange *offen* nicht leer und Zeitlimit nicht erreicht
4 $naechster$ = erstes Element von *offen*
5 Für alle Nachfolger $nachfolger$ von $naechster$
6 Wenn $f(nachfolger) < f(ergebnis)$
7 $ergebnis = nachfolger$
8 Sortiere $nachfolger$ mit f aufsteigend in $offen$ ein.

Output: Lösungszustand: $ergebnis$

Die Bewertungsfunktion f wird gelegentlich in zwei Teile zerlegt: $f = g + h$, wobei g die bisher entstandenen Kosten auf dem Pfad und h eine Abschätzung der Kosten bis zum Ziel bedeutet. h nennt man dabei auch den *Restschätzer*. Setzt man $g \equiv 0$ erhält man *Greedy-Search* [Russell & Norvig 1995]. Greedy-Search versucht die geschätzten Kosten zum Ziel zu minimieren. Setzt man g gleich den entstandenen Kosten auf einem Pfad und $h \equiv 0$, handelt es sich um eine *Uniform-Cost-Search* [Russell & Norvig 1995]. Diese hat das Ziel, die Kosten eines Pfades zur Lösung zu reduzieren. Eine Uniform-Cost-Search, bei der g gleich der Rekursionstiefe ist, entspricht der *Breitensuche*. Hat h die Eigenschaft, daß sie die tatsächlich noch entstehenden Kosten immer unterschätzt oder richtig schätzt, handelt es sich um eine *A*-Suche*. Diese Eigenschaft ist sehr vorteilhaft, da die A*-Suche sehr effizient ist und dabei noch die beste Lösung findet:

> „A* is optimally efficient for any given heuristic function. That is, no other optimal algorithm is guaranteed to expand fewer nodes than A*.“ [Russell & Norvig 1995]

Algorithmus	Heuristik	Minimierungsziel
Uniform-Cost	g $(h \equiv 0)$	Pfadkosten bisher
Greedy	h $(g \equiv 0)$	Geschätzte Kosten zum Ziel
A	$g + h$	Geschätzte Gesamtkosten
A*	$g + h$	Unterschätzte Gesamtkosten

Tabelle 4.3: Eine Übersicht über die Unterschiede der Varianten der Bestensuche.

4.5.2 Reine Bestensuche

Eine reine Bestensuche im Konstruktionssuchraum für Zuordnungsprobleme ist sehr ähnlich der Tiefensuche (beschrieben in Abschnitt 4.2.1). Bewertet werden jedoch nicht nur Blattzustände, sondern auch innere Knoten. Die Bewertung liefert die Expansionsreihenfolge aller bereits besuchten Zustände. Expandiert wird jeweils der innere Knoten mit der besten Bewertung, bis entweder ein Blattzustand ohne Verletzung gefunden ist, oder kein Knoten mehr expandiert werden kann. Der beste gefundene Blattzustand wird jeweils gemerkt für den Fall, daß keine verletzungsfreie Zuordnung existiert.

Im folgenden werden die Bestandteile der Bewertungsfunktion für Bestensuche anhand der Zuordnungsprobleme erläutert. Restriktionen können im allgemeinen auch so beschaffen sein, daß eine weitere Zuteilung eine Restriktionsverletzung beseitigt oder verringert (vergleiche Abschnitt 3.4). Deshalb ist die Kostenfunktion bzw. Bewertungsfunktion auf einem Pfad nicht unbedingt monoton steigend. Die bisher entstandenen Verletzungen sind also nur eine Schätzung der Kosten, die bis zur vollständigen Zuordnung entstehen können. Für die in Abschnitt 4.5.1 definierte Notation bedeutet dies $g \equiv 0$ und h ist die Verletzungsbewertung der partiellen Zuordnung. Eine Definition eines speziellen „Restkostenschätzers“, der die Verletzungen der noch nicht zugeteilten Nachfrager abschätzt, kann auch als Restriktion mit entsprechendem Verletzungsgewicht erfolgen. Auch wenn sich dieser Restkostenschätzer vom Verständnis von der Verletzung der partiellen Zuordnung unterscheidet, gibt es keine algorithmische Unterscheidung.

Interessant in Bezug auf die Effektivität ist noch zu untersuchen, ob die Bedingungen für A* erfüllt sind. [Poeck 1995] schreibt dazu: „Eine Unterschätzung ist allerdings auch nicht zu erreichen, da die restlichen Zuordnungen[9] im besten Fall keine weiteren Kosten verursachen.“. Wichtiger ist, daß durch die restlichen Zuteilungen bestehende Verletzungen sogar reduziert werden können; eine Unterschätzung ist also nicht gegeben.

Bestensuche findet durch die heuristische Vorgehensweise im Schnitt schneller eine Lösung als die Tiefensuche. Sie ist ebenfalls vollständig und besitzt die selbe worst-case-Laufzeit. Das Speichern der expandierten Knoten benötigt jedoch im Gegensatz zur Tiefensuche wesentlich mehr Platz.

So stellte auch [Poeck 1995] zur Bestensuche im Konstruktionstyp fest:

> „Im schlimmsten Fall kann der Algorithmus dazu führen, daß alle Knoten besucht werden, also ein exponentieller Zeitverbrauch benö-

[9] Zuordnung ist Zuteilung nach der hier gewählten Terminiologie

tigt wird und daß auch exponentiell viele Pfade gespeichert werden müssen, also auch ein exponentieller Platzverbrauch eintritt."

Zusammenfassend kann man feststellen, daß die reine Bestensuche für komplexe Zuordnungsprobleme, wie wir sie betrachten, nicht von praktischer Relevanz ist. Die benötigte Rechenzeit und insbesondere der Speicherplatzbedarf stehen dem entgegen.

4.5.3 Bestensuche in V&V

[Poeck 1995] beschreibt „Vorschlagen und Vertauschen" mit Bestensuche bei der lokalen und globalen Optimierung. Als Alternative nennt er einfaches Hillclimbing. In der Zuordnungsshell COKE wurde die Bestensuche implementiert und hat seine Eignung in den bereits bestehenden Anwendungen bereits gezeigt.

Analog zur Tiefensuche (Abschnitt 4.2.2) eignet sich für die Anwendung der Bestensuche in V&V lediglich die lokale Optimierung. Hier ist wieder das Optimierungsziel eingeschränkt und deshalb der zu erforschende Suchraum kleiner als bei der direkten Anwendung der Bestensuche auf das Gesamtproblem.

Es gibt ebenfalls wie bei der Tiefensuche folgende Probleme und dazugehörige Lösungsmöglichkeiten:

- Zyklen
  Der Suchraum der lokalen Suche ist vom Zustandstyp oder vom Störenfriedtyp. Er kann Zyklen enthalten und damit ein Terminieren der Bestensuche verhindern. Als Gegenmaßnahmen bieten sich wieder die schon beschriebenen verschiedenen Zyklustests (siehe Abschnitt 4.1.3) oder eine Beschränkung der Suchtiefe an.

- Größe des Suchraumes
  Durch das Abschneiden von Ästen, auf denen keine Verbesserung mehr möglich ist, kann die Größe des durchsuchten Suchraums verkleinert werden (siehe Abschnitt 4.5.4). Die Größe des Suchraumes ist insbesondere auch aufgrund der Speicherkomplexität der Bestensuche problematisch. Mit einer Variante der Bestensuche, dem sogenannten Beam-Search [Norvig 1992] kann man die Speicherkomplexität reduzieren. Beam-Search merkt sich nicht alle offenen[10] Knoten, sondern nur eine begrenzte Anzahl offener Knoten (z.B. k). Es werden dazu jeweils die k Knoten mit der geringsten Verletzungsbewertung gewählt.

Eine Analyse der Bewertungsfunktion zeigt wieder, daß es keine monoton wachsenden Pfadkosten g gibt. Die Verletzungsbewertung gibt die aktuelle Güte eines Zustandes an. Diese kann sich während der Suche auf einem Pfad des Lösungsweges verbessern (z.B. durch Korrekturen) oder verschlechtern (nicht geglückte Korrekturen). Die aktuelle Verletzungsbewertung kann also auch nur als Schätzung (in der verwendeten Notation: h) verwendet werden, wie nahe man der Lösung bereits ist. Diese Distanz ist nicht absolut, da wir die Verletzungsbewertung der optimalen Zuordnung nicht kennen. Bei der lokalen Optimierung

[10] offen heißt „schon besucht und expandiert"

kann man also keine Pfadkosten g definieren und die Schätzung h ist im allgemeinen keine Unterschätzung. Dennoch ist unter bestimmten Voraussetzungen eine A*-Suche möglich. Das ist Inhalt des folgenden Abschnittes.

4.5.4 A* bei der lokalen Optimierung

Wie bereits angedeutet kann man unter einigen Voraussetzungen auch eine A*-Suche bei der lokalen Optimierung erreichen. Zuerst werden nun die notwendigen Voraussetzungen definiert und dann folgt die Argumentation. Da die Kausalkette nicht sehr leicht nachvollziehbar ist, wurde die Begründung in einige Lemmas zerlegt.

Voraussetzung 4.5.1 (Monotone Restriktionen) *Alle bei der lokalen Optimierung verwendeten Restriktionen sind monoton, d.h. eine Verletzung kann nicht durch eine weitere Zuteilung beseitigt oder ihr Gewicht reduziert werden.*

Voraussetzung 4.5.2 (Spezieller Störenfriedtyp) *Das verwendete Suchraummodel bei der lokalen Optimierung ist der Störenfriedtyp mit folgenden Einschränkungen: In einem Zustand ist maximal ein Störenfried zurückgezogen. Ein Nachfolgezustand entsteht entweder durch das Setzen eines Störenfriedes oder durch das Setzen eines Störenfriedes und sofort folgendes Zurückziehen eines ihn störenden Nachfragers. Ausgangspunkt der Suche ist ein Zustand mit einer zurückgezogen Zuteilung.*

Voraussetzung 4.5.3 (Spezielle Veränderungsbewertung) *Beim Zurückziehen wird die Bewertungsveränderung am auslösenden Nachfrager, nicht am zurückgezogenen Nachfrager gemessen.*

Lemma 4.5.4 (Folge für Veränderungsbewertung) *Aus Voraussetzung 4.5.1 folgt, daß Verletzungsbewertungen durch das Zurückziehen von Nachfragern nur reduziert werden können. Die Bewertungsveränderung am auslösenden Nachfrager beim Zurückziehen ist also eine Verbesserung.*

Lemma 4.5.5 (Schlechtere Nachfolger) *Ein Nachfolger eines gestörten Zustands ist schlechter bewertet als sein Vorgänger.*
Aufgrund von Voraussetzung 4.5.2 gibt es folgende zwei Fälle:

- *ein Nachfrager wird gesetzt*
  $\Rightarrow$ *die Bewertung wird schlechter, da alle Restriktionen monoton sind.*
- *ein Nachfrager wird gesetzt - ein Störenfried zurückgezogen*
  $\Rightarrow$ *die Bewertung wird schlechter, da das Zurückziehen des Störenfrieden nur Verletzungen am vorher gesetzten Nachfrager reduziert (Lemma 4.5.4). Die Bewertungsänderung ist also positiv.*

Insgesamt ist also der Nachfolger eines gestörten Zustands immer schlechter als sein Vorgänger.

Lemma 4.5.6 (Monoton steigende Bewertungswerte) *Auf einem Pfad vom Startzustand zu einem Blattzustand steigen die Werte der Bewertungsfunktion monoton an.*

Dies folgt aus Lemma 4.5.5.

Satz 4.5.7 (Abschneiden von Ästen) *Der Zweig nach einem Knoten, der schlechter ist als ein bisher gefundener Blattzustand oder der Startzustand, kann abgeschnitten werden.*

Da die Bewertung auf einem Pfad nur schlechter werden kann (Lemma 4.5.6) sind mögliche Lösungen hinter dem betrachteten Zustandsknoten schlechter als der bereits gefundene.

Satz 4.5.8 (A*-Suche) *Unter den gegebenen Voraussetzungen (Voraussetzung 1, 2 und 3) ist die Bestensuche eine A*-Suche.*

Da die Bewertung auf einem Pfad nie abnehmen kann (Folge aus Lemma 4.5.6), ist die gegenwärtige Zustandsbewertung stets eine Unterschätzung der Verletzungsbewertung eines Blattzustandes. Aus diesem Grunde handelt es sich bei der Bestensuche unter diesen Voraussetzungen (Voraussetzung 1 2 und 3) um eine A-Suche.*

Die Zuordnungsshell COKE in ihrer bisherigen Implementierung der lokalen Optimierung unterstützt Voraussetzung 4.5.2 und Voraussetzung 4.5.3. Sie realisiert eine Bestensuche und, falls die Wissenskomponente nur monotone Restriktionen enthält, sogar eine A*-Suche. Sie schneidet unvollständige schlecht bewertete Äste bereits ab. Beim Vorhandensein nichtmonotoner Restriktionen ist dieses Abschneiden der Äste nicht korrekt. In den COKE-Extensions wird ein Abschalten des „Pruning“ (= Abscheiden der Äste) ermöglicht. Kapitel 8.2.7 evaluiert die Unterschiede.

4.5.5 Fazit

Best-First-Search ist vollständig und geht zielgerichtet vor. Sie ist im Bereich der lokalen und globalen Optimierung sehr erfolgreich. Trotz des theoretisch möglichen exponentiellen Speicherbedarfs der Suche, traten bei der Verwendung der Bestensuche im praktischen Einsatz bisher keine Speicherprobleme auf. Es läßt sich deshalb vermuten, daß der Verzweigungsfaktor der Suche, der durch den stark wissensbasierten Ansatz relativ gering gehalten wird, klein genug ist, so daß die Rechenzeit vor dem Speicherbedarf zum begrenzenden Faktor wird. Im Bereich der globalen Optimierung hat sich eine Mischform aus Hillclimbing und Bestensuche bereits bewährt. Diese wurde in Kapitel 3.6.2 beschrieben.

Innerhalb der lokalen Optimierung gibt es für die Bestensuche zwei interessante Varianten:

- A* mit Pruning
- Bestensuche ohne Pruning

Ob eine A^*-Suche möglich ist und ob man schlecht bewertete Teiläste des Suchbaumes von der Suche ausschließen kann, hängt von der Art der Implementation und den Restriktionen der Probleme ab. Viele Probleme besitzen nichtmonotone Restriktionen, die vor allem bei der globalen Optimierung beachtet werden sollen. Ein Beispiel dafür ist eine Restriktion, die eine geforderte Mindestwochenarbeitszeit in der Personaleinsatzplanung überprüft. Sie ist nichtmonoton,

da eine Verletzung durch weitere Zuteilungen (damit einer Erhöhung der Wochenarbeitszeit) verbessert werden kann. Eine solche Restriktion wird man typischerweise erst bei der globalen Optimierung beachten. Dann ist es von Vorteil, bei der lokalen Optimierung die effiziente A*-Suche mit Pruning zu verwenden, bei der globalen Optimierung die notwendige Bestensuche ohne Pruning.

4.6 Andere Suchverfahren

Die untersuchten Suchverfahren stellen eine Auswahl aus den klassischen Suchverfahren der KI dar. Um den Umfang dieser Arbeit nicht zu sprengen, wurden nur diese ausführlich untersucht. Verzichtet wurde auf die Analyse der Suchverfahren, die aufgrund fehlender Voraussetzungen nicht für Zuordnungsprobleme verwendet werden können, so z.B. auch die *Bidirektionale Suche*, bei der ein Lösungspfad ausgehend von Startzustand vorwärts und Zielzustand rückwärts gesucht wird. Bei der Zuordnung ist ein Zielzustand nicht bekannt. Ebenfalls verzichtet wurde auf eine Untersuchung von Suchverfahren, bei denen keine wesentlichen Erkenntnisse zu erwarten waren. Dazu gehört z.B. die *Breitensuche*, die im Zweifelsfall schlechter ist als die Bestensuche. Die Bewertungsfunktion, die als heuristische Funktion dienlich ist, senkt die average-case-Laufzeit und den average-case-Speicherbedarf.

[Poeck 1995] beschreibt neben „Vorschlagen und Vertauschen“ auch noch weitere reparaturbasierte bzw. verbesserungsbasierte Verfahren für Zuordnungsprobleme darunter die *Min conflicts-Methode*, das *Iterative Repair-Verfahren* und den *GSAT-Algorithmus*. Diese bestehen wie V&V aus einer Hillclimbing-Komponente und einem verletzungsabhängigen Reparatursystem. Es bestehen starke Ähnlichkeiten untereinander und zu V&V. Grundsätzlich sind in „Vorschlagen und Vertauschen“ schon die wesentlichen Elemente enthalten, abgesehen vom „Simulierten Ausglühen“, das in Kapitel 4.4 bereits erwähnt wurde.

Kapitel 5

Sonstige Verbesserungen

In diesem Kapitel werden einige Verbesserungsideen für das „Vorschlagen und Vertauschen“ behandelt, die nicht in die Struktur der letzten beiden Kapitel gepaßt haben bzw. aufgrund ihres Umfangs ausgegliedert wurden.

5.1 Plausibilitätsprüfung

Manchmal ist bei der Aufgabenstellung noch nicht bekannt, ob sie überhaupt lösbar ist. Wenn ein System, z.B. ein V&V-System, keine brauchbare Lösung findet, ist es schwierig, zu erkennen, ob dies an der Aufgabenstellung oder an einer nicht geglückten Planung liegt. Im Beispiel der Pflegepersonalplanung ist es wünschenswert zu wissen, warum kein verletzungsfreier Dienstplan erstellt werden kann. Ein Grund kann sein, daß nicht genügend Personal für die Menge an Schichten zur Verfügung steht. In diesem Falle wurde also nicht schlecht geplant, sondern es wurden ungeeignete Eingangsparameter gewählt. Eine *Plausibilitätsprüfung* soll helfen, festzustellen, ob eine Problemstellung überhaupt lösbar ist und - eventuell noch - an welchen Rahmenbedingungen die Optimierung scheitert.

Die Idee ist zunächst zu prüfen, ob das Problem lösbar ist, wenn nur Konsistenzbedingungen eingehalten werden müssen und Optimierungsrestriktionen zunächst außer Acht gelassen werden. Ist dieses einfachere Problem schon nicht lösbar, kann man davon ausgehen, daß die Problemstellung nicht plausibel ist. Wird eine Lösung gefunden, kann man nun schrittweise Prüfläufe mit steigender Anzahl von Restriktionen durchführen. So erkennt man auch, ab welcher Restriktion oder ab welcher Restriktionsgruppe keine Lösung mehr gefunden wird. Für diese Restriktionen bzw. deren Korrekturen lohnt dann vielleicht ein erneuter Wissenserwerb, z.B. eine Verbesserung der Korrektur- und Störenfriedfunktionen.

Zur Umsetzung kann ein allgemeinerer Ansatz gewählt werden. Notwendig ist dazu eine Erweiterung, die V&V-Batchläufe mit verschiedenen Profilen durchführt und die Ergebnisse protokolliert. In den verschiedenen Profilen kann man beliebige Teilmengen von Restriktionen aktiv setzen. Wenn in den Profilen auch die Verletzungsgewichte verändert werden können, eignet sich das Batch-Werkzeug auch für das Prüfen verschiedener Restriktionsgewichtungen und ihren Auswirkungen. Für einen Standard-Plausibilitätstest können automatisch

	Verletzungsstufen der aktiven Restriktionen
Profil 1	Konsistenzverletzung
Profil 2	Konsistenzverletzung, schwere Verletzungen
Profil 3	Konsistenzverletzung, schwere und mittelschwere Verletzungen
Profil 4	Konsistenzverletzung, schwer, mittel und leicht
Profil 5	Alle Verletzungsstufen

Tabelle 5.1: Standardprofile für die Plausibilitätsprüfung. Die Profile werden im Batchlauf getestet.

Profile aus den Verletzungsgewichtsklassen generiert werden. Tabelle 5.1 zeigt die Standardprofile aus den COKE-Extensions.

Wie nützlich der allgemeine Ansatz ist, zeigt auch folgendes Beispiel. Angenommen man hat Konsistenzbedingungen mit sehr hohem Gewicht, Optimierungskriterien mit mittlerem Gewicht und zusätzlich noch Restriktionen mit rein strategischem Wert[1]. In diesem Falle ist es besser zum Prüfen der Plausibilität zuerst die Optimierungskriterien auszublenden, die strategischen Restriktionen jedoch noch aktiv zu lassen. Ein reines Vorgehen nach dem Verletzungsgewicht, würde zuerst die strategischen Hilfen ausblenden und erst später die Optimierungskriterien. Das Problem würde, statt leichter, durch die fehlenden Strategiehinweise schwerer werden.

Da es sich bei Zuordnungsproblemen um Optimierungsprobleme handelt, ist es für den allgemeinen Fall nicht genau definiert, wann eine Lösung gefunden wurde. „Vorschlagen und Vertauschen“ findet immer Lösung nämlich eine vollständige Zuordnung. Offen ist nur wieviele und wie stark gewichtete Verletzungen bestehen. Zu einem konkreten Problem gibt es jedoch meist sehr konkrete Anforderungen an eine „brauchbare“ Lösung. Im Falle der Dienstplanung wird dies meist das Einhalten der gesetzlichen Vorschriften oder logischer Notwendigkeiten[2] sein.

Deshalb muß man für eine Plausibilitätsprüfung auch bestimmen, ab wann eine Lösung akzeptiert wird. Dafür bieten sich folgende Kriterien an: Zum einen kann eine maximale Verletzungsklasse angegeben werden. Einzelverletzungen bis zu diesem Gewicht dürfen dann in einer Lösungszuordnung vorkommen. Zum anderen kann man auch eine maximale Verletzungssumme für die gesamte Zuordnung angeben. Ein kleines Problem dabei ist, daß man die maximale Verletzungssumme nicht unbedingt als Güteskala verwenden kann. Die Verletzungssumme ist abhängig von der Menge der Zuteilungen und deswegen kein absolutes Qualitätsmaß für eine Zuordnung. Ein Beispiel: In der Pflegepersonalplanung ist eine Verletzungssumme von 400 für einen Planungszeitraum von 4 Wochen weniger fatal als eine Verletzungssumme der Höhe 300 für 2 Wochen. Als dritte Möglichkeit bietet sich daher an, die durchschnittliche Verletzungsgewicht pro Zuteilung zu beschränken. Zusammengefaßt kann man also folgende Kriterien zur Akzeptierung einer Lösung angeben:

[1] In Abschnitt 3.2.3 wurde der Einsatz von solchen Restriktionen angedeutet, mit denen man die Entscheidung für den am wenigsten einschränkenden Anbieter forcieren kann.

[2] Die Bedingung, daß eine Dienstagsschicht nur an einem Dienstag eingetragen werden kann ist z.B. eine logische Notwendigkeit.

- Maximale Verletzungsklasse
- Maximales Verletzungsgewicht
- Maximales durchschnittliches Verletzungsgewicht pro Zuteilung

Eine Plausibilitätsprüfung für „Vorschlagen und Vertauschen“ ist keine algorithmische Verbesserung in dem Sinne, daß sie eine Verkürzung der Laufzeit oder eine Verbesserung der Ergebnisse mit sich bringt, sie kann jedoch den Einsatz in der Praxis erleichtern, da leichter festgestellt werden kann, wo die Schwierigkeiten bei der Problemlösung liegen. Hat man die Schwierigkeiten erst erkannt, kann der Benutzer eingreifen, sei es durch Veränderung der Problemstellung, durch Relaxierung einiger Restriktionen oder sogar durch eine Verbesserung der Wissenskomponente. Der Effizienzgewinn liegt also bei der Benutzbarkeit bzw. der Interaktion.

Das eben vorgestellte Werkzeug für Versuchsläufe mit verschiedenen Profilen wurde implementiert und wird in Abschnitt 8.5 dazu benutzt, festzustellen, ob man damit sinnvoll die Plausibilität einer Aufgabenstellung beurteilen kann. Es stellte sich heraus, daß dies unter bestimmten Voraussetzungen zwar möglich ist, das Ergebnis des Testes jedoch immer einer Interpretation bedarf.

5.2 Laufzeitbegrenzung

In Abschnitt 2.3.2 wurde bereits kurz die Anytime-Eigenschaft des V&V-Algorithmus erläutert. Dem Anwender stellt sich die Frage des Trade-Offs zwischen der Güte der Lösung und der benötigten Zeit. In diesem Kapitel werden verschiedene Verfahrensweisen dargestellt, die dem Anwender ein Begrenzen der Laufzeit ermöglichen.

Dazu wird zunächst der Begriff „Anytime“ kritisch betrachtet. Anschließend werden verschiedene Modelle zur Begrenzung der Laufzeit vorgestellt und ihre Eigenschaften untersucht. Ziel ist es, die Anytime-Eigenschaft möglichst gut und effektiv zu realisieren und dem Anwender eine gute Kontrolle über das Laufzeitverhalten des Algorithmus zu geben.

5.2.1 Definition Anytime-Algorithmus

[Russell & Zilberstein 1991] definieren den Begriff *Anytime-Algorithmus* auf folgende Weise:

> „Anytime algorithms are algorithms whose quality of results degrades gracefully as computation time decreases, hence they introduce a tradeoff between computation time and quality of results.“

Die Autoren unterscheiden weiterhin zwischen *interruptible algorithms* und *contract algorithms*. Erstere liefern Lösungen der beschriebenen Qualität, sogar wenn sie unerwartet unterbrochen werden. Die zweite Klasse von Algorithmen kann ebenfalls Lösungen liefern, deren Qualität in Relation mit der zur Verfügung gestellten Zeit steht. Allerdings muß das Zeitkontingent im voraus bestimmt sein. Wenn ein contract-Algorithmus früher als in der vereinbarten Zeit abbricht, findet er kein gültiges Ergebnis [Russell & Zilberstein 1991].

5.2.2 Anytime-Eigenschaft in „Vorschlagen und Vertauschen“

Da V&V kein einfaches Suchverfahren ist, sondern aus mehreren Suchen zusammengesetzt, ist es notwendig, die Eigenschaften des Algorithmus in Bezug auf die Anytime-Eigenschaft differenziert zu untersuchen.

Die lokale und globale Optimierung besitzen die Anytime-Eigenschaft in beiderlei Hinsicht: Sie sind zu jedem Zeitpunkt unterbrechbar und liefern dabei eine Lösung[3]. Außerdem variiert die Qualität des Ergebnisses mit der vorher vereinbarten Optimierungszeit.
Die Konstruktion der Lösung ist ein contract-Algorithmus. Die vorher festgelegte Lösungszeit kann auf die Zuteilungen der Nachfrager verteilt werden. Allerdings ist die Lösungskonstruktion nicht unterbrechbar. Eine partielle Zuordnung wäre als Ergebnis im allgemeinen nicht nützlich.

Die Qualität der Lösung steht zwar mit der verwendeten Zeit in Relation, sie steigt jedoch nicht beliebig mit immer größeren Zeitkontingenten. Die Wissenskomponente und ihr Einsatz werden in den meisten Fällen dazu führen, daß auch bei unendlicher Zeit nur ein bestimmter Bereich des Suchraumes durchsucht wird. Es zeigte sich in Versuchen mit den COKE-Anwendungen WIZARD, PEPSI und dem Nurse Scheduler, daß die lokale Optimierung häufig auch terminiert, ohne eine Lösung gefunden oder die Zeitschranke überschritten zu haben. In diesem Falle wurden bei der Suche keine Nachfolgerzustände mehr erzeugt. Dies kann daran liegen, daß von der Wissenskomponente zu wenige Korrekturen initiiert werden oder daß von Seiten des Algorithmus zu wenige Nachfolgerzustände erzeugt werden (z.B. weil sie durch effiziente aber nicht korrekte Zyklustests abgeschnitten werden). Bei einer drastischen Erhöhung des Zeitkontingentes ist es also auch notwendig, großzügiger mit Vorschlägen und Störenfrieden zu sein bzw. mehr Nachfrager als störend anzugeben, damit ausreichend Korrekturen erzeugt werden.

Die *contract-Eigenschaft* des Algorithmus bzw. der Optimierung wirft weiterhin folgende Frage auf: Wieviel Zeit muß man investieren, um eine gute Lösung zu erhalten? Zwei Faktoren spielen hier eine wichtige Rolle: Der erste sind Zeitvorgaben durch den Benutzer. In den verschiedenen Domänen werden hier verschiedene Anforderungen gestellt. In einigen Fällen ist die Anforderung gegeben, daß eine Lösung kurzfristig erstellt werden muß, in anderen kann man Stunden oder gar Tage auf die Lösung warten. Der zweite Faktor für den Zeitbedarf ist die Schwierigkeit des Problems. Einfache Zuordnungsprobleme mit wenigen Nachfragern oder wenigen Restriktionen sind schneller zu lösen als komplexere Zuordnungsprobleme mit vielen Nachfragern oder entsprechend komplexeren Randbedingungssystem. Zum Beispiel hat man bei der Einsatzplanung für Zugbegleitpersonal üblicherweise Nachfragermengen bis zu 50 Nachfragern (siehe [Herrler 1999]), bei der Produktionsplanung dagegen mehrerer tausend Nachfrager (siehe [Busch 1997]). Die Zeit, die man dem System zur Lösung zur Verfügung stellen soll, ist also auch problemabhängig.

[3] Lösung bedeutet hier nicht Freiheit von Konsistenzverletzung. Für V&V ist eine Verletzung bei dieser Betrachtungsweise nur Optimierungskriterium und nicht Kriterium für die Gültigkeit einer Lösung.

5.2.3 Einfache Modelle zur Zeitkontingentierung

Im folgenden werden drei einfache Modelle vorgestellt, wie die Optimierungszeit angegeben und verwendet werden kann. Nach der Contract-Eigenschaft nenne ich sie *Contract-Modelle* oder *Modelle zur Zeitkontingentierung*:

- **Zeit pro Zuordnung**
  Die Zeit für den ganzen Zuordnungsprozeß kann vom Benutzer begrenzt werden. Diese zur Verfügung stehende Zeit wird auf die einzelnen Zuteilungen aufgeteilt. Falls eine Zuteilung und ihre lokale Optimierung schneller abläuft als ihre Zeitschranke, so terminiert der Algorithmus früher. Die Zeitschranke ist also nur eine Obergrenze.

- **Zeitangabe pro Zuteilung**
  Im vorherigen Modell haben Zuordnungsprobleme mit vielen Nachfragern (das bedeutet auch: mit vielen Zuteilungen) weniger Zeit für jede einzelne lokale Optimierung. Alternativ kann man deshalb auch die Zeit pro Zuteilung angeben und beschränken. Jede verletzte Zuteilung wird maximal diese Zeitspanne lang korrigiert. Da die Anzahl der verletzen Zuteilungen und die Dauer der Korrekturen nicht bekannt ist, ist ein Abschätzen der gesamten Laufzeit auf diese Weise problematisch. Es kann wiederum lediglich eine Obergrenze bestimmt werden.

- **Zeitangabe pro Verletzungsgewichtseinheit**
  Da eine leichte Verletzung weniger ausmacht als eine schwere Verletzung, bietet sich auch folgendes Modell an: Die maximale Korrekturzeit pro Zuteilung wird je nach Verletzungsgewicht bestimmt. Bei einer schwereren Verletzung darf also länger lokal optimiert werden. Da die Anzahl der Verletzungen und ihr Gewicht nicht von vornherein bekannt sind, ist die Rechenzeit bei diesem Modell sehr schwer kalkulierbar.

Diese einfachen Modelle zur Zeitkontingentierung bei der Lösungskonstruktion haben Schwachstellen. Die Zeitbegrenzung pro Zuteilung oder pro Zuordnung nutzt die vom Benutzer zur Verfügung gestellte Zeit in vielen Fällen nicht aus. Nötige Optimierungszeit wird damit „verschenkt“ und kann an entscheidenden Stellen fehlen. Die Zeitangabe pro Verletzungsgewichteinheit wendet dagegen gezielt mehr Zeit für schwere Verletzungen auf, das Laufzeitverhalten ist aber vom Benutzer nicht mehr ausreichend steuerbar.

5.2.4 Verbesserte Modelle zur Zeitkontingentierung

Ausgehend von den einfachen Grundmodellen werden in diesem Kapitel zwei verbesserte Modelle konstruiert, die die Nachteile der einfachen ausgleichen können. Zwei Ziele sollen verfolgt werden: Zum einen soll der Benutzer eine Maximalzeit für die gesamte Zuordnung angeben können, zum anderen soll diese Zeit mit heuristischem Wissen so eingesetzt werden, daß mehr Energie (Rechenzeit) in schwierige Zuteilungen gesteckt wird.

Allgemeine Problemstellenheuristik
Wenn der Benutzer die Zeit pro Zuordnung begrenzt hat, ist es sinnvoll, diese Gesamtzeit nicht zu gleichen Teilen zur Optimierung der einzelnen Zuteilungen

zu verwenden. Besser ist es, mehr Zeit auf die *Problemstellen* oder - wie sie vorher genannt wurden - *schwierigen Zuteilungen* zu verwenden. Problemstellen sind Zuteilungen, die verletzt sind und deren Korrektur nicht sehr schnell gefunden werden kann. Häufig bedeutet dies, daß mehrstufige Veränderungen zur Verletzungskorrektur notwendig sind. Die ersten Zuteilungen sind selten Problemstellen, da die zuzuteilenden Nachfrager noch wenig eingeschränkt (vergleiche Abschnitt 3.1) sind. Es stehen viele ungenutzte Angebotsobjekte zur Verfügung und wenige konkurrierende Nachfrager sind zugeteilt. Bei den frühen Zuteilungen ist also der Zeitbedarf zur Korrektur gering, viele sind sogar verletzungsfrei. Je mehr Zuteilungen bereits gemacht sind, desto eingeschränkter ist die Suche. Es gibt mehr Verletzungen und die Korrekturen sind aufwendiger. Die allgemeine Problemstellenheuristik ist also folgende: Problemstellen werden bei späteren Zuteilungen häufiger bzw. die Zuteilungen werden im Laufe der Zuordnung schwieriger.

Wie gleich noch begründet werden wird, kann man eine exakte Funktion für den Zeitbedarf schwer geben. Ich nehme daher vereinfachend einen linear steigenden Zeitbedarf an. Ist t_g die durch den Benutzer vorgegebene maximale Gesamtzeit für die Zuordnung, so führt folgende Rechnung zu den Zeitkontingenten der einzelnen Zuteilungen:

Sei n die Anzahl der Nachfragerobjekte, t_g die Gesamtzeit, die für die Zuordnung aller Nachfragerobjekte verwendet werden darf. t_i für $i \in \{1..n\}$ ist die Zeit, die bei der Zuordnung des i-ten Nachfragerobjektes verwendet wird. Offensichtlich gilt folgendes:

$$\sum_{i=1}^{n} (t_i) = t_g$$
$$\sum_{i=0}^{n} \left(\frac{i}{n} \cdot t_n\right) = t_g$$
$$\sum_{i=0}^{n} \left(\frac{i}{n}\right) = \frac{t_g}{t_n}$$
$$t_n = \frac{t_g \cdot n}{\sum_{i=0}^{n} i}$$

Aus t_n lassen sich auch die restlichen t_i berechnen, da gilt:

$$t_i = \frac{i}{n} \cdot t_n$$
$$t_i = \frac{i \cdot t_g}{\sum_{j=0}^{n} j}$$

Bei einem linear steigenden Zeitbedarf ist dies nun die Berechnungsvorschrift für die Zeit pro Zuteilung.

Die Annahme eines linear anwachsenden Zeitbedarfes ist willkürlich. Ebensogut könnte man einen exponentiell wachsenden Zeitbedarf voraussetzen, da auch der Suchraum abhängig von den Nachfragern exponentiell wächst. Aufgrund der Vielzahl von beeinflußenden Faktoren ist eine geeignete statistische Funktion für den Zeitbedarf schwer zu ermitteln.

Wie bei der einfachen Zeitangabe pro Zuordnung kann die Gesamtzeit nicht überschritten werden. Allerdings ist zu erwarten, daß das Zeitkontingent besser ausgeschöpft wird, da sich die Zeitverteilung an der statistischen Verteilung der Problemstellen orientiert. So steht im Duchschnitt Problemstellen mehr Zeit zur Optimierung zur Verfügung als einfachen Zuteilungen.

Spezielle Problemstellenstatistik durch einen Vorlauf

Um genauer die wirklichen Problemstellen eines konkreten Zuordnungsproblems zu identifizieren und gezielt dort zu optimieren, kann man den Zuordnungsvorgang in zwei Läufe aufteilen. In einem kurzen zeitbeschränkten Vorlauf ermittelt man die Problemstellen, das bedeutet alle Zuteilungen, die verletzt waren und in der kurzen Optimierungszeit nicht korrigiert werden konnten. Das Gewicht der Restverletzung kann als Schwere der Problemstelle verwendet werden. Die gewonnene Information kann in einem zweiten Durchlauf (kurz Zweitlauf) der Zuordnung berücksichtigt werden.

Es gibt viele Möglichkeiten, wie man die Gesamtzeit auf die Problemstellen verteilen kann. Es sei hier zur Anschauung nur eine Variante gezeigt: Stellt der Benutzer dem System also t_g Sekunden zur Verfügung, so könnte man folgende Aufteilung machen:

$$t_v = 0.1 \cdot t_g \text{ (Zeit für den Vorlauf)}$$
$$t_z = 0.9 \cdot t_g \text{ (Zeit für den Zweitlauf)}$$

Es sei n die Anzahl der Nachfrager. t_v ist die Zeit, die für einen zeitbeschränkten Vorlauf verwendet wird. Aus diesem Vorlauf kennt man p_i für alle $i \in \{1..n\}$, wobei p_i die verbliebene Restverletzung bei der Zuteilung des i-ten Nachfragers im Vorlauf ist. Die Zeit t_i, das Zeitkontingent für die i-te Zuordnung sei beispielsweise folgendermaßen definiert:

$$t_i = \frac{i \cdot (t_z \cdot 0.2)}{\sum_{j=0}^{n} j} + \frac{p_i \cdot (t_z \cdot 0.8)}{\sum_{j=0}^{n} p_j} \tag{5.1}$$

Der erste Bruch gibt hierbei die Basisberechnungszeit an. Diese Basisberechnungszeit nimmt linear mit der Anzahl der Zuteilungen zu. Der zweite Bruch liefert die Extrazeit zum Optimieren der Problemstellen. Nicht nur die Gesamtzeit zerfällt also bei der beschriebenen Berechnungsvorschrift in zwei Anteile, auch die Zeit für den Zweitlauf besteht aus zwei Teilen: die Basisberechnungszeit und die zusätzliche Zeit zum Optimieren der Problemstellen.

Diese letzte Vorgehensweise soll also so gut wie möglich die zur Verfügung stehende Zeit zur Optimierung der Zuordnung verwenden. Das bedeutet, die vorgegebene Zeitschranke nicht zu überschreiten, jedoch auch möglichst wenig zu unterschreiten und dabei noch gute Optimierungsergebnisse zu erzielen.

5.2.5 Laufzeitbegrenzung bei der globalen Optimierung

Bei der globalen Optimierung gibt es ähnliche Möglichkeiten zur Zeiteinteilung wie bei der lokalen Optimierung. Es kann die Optimierungszeit pro verletzter Zuteilung beschränkt werden oder die Optimierungszeit für die gesamte globale Optimierung. Im folgenden werden kurz die beschriebenen 5 Modelle zur Zeitkontingentierung bei der Lösungskonstruktion aufgezählt und zu den Modellen gegebenfalls Anmerkungen über die Besonderheiten bei der globalen Optimierung gemacht:

- **Zeit pro Globaler Optimierung**
  Dies entspricht dem Modell „Zeit pro Zuordnung“. Die Gesamtzeit wird zu gleichen Teilen für die Optimierung aller verletzten Zuteilungen eingesetzt.

- **Zeitangabe pro Zuteilung**
  Äquivalent zur Lösungskonstruktion. Jede verletzte Zuteilung wird eine vorgegebene Zeit lang optimiert.

- **Zeitangabe pro Verletzungsgewichtseinheit**
  Äquivalent zur Lösungskonstruktion. Jede verletzte Zuteilung wird eine Zeitspanne abhängig vom Verletzungsgewicht optimiert.

- **Allgemeine Problemstellenheuristik**
  Eine allgemeine Problemstellenheuristik, wie sie für die Lösungskonstruktion beschrieben wurde, gibt es bei der globalen Optimierung nicht. Früh zur Korrektur vermerkte Zuteilungen sind potentiell genauso eingeschränkt wie spätere Zuteilungen.

- **Optimierungszeit nach Verletzungsgewicht**
  Dieses Modell ist eine Abwandlung des Modells „Problemstellenheuristik durch einen Vorlauf“. Im Unterschied dazu kennt man die Problemstellen schon ohne einen Vorlauf. Eine Bewertung der aktuellen Zuordnung bzw. ihrer Verletzungen ist ausreichend. Die vorangegangene Lösungskonstruktion entspricht dem Vorlauf.

5.2.6 Fazit

Das Ziel war es, möglichst gute Ergebnisse bei einer möglichst starken Benutzerkontrolle (in Bezug auf die Laufzeit) zu erreichen. Sehr leicht zu erreichen ist eine Laufzeitgarantie, die die Laufzeit des Algorithmus nach oben abgrenzt; als erheblich schwieriger erweist es sich, **genau** die zur Verfügung gestellte Zeit zu nutzen.

Im Gegensatz zum Laufzeitverhalten, das durch diese vorgeschlagenen Modelle recht genau beschrieben wird, läßt sich das Lösungsverhalten bei ihrer Anwendung schwer voraussagen. Die drei einfachen Modelle wurden im Rahmen des Praxisteiles dieser Arbeit umgesetzt und sie werden in Kapitel 8.4 noch verglichen. Da der Schwerpunkt der Arbeit auf einer Verbesserung der Ergebnisse liegt und eine Umsetzung und Evaluierung der verbesserten Modelle sehr aufwendig wäre, wurde auf die Implementierung dieser Varianten verzichtet. Mit ein wenig Erfahrung in der jeweiligen Problemdomäne kann man jedoch auch mit der Angabe der Optimierungszeit pro verletzter Zuteilung die Gesamtlaufzeit ausreichend gut abschätzen.

COKE in seiner ursprünglichen Implementierung kennt nur die Zeitangabe pro Verletzungsgewichteinheit. Da das Ausmaß und die Anzahl der Verletzungen sehr variieren, ist ein Abschätzen der Optimierungszeit damit sehr schwierig. Das Verletzungsgewicht von Verletzungen ist zudem nur relativ und abhängig von der Beurteilung durch den Wissensingenieur, der die Restriktionen definiert. Mancher beurteilt sehr selten Verletzungen als schwer, und dementsprechend selten kommt diese Bewertung vor. Für die Problemlösung ist dies unerheblich, da nur der Gewichtsvergleich der Verletzungen untereinander eine Rolle spielt; das niedrige Verletzungsgewicht hat allerdings eine kürzere Optimierungszeit zur Folge. Dies ist nicht beabsichtigt und erschwert zusätzlich das Abschätzen der Optimierungszeit.
Bei der globalen Optimierung wird in COKE die Optimierungszeit ebenfalls

verletzungsabhängig bestimmt. Es wird mehr Zeit pro Verletzungsgewicht optimiert, mit dem Ziel, daß Korrekturen gefunden werden, die bei der lokalen Optimierung nicht gefunden wurden.
Insgesamt ist diese Art der Laufzeitkontrolle für den Benutzer sehr undurchschaubar; die Angabe einer Maximalzeit pro Zuteilung ist intuitiver.

Kapitel 6

Implementierungsalternativen

In den vorausgegangenen Kapiteln wurden theoretische Verbesserungsmöglichkeiten, Implementierungsalternativen und generell Ablaufbeschreibungen für ein effizientes „Vorschlagen und Vertauschen“ gegeben. Für einen besseren abschließenden Überblick werden in folgender Aufstellung die Alternativen der einzelnen Schritte noch einmal zusammengefaßt. Die Übersicht besteht aus den teilweise schon früher zur Zusammenfassung gezeigten Hierarchien. Rechts neben den Alternativen verdeutlichen Symbole, ob diese Variante nur theoretisch beschrieben wurde, ob sie im Rahmen dieser Arbeit implementiert wurde oder ob sie schon zuvor in COKE implementiert war.

Die Symbole haben folgende Bedeutung:

- □ Diese Variante ist nur theoretisch beschrieben.
- ⊡ Diese Variante war schon in COKE vor dieser Arbeit enthalten.
- ⊠ Diese Variante wurde im Rahmen dieser Arbeit implementiert und ist Teil der COKE-Extensions.

Die Numerierung der Varianten wird in der Systembeschreibung und der Beschreibung der Konfigurationen bei der Evaluation verwendet.

1 Auswahl eines Nachfragers

- 1.1 Keine Sortierung / Zufällige Auswahl ⊡
- 1.2 Statische Sortierung ⊡
- 1.3 Statische Sortierung mit dynamischen Veränderungen ⊡
- 1.4 Komplette dynamische Sortierung
  - 1.4.1 Angabe einer dynamischen Sortierfunktion ⊡
  - 1.4.2 Exakte Berechnung der Freiheitgrade □
  - 1.4.3 Freiheitsgrade über Heuristik (Aggregation) bestimmen □

2 Auswahl eines Anbieters

- 2.1 Vorschlagsinterpretation
  - 2.1.1 nur Vorschläge verwenden ⊡
  - 2.1.2 Vorschläge ergänzen □

2.2 Auswahlkriterium

2.2.1 Verletzungsbewertung (alle Vorschläge testen) ⊡

2.2.2 Verletzungsbewertung (nicht alle Vorschläge testen) □

2.2.3 Verletzungsbewertung und Freiheitgrade

2.2.3.1 Freiheitsgrade generisch bestimmen □

2.2.3.2 Freiheitsgrade heuristisch bestimmen □

3 Lokale Optimierung

3.1 Nachbarschaften

3.1.1 Elementare Veränderungsart

3.1.1.1 Umsetzen ⊡

3.1.1.2 direktes Vertauschen □

3.1.1.3 mehrstufiges Vertauschen □

3.1.2 Verwendetes Wissenselement

3.1.2.1 kein spezielles Wissenselement ⊠

3.1.2.2 Störenfriede zuteilungsabhängig ⊡

3.1.2.3 Störenfriede verletzungsabhängig ⊠

3.1.3 Behandlung des Wissenselementes

3.1.3.1 alle Störenfriede gemeinsam zurückziehen ⊠

3.1.3.2 Störenfriede einzeln zurückziehen ⊡

3.1.3.3 Störenfriede in Gruppen zurückziehen □

3.1.4 Veränderungsbewertung

3.1.4.1 Gesamtsituation neu bewerten □

3.1.4.2 Veränderung am Störenfrieden ⊠

3.1.4.3 Veränderung am Initiator der Veränderung ⊡

3.1.5 Zyklustest

3.1.5.1 Wiederherstellen aller Zustände □

3.1.5.2 Selben Nachfrager nicht öfter umsetzen ⊡

3.1.5.3 Nachfrager nicht zweimal auf denselben Anbieter setzen ⊠

3.1.5.4 Zyklustest mit Tabuliste der Zustände □

3.2 Suchstrategie

3.2.1 Bestensuche ⊡

3.2.2 Beschränkte Tiefensuche ⊠

3.2.3 Hillclimbing ⊠

3.2.4 Beam-Search □

3.2.5 Simulated-Annealing □

3.3 Verfahren mit schlecht bewerteten Ästen

3.3.1 Pruning von Ästen ⊡

3.3.2 ohne Pruning von Ästen ⊠

4 Globale Optimierung

4.1 Grundstruktur

4.1.1 direkt ohne Hillclimbing □

- 4.1.2 Hillclimbing über einzelne Verletzungen ⊡
- 4.1.3 Hillclimbing über verletzte Zuteilungen ⊠
- 4.2 Suchverfahren
  - 4.2.1 Hillclimbing ⊡
  - 4.2.2 Simulated Annealing □

Folgendes ist eine Liste der sonstigen Erweiterungsideen, die nicht in obigen Struktur integriert sind:

- 5 Laufzeitsteuerung
  - 5.1 Einfache Contract-Modelle
    - 5.1.1 Zeit pro Zuordnung ⊠
    - 5.1.2 Zeit pro Zuteilung ⊠
    - 5.1.3 Zeit pro Verletzungsgewichteinheit ⊡
  - 5.2 Verbesserte Contract-Modelle
    - 5.2.1 Allgemeine Problemstellenheuristik □
    - 5.2.2 Problemstellenheuristik durch Vorlauf bestimmen □
- 6 Plausibilitätsprüfung ⊠

Kapitel 7

Systembeschreibung

7.1 Was wurde implementiert

Um die eben gekennzeichneten Varianten in COKE einzubauen, war es notwendig, einen Teil des COKE-Kernels, insbesondere das Zustandssystem und die Korrekturfunktionen zu reimplementieren. Man kann die neuen Varianten deshalb nicht einzeln zu COKE in seiner alten Form dazuschalten. Es besteht die Möglichkeit entweder den alten COKE-Kernel zu verwenden, oder den neuen, der in den Bereichen konfigurierbar ist, die im Theorieteil behandelt wurden und in Kapitel 6 gekennzeichnet sind.

Erfreulich ist, daß die Reimplementation schon alleine durch das Beseitigen einiger Fehler bessere Ergebnisse bei gleicher Konfiguration wie der frühere COKE-Kernel liefert. Diese Änderungen sollen permanent übernommen werden.

Neu ist eine Statistik-Komponente, die verschiedene Daten während der Konstruktion und Optimierung protokolliert. Sie soll die Vorgänge bei der Optimierung durchschaubarer machen und beim Bewerten der Evaluationsläufe helfen. So werden z.B. die Anzahl der Zeitüberschreitungen oder die Erfolge jeder lokalen Optimierung gemerkt.

Darüberhinaus wurde COKE noch um weitere Möglichkeiten zur Laufzeitsteuerung und einem Werkzeug zur Plausibilitätsprüfung ergänzt. Im folgenden wird das System der COKE-Extensions, die im Rahmen dieser Arbeit programmiert wurden, beschrieben. Zunächst wird der Aufbau der beiliegenden CD erläutert. Danach folgt eine Anleitung, wie man die COKE-Extensions in bestehende COKE-Anwendungen integriert und wie die zur Evaluation verwendeten Anwendungen gestartet werden können. Schließlich wird anhand der Einstellungsdialoge die Bedienung der COKE-Extensions erläutert.

7.2 Aufbau der CD

Auf der CD, die dieser Arbeit beiliegt, befinden sich die Quellen zum Programmierteil der Arbeit, lauffähige COKE-Anwendungen und Protokolle der Evaluationsläufe. In den einzelnen Ordnern befinden sich speziell:

- Evaluationsdaten
  Hier befinden sich die Protokolldaten, die während der Evaluation der

COKE-Extensions gesammelt wurden. Aufgrund der Menge der Daten befinden sich diese Protokolle nicht innerhalb dieser Arbeit, sondern werden nur auszugsweise aufgeführt.

- Anwendungen
  In einem jeweils eigenen Ordner befinden sich die Zuordnungsshell COKE und die Anwendungen PEPSI, NURSE und WIZARD. Es handelt sich hierbei um Versionen, die auf einem Computersystem mit installiertem Allegro-Common-Lisp gestartet werden können.

- Quellen
  Hier befinden sich die Quellen zu allen hier beschriebenen COKE-Anwendungen und insbesondere auch die Quellen zu den Programmerweiterungen dieser Arbeit. Sie befinden sich im Pfad `\Quellen\v4.1\sources\coke\coke-x\`.

7.3 Hinzufügen der COKE-Extensions

In diesem Abschnitt werden die notwendigen Schritte beschrieben, um die Extensions in eine bestehende COKE-Anwendung einzubinden.

Zum Verständnis dieses Absatzes ist es wichtig, daß man bereits eine COKE-Anwendung programmiert hat und die Schritte zum Kompilieren der Anwendung kennt. Eine Beschreibung dafür existiert leider noch nicht. Dies wäre Aufgabe einer Technischen Referenz und kann aufgrund des Umfangs einer solchen Beschreibung nicht an dieser Stelle geschehen.

Die Variable `*inkad-system-description*` enthält eine Definition des zu ladenden Systems bzw. der zu erstellenden Anwendung. Die Definition liegt in Form einer Aliasliste vor. Durch das Hinzufügen des Schlüssels `:coke-addons` mit dem Wert `(:coke-system-x)` wird das System um die COKE-Extensions erweitert.

Der Schlüssel `:KNOWLEDGE-BASE-RUNTIME` enthält eine Liste der zur Laufzeit benötigten Wissensbasen. Insbesondere werden hier auch die Menüleisteneinträge definiert. Um die Menüeinträge für die zusätzlichen Konfigurationsdialoge hinzuzufügen, muß man der Wissensbasisliste eine weitere Wissensbasis, die Wissensbasis `coke-x-gui`, hinzufügen. Diese findet man im COKE-Referenzordner. Abbildung 7.1 zeigt ein Beispiel, in dem die notwendigen Zusätze eingefügt sind.

7.4 Laden der Systeme

COKE und seine Anwendungen wurden in der Programmiersprache LISP entwickelt. Die aktuelle Version bei Erstellung dieser Arbeit lief mit Allegro Common Lisp 5.0.1 unter Windows 95/98 und Windows NT 4.0.

Voraussetzung zum Start der Anwendungen ist ein Rechnersystem, auf dem bereits ACL 5.0.1 installiert ist. Man kann zu einen mit den Quellen der CD ein jeweils neues System bauen. Alternativ findet man im Ordner `\Anwendungen\` bereits kompilierte Versionen der Programme COKE, PEPSI und WIZARD als LISP-Image-Dateien. Es genügt zum Starten der Anwendung ein Doppelklick

```
(defvar *inkad-system-description*
        '(:KNOWLEGDE-BASE-SYSTEM-USER-CLASSES
            PEPSI-BENUTZERKLASSEN
            :KNOWLEDGE-BASE-SYSTEM COKE-systemklassen
            :KNOWLEDGE-BASE-RUNTIME (PEPSI-LAUFZEIT COKE-GUI
                                     PEPSI-GUI coke-x-gui)
            :BUILD-IMAGE NIL
            :DOCU-TOOL NIL
            :DEVELOPER-LIBRARY T
            :SYSTEM :pepsi
            :coke-addons (:coke-system-x)))
```

Abbildung 7.1: Diese „Systembeschreibung“ gibt an, welche Wissensbasis und welches COKE-Zusatzmodul geladen werden soll, um die COKE-Extensions in PEPSI zu nutzen.

Datei	Anwendung
`\Anwendungen\Pepsi\pepsi-mit-x`	PEPSI mit COKE-Extensions
`\Anwendungen\Pepsi\pepsi-ohne-x`	PEPSI ohne COKE-Extensions
`\Anwendungen\Wizard\wizard-mit-x`	WIZARD ohne COKE-Extensions
`\Anwendungen\Wizard\wizard-ohne-x`	Wizard ohne COKE-Extensions
`\Anwendungen\COKE\coke-ohne-x`	Reines COKE
`\Anwendungen\COKE\coke-mit-x`	COKE mit Extensions

Tabelle 7.1: Diese LISP-Images befinden sich auf der beiliegenden CD.

auf die entsprechende `dxl`-Datei. Tabelle 7.1 zeigt, welche Datei für welche Anwendung gewählt werden muß.

Das Laden eines Nurse-Scheduling-Systems (kurz NURSE) ist etwas aufwendiger. Um das System zu laden, führen Sie folgende Schritte aus:

1. Wechseln Sie in das Verzeichnis `\Anwendungen\Nurse\`.
2. Starten Sie das LISP-Image `coke-mit-x`.
3. Melden Sie sich als Nurse-Scheduling-Benutzer an.
4. Öffnen Sie im ACL-Entwicklerfenster das Projekt `\Quellen\v4.1\sourcen\nurse\PresentationTool\pesentation-tool.lpr`.
5. Öffnen Sie nun das Projekt `Dienstplaner GUI (needs PT).lpr` im Verzeichnis `\Quellen\v4.1\sourcen\nurse\DienstaplanerV0.92\`.
6. Starten Sie das Projekt durch Drücken der Tasten `Strg + Shift+ „R“`.

Für genauere Hinweise zur Bedienung der Systeme verweise ich auf Handbücher und Systembeschreibungen der Programme. Eine ausführliche Beschreibung von PEPSI kann man in [Herrler 1999] finden. Hinweise zur Bedienung und zu den Fähigkeiten des Nurse-Schedulers finden Sie in [Forster 1999]. Zu WIZARD gibt es ein Handbuch und eine umfassende Online-Hilfe [Inkad 1999].

Abbildung 7.2: Der Einstellungsdialog für die COKE-Extensions. Der Reiter „Lokale Optimierung“ ermöglicht eine Auswahl der Alternativen, die für sie implementiert wurden.

7.5 Bedienung der COKE-Extensions

Anhand der Dialoge zur Einstellung und Benutzung der Varianten wird in diesem Abschnitt in die Bedienung der neu hinzugekommenen Funktionen eingeführt.

7.5.1 Extensions-Einstellungsdialog

Mit dem Menupunkt *Zuordnung/Einstellungen* kann man die COKE-Extensions konfigurieren. Hier ist es möglich für die lokale und globale Optimierung unter den implementierten Varianten zu wählen. Die Standardeinstellung ist dabei so gewählt, daß sie der ursprünglichen COKE-Implementierung am nächsten kommt.

Abbildung 7.2 zeigt den Konfigurierungsdialog, bzw. seinen ersten Reiter. Hier kann man zwischen den schon früher bestehenden Varianten und den neuen Alternativen für die lokale Optimierung wählen. Folgende Liste erläutert die einzelnen Auswahloptionen. In eckigen Klammern wird, falls möglich, die entsprechende Numerierung der Übersicht aus Kapitel 6 angegeben (siehe Seite 83).

- **Lokale Optimierung**
  Wählt man hier „COKE-Original“ wird bei der lokalen Optimierung in den alten COKE-Kernel verzweigt. Unwirksam werden damit alle gewählten

Varianten und auch die Statistikkomponente. Auch die neue Laufzeitsteuerung kann hier nicht benutzt werden. Wählt man „Coke-Extensions“, so wird die die neue Implementierung verwendet.

- **Pruning von Ästen**
  Das Standardverhalten von COKE ist, daß schlecht bewertete Äste aus Performancegründen abgeschnitten werden. Dies kann hier ausgeschaltet werden. Wenn bei der Optimierung nichtmonotone Restriktionen verwendet werden, ist das Pruning nicht korrekt und kann gegebenenfalls Lösungsäste abschneiden (siehe Kapitel 4.5.4).
  [3.3.1] [3.3.2]

- **Mehrfache Störenfriede**
  Diese Auswahl entscheidet, ob jeweils nur ein Störenfried zurückgezogen wird, oder ob, wie in Kapitel 3.4.2 beschrieben, alle Störenfriede gemeinsam zurückgezogen werden.
  [3.1.3.1] [3.1.3.2]

- **Suchalgorithmus**
  Mit diesem Schalter kann man den Suchalgorithmus für die lokale Optimierung festlegen. Zur Auswahl stehen hier Bestensuche, Hillclimbing und Tiefensuche.
  [3.2.1] [3.2.2] [3.2.3]

- **Störenfriede**
  Im ersten Schritt werden die Störenfriede immer verletzungsabhängig bestimmt. Für alle weiteren kann mit diesem Schalter gewählt werden, ob die Störenfriede verletzungsabhängig oder nur abhängig von der Zuteilung ermittelt werden sollen. Einen Sonderfall bilden die Schalterstellungen „alle“ und „aus“. Bei „aus“ gibt es keine Störenfriede, also auch keine Korrekturen, bei „alle“ werden alle bereits zugeteilten Nachfrager als Störenfriede angesehen.
  [3.1.2.1] [3.1.2.2] [3.1.2.3]

- **Veränderungsbewertung**
  Hier wird bestimmt, wie Veränderungen bewertet werden sollen. Zur Wahl stehen die beiden effizienten Verfahren, die Verletzungsveränderung am Störenfried zu verwenden oder die Verletzungsveränderung am Initiator der Verletzung zu berücksichtigen. Die Unterschiede wurden in Abschnitt 3.4.3 beschrieben.
  [3.1.4.2] [3.1.4.3]

- **Zyklustest**
  Es wurden zwei der in Abschnitt 4.1.3 beschriebenen vier Zyklustests implementiert. Die Schalterstellung „Einfach“ realisiert einen Zyklustest, der mehrfaches Umsetzen desselben Nachfragers unterbindet. Die Schalterstellung „Besser“ aktiviert den Zyklustest, der mehrfaches Setzen desselben Nachfrages auf denselben Anbieter unterbindet.
  [3.1.5.2] [3.1.5.3]

Auf dem zweiten Reiter des Einstellungsdialoges kann man die Varianten der globalen Optimierung wählen. Die Variante „COKE-original“ verzweigt in die ur-

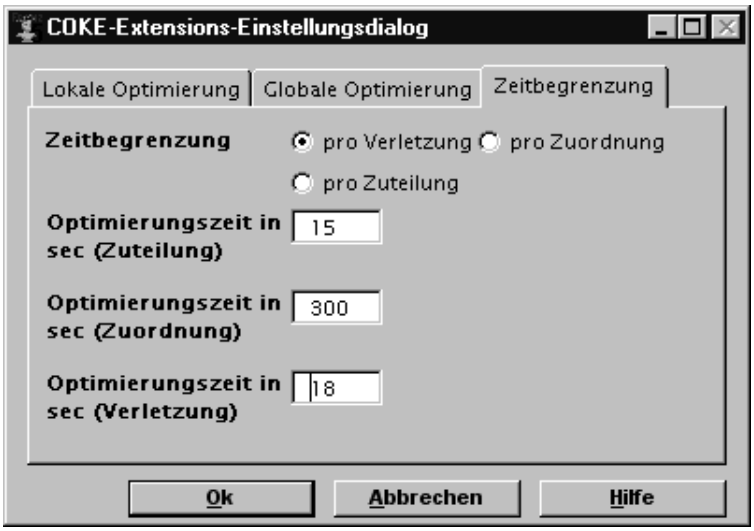

Abbildung 7.3: Der Reiter für die Einstellung der Laufzeitbegrenzung. Man kann wahlweise die Zeit pro Zuordnung oder pro Zuteilung (bzw. deren Korrektur) angeben. Eine dritte Möglichkeit ist, die Zeit abhängig vom Verletzungsgewicht anzugeben.

sprüngliche COKE-Implementation. Die Schalterstellung „pro Verletzung“ wendet ein Hillclimbing über Verletzungen zur Optimierung an [4.1.2]. Die Variante „pro Zuteilung“ realisiert ein Hillclimbing über die verletzten Zuteilungen [4.1.3].

Einstellungen zur Laufzeitbegrenzung kann man auf dem dritten Reiter vornehmen (vergleiche Abbildung 7.3). Standardmäßig ist die Zeit pro Verletzungseinheit angegeben wie in der ursprünglichen Implementation. Alternativ kann man die maximale Zeit pro Zuteilung begrenzen oder die Zeit für den gesamten Zuordnungprozeß.

Nach der Konfigurierung der Problemlösungsmethode startet man wie gewohnt die Zuordnung mit dem Menüpunkt *Zuordnung/Zuordnung starten*.

7.5.2 Dialog Plausibilitätstest

Die Plausibilitätstestkomponente startet man mit dem Menüpunkt *Einstellungen/Plausibilitätstest*. Abbildung 7.4 zeigt einen bereits gestarteten Test. Falls eine Zuordnung mit einen Restriktionsset kein ausreichend gutes bzw. brauchbares Ergebnis liefert, werden der Reihe nach zunächst die niedrig gewichteten Optimierungsrestriktionen und später auch Restriktionen mit mittlerem und schwerem Gewicht ausgeschaltet. Am Ende verbleiben nur noch die Konsistenzbedingungen. Für jedes solche Profil wird ein kompletter „Vorschlagen und Vertauschen“-Lauf durchgeführt.

Wenn ein Optimierungsergebnis ausreichend gut ist, wird das nach dem Versuchslauf durch eine grüne LED signalisiert. Die rote LED steht für einen gescheiterten Versuchslauf und eine graue für einen noch ausstehenden „Vorschlagen und Vertauschen“-Lauf. Die Beurteilung von „ausreichend gut“ kann vom Benutzer konfiguriert werden. Dazu öffnet man den Konfigurierungsdialog mit der Schaltfläche „Konfigurieren“.

Der nun geöffnete Dialog ist in Abbildung 7.5 gezeigt. Auf dem dargestellten

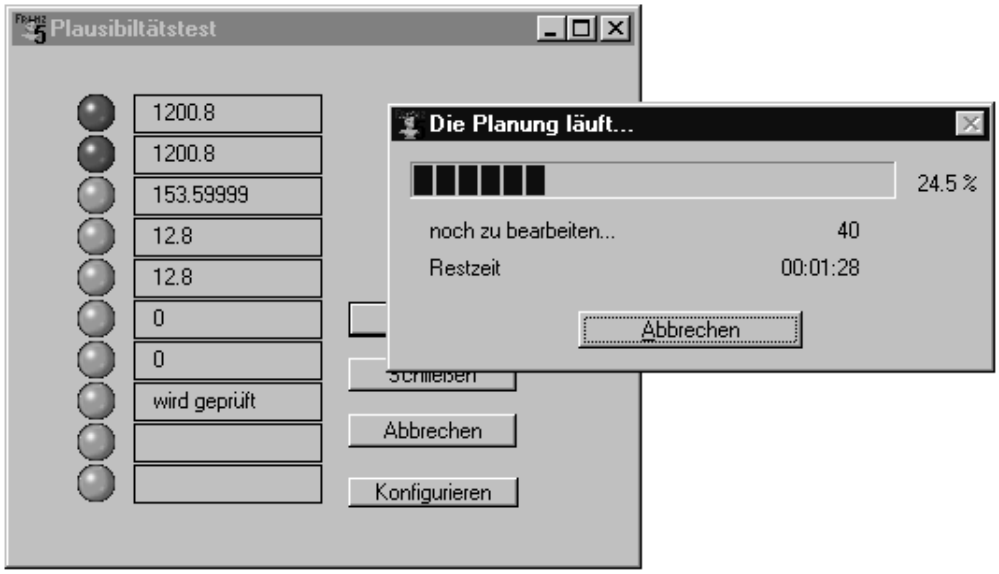

Abbildung 7.4: Beim Plausibilitätstest werden schrittweise Restriktionen weggelassen um festzustellen, ob das Problem mit weniger Restriktionen lösbar ist. Jede LED entspricht einem Restriktionsprofil. Diese werden der Reihe nach zur Zuordnung verwendet, wobei die LED Erfolg oder Mißerfolg signalisiert .

vordersten Reiter können einige Einstellungen für den Ablauf des Tests vorgenommen werden. Diese sind:

- **Steuerung**
  Hier kann man entscheiden, wie die Profile für den Plausibilitätstest erstellt werden. Der Schalter „generisch“ erzeugt automatisch 10 Profile, orientiert an den 10 Verletzungsstufen in COKE. Aktiviert man die Option „über Profile“, so werden die benutzerdefinierten Profile verwendet. Auf dem hinteren Reiter kann man dafür selbst bis zu 5 verschiedene Profile eingeben. Für jedes Profil ist es dort möglich, beliebige Restriktionen aktiv zu setzten und ihr Gewicht zu verändern.

- **Mit Optimierung**
  Wenn hier „Ja“ gewählt ist, wird nach der Zuordnung eine globale Optimierung gestartet wird. Erst nach dieser wird die Bewertung durchgeführt und nach den vorgegebenen Kriterien als „ausreichend gut“ oder nicht beurteilt.

- **Maximale plausible Bewertung**
  Zuordnungen mit einem Gesamtverletzungsgewicht, das größer ist als das hier angegebene, werden als nicht „ausreichend gut“ eingestuft. Sie werden beim Plausibilitätstest mit der roten LED dargestellt.

- **Maximal plausibler Verletzungsgrad**
  COKE besitzt 10 Stufen von Verletzungsgraden. Diese werden mit den Zahlen 1 bis 10 repräsentiert. Dabei ist 10 die stärkste Verletzung - häufig wird sie auch Konsistenzverletzung genannt. Die Verletzungsstufe 1 ist eine sehr leichte Verletzung und häufig nur Optimierungskriterium und

Abbildung 7.5: Der erste Reiter des Dialoges zur Konfigurierung der Plausibilitätsprüfung.

keine feste Voraussetzung. Zuordnungen mit einer Verletzungsstufe größer als der hier angegebenen werden als nicht ausreichend gut eingestuft. Verletzungen niedrigerer Stufen sind zulässig. Dabei ist es unerheblich, wie häufig diese Verletzung ist, solange nicht die maximal plausible Bewertung überschritten wird.

Die Schaltfläche „Speichern“ übernimmt die Konfiguration als Standard ins Benutzerprofil. Mit „Ok“ kann man den Einstellungsdialog verlassen und anschließend im Plausibilitätstestfenster mit „Starten“ den Batchlauf starten.

Profil für Profil wird nun „Vorschlagen und Vertauschen“ auf die Problemstellung angewendet. Die LED's zeigen den Verlauf der bisherigen Läufe an. Ein Fortschrittsbalken signalisiert den Status des aktuellen Laufes. In dem Feld rechts einer jeden LED wird nach Abschluß des Laufes das resultierende Verletzungsgewicht angezeigt.

Falls am Ende aller Läufe keine brauchbare Lösung gefunden ist, steht fest, daß das System auch ohne Optimierungskriterien zu berücksichtigen keine Lösung finden kann. Mit großer Wahrscheinlichkeit ist die Problemstellung damit nicht plausibel. Sicher kann man das allerdings nicht sagen, da eine Suche mit V&V, auch wenn nur Konsistenzbedingungen aktiv sind, nicht vollständig ist. Unter Umständen kann man anhand der gescheiterten Profile Schlüsse darauf ziehen, welche Restriktionen für den Mißerfolg verantwortlich sind.

Kapitel 8

Evaluation

Nach Abhandlung der Theorie und der Beschreibung des implementierten Systems folgt nun eine Evaluation der erzielten Ergebnisse. In Abschnitt 8.1 werden zunächst allgemeine Grundlagen der durchgeführten Evaluationstests behandelt. Die Abschnitte 8.2 bis 8.5 präsentieren dann die Ergebnisse der Untersuchungen und bewerten diese.

8.1 Durchführung der Evaluation

8.1.1 Wahl der Anwendung

Zur Evaluation wurden die beiden auf COKE basierenden Dienstplananwendungen PEPSI und Nurse-Scheduler benutzt. In PEPSI wurden realle Schichtdaten aus der Dienstplanung des Vorjahrs verwendet. Bei dem Test mit dem Nurse-Scheduler wurde mit einem fiktiven Planungsszenario „30 Tage ab dem 01.01.2000“ gearbeitet. Die Personalanforderungen orientierten sich an den möglichen Anforderungen in Krankenhäusern.

Die Zuordnungsprobleme Einsatzplanung bei der Bundesbahn und Einsatzplanung im Krankenhaus unterscheiden sich sehr in ihrer Komplexität. Die Menge der Nachfrager bzw. der Schichten in einem Schwesternplan ist im allgemeinen ca. 10 mal so groß als die Menge der Nachfrager in einem Dienstplan für Zugbegleiter. Unterschiedlich ist auch der Anspruch an die Optimierungsfähigkeiten der Problemlösemethode. Mit PEPSI wurden bereits ohne die COKE-Extensions vor Beginn dieser Arbeit gute Ergebnisse erzielt und die Brauchbarkeit der Wissenskomponente evaluiert. Die Dienstplanung für das Pflegepersonal befand sich zu Beginn der Arbeit noch in einem Entwicklungsstadium, in dem COKE bzw. die bis dahin existierende Implementation der Problemlösemethode keine zufriedenstellenden Lösungsergebnisse aufweisen konnte. Ob das allein mit Änderungen und Erweiterungen an COKE oder nur mit einer Verbesserung des Expertenwissens (Restriktionen, Vorschläge und Korrekturen) erreicht werden kann, ist eine Frage, die die Evaluation der Neuerungen hier beantworten soll.

8.1.2 Aufbau der Evaluationstests

Aufgrund der verschiedenen Anforderungen an die Tests wurde die Evaluation in vier Teile geteilt, die getrennt voneinander untersucht werden. Diese sind:

Wert	Einfacher Zyklustest	Genauerer Zyklustest
Gebrauchte Zeit in Sek.	130	379
Verletzungsbewertung	287*	287*
Generierte Zustände	4131	12060
Blattzustände	972	2403
Entdeckte Zyklen	1314	630
Zeitueberschreitungen	0	3

Tabelle 8.1: Gemessene Werte bei der Erstellung des PEPSI-Plans 701. Links mit einem einfachen Zyklustest [3.1.5.2] und rechts ein aufwendigerer Zyklustest [3.1.5.3]. Die Größe des erforschten Suchraum wächst, wie die Anzahl der generierten Zustände zeigt. Es werden weniger Zyklen fälschlicherweise als Zyklus erkannt. Aufgrund der gründlicheren Suche muß die Suche 3 mal durch das Zeitlimit unterbrochen werden, statt selbst aufgrund fehlender noch nicht besuchter Nachbarn zu terminieren.

- lokale Optimierung
- globale Optimierung
- Laufzeitbegrenzung
- Plausibilitätstest

Die beim Test verwendete Konfiguration wird an den entsprechenden Stellen als Konfigurationstupel angegeben. Die *Standardkonfiguration* z.B. ist (3.1.1.1 3.1.2.2 3.1.3.2 3.1.4.3 3.1.5.2 3.2.1 3.3.1 4.1.2 4.2.1 5.1.3). Als alternative Notation werden häufig auch nur die Unterschiede zur Standardkonfiguration in folgender Notation angegeben: [3.1.5.3] bedeutet z.B., daß die eine Konfiguration äquaivalent der Standardkonfiguration verwendet wird bis auf den Zyklustest. Hier wird eine alternative Variante gewählt.

8.1.3 Gemessene Werte

Eine neu implementierte Statistikkomponente mißt verschiedene Werte während der Zuordnung und Optimierung. Sie sollte zur Interpretation des Suchverlaufes und Beurteilung der Lösungsergebnisse verwendet werden. Tabelle 8.1 zeigt ein Beispiel mit einem Vergleich der Daten zu den beiden verschiedenen Zyklustests. Die Werte werden im folgenden erläutert:

- **Gebrauchte Zeit**
  Die gebrauchte Zeit wird in Sekunden angegeben. Sie ist neben der Lösungsgüte der wichtigste Indikator. Ziel ist nicht notwendigerweise eine Minimierung der benötigten Zeit, allerdings sollte zur Lösung benötigte Zeit in einem für den Anwender akzeptablen Rahmen bleiben. Desweiteren gibt die gebrauchte Zeit Hinweise auf die Güte einer Heuristik (gutes heuristisches Wissen führt schnell zu einer Lösung) und die Zeiteffizienz der Implementierung.

- **Verletzungsbewertung**
  Die Maßgabe für die Lösungsgüte ist die Verletzungsbewertung, die Summe der Gewichte aller Einzelverletzungen. Der Wert wird zusätzlich mit einem $\star$ markiert, wenn keine harten Verletzungen mehr vorliegen, die Lösung also prinzipiell schon für den Benutzer akzeptabel ist.

- **Generierte Zustände**
  Die Anzahl der generierten Zustände ist ein Indikator dafür, wie weitreichend die Suche war. Sie zeigt damit Eigenschaften und Auswirkungen der Suchverfahren und der Nachbarschaftsdefinition. Zusammen mit den erreichten Verletzungskorrekturen kann die Anzahl der generierten Zustände eine Qualitätsbeurteilung für die verwendeten Heuristiken und Suchverfahren sein. Eine gute Heuristik bzw. ein für den speziellen Fall geeignetes Suchverfahren kann bereits mit weniger generierten Zuständen Verletzungen korrigieren.

- **Zeitüberschreitungen**
  Dieser Meßwert gibt an, wie oft die lokale Suche beendet wurde, ohne ein Ergebnis zu finden. Er hilft zu erkennen, ob keine Lösung gefunden wurde, weil die Zeit nicht ausreichend war oder ob die Suche auch bei mehr Zeit zum selben Ergebnis geführt hätte. Manchmal bringt ein Ausbau der Nachbarschaften eine Steigerung der Zeitüberschreitungen, da der Suchraum gründlicher - damit auch länger - durchsucht wird. Der limitierende Faktor wird damit die Zeit.

- **Entdeckte Zyklen**
  Dieser Wert dient vor allem dazu, die Auswirkungen der verschiedenen Zyklustests zu beurteilen. Neben dem Test der Qualtität der Zyklustests können sie auch Aufschlüsse über die Struktur des Suchraumes gewonnen werden.

- **Lösungsbeschreibung**
  Wenn eine Verletzung entsteht, die ausgebessert werden muß, wird der lokale Vertauschen-Prozeß angestoßen. Das Ergebnis dieses Prozesses wird protokolliert. Die Abbildung 8.1 zeigt ein Beispiel bzw. einen kurzen Ausschnitt aus diesem Protokoll. Der Vertauschungsprozeß kann *optimal* ablaufen, d.h. die Verletzung wurde durch die Vertauschungen beseitigt, ohne daß eine neue entstand. Er kann *suboptimal* ablaufen, d.h. es wurde zwar eine Verbesserung erzielt, die Verletzung wurde jedoch nicht gänzlich aufgehoben, oder eine andere, geringere muß dafür in Kauf genommen werden. Schließlich kann ein Vertauschungsprozeß auch *ohne Lösung* bleiben, was bedeutet, daß die Suche nach einer Verletzungsauflösung oder Verletzungsverringerung erfolglos blieb.

  Interessant im Hinblick auf die Fähigkeiten des Suchverfahrens ist noch die *Stufe* einer Vertauschung. Eine Verletzungsauflösung durch eine *5-stufige* Vertauschung bedeutet, daß fünf Vertauschungen nötig waren, um eine Verletzung letztendlich zu beseitigen. Da die komplette Angabe von Lösungsbeschreibungen von bis zu 200 Optimierungen pro Versuch in dieser Arbeit keinen Platz findet, wird nur jeweils die maximale Stufe der Vertauschungen angegeben.

```
((1-stufig suboptimal)
  (5-stufig optimal)
  (0        keine lösung)
  (2-stufig suboptimal))
```

Abbildung 8.1: Beispiel für eine Lösungsbeschreibung. Die erste lokale Optimierung konnte die Verletzung mit einer einstufigen Verletzung reduzieren. Die zweite lokale Optimierung hat die Verletzung nach einer 5-stufigen Vertauschung gänzlich beseitigt, die dritte fand keine Lösung, usw.

> Die Lösungsbeschreibung läßt eine genauere Analyse des Abläufe zu. So kann z.B. erkannt werden, wie oft die lokale Optimierung überhaupt in Aktion tritt. Ein Vergleich zwischen zwei Versuchsläufen kann daraufhin untersucht werden, welche Verletzung in welchem Fall beseitigt werden konnte.

Diese Werte wurden bei allen Versuchsläufen erhoben und dienen als Grundlage zur Analyse und Bewertung der implementierten COKE-Erweiterungen.

8.2 Evaluation der lokalen Optimierung

Für die lokale Optimierung wurden insgesamt 8 neue Detailvarianten implementiert. Zusammen mit den zuvor bestehenden 7 gibt es sehr viele Kombinationsmöglichkeiten. Um Testkonfigurationen für die Effizienzuntersuchungen zu erhalten, wurde deshalb so vorgegangen, daß die Standardkonfiguration, die sich am Verhalten des alten COKE orientiert, als Basis zum Vergleich festgesetzt wurde. Davon in jeweils einer Detailvariante abweichend werden in Tabelle 8.2 acht Testkonfigurationen definiert. Um festzustellen, ob sich einzelne Verbesserungen auch additiv auswirken oder ob andere Kombinationen ein besseres Zusammenspiel ergeben, wurden zusätzlich noch drei Kombinationen von Detailvarianten zur Untersuchung ausgewählt.

Mit allen in Tabelle 8.2 aufgeführten Testkonfigurationen wurden Messungen in den beiden Problemdomänen Zugbegleiterplanung und Pflegepersonalplanung durchgeführt. Die Ergebnisse aller Versuchsläufe befinden sich in den Tabellen des Anhangs A. Die Übersicht über die Ergebnisse der Evaluation mit dem Nurse-Scheduler zeigt Tabelle A.3 auf Seite 118. In PEPSI wurden zwei Versuchsreihen durchgeführt: Tabelle A.1 zeigt die Ergebnisse des Dienstplans 714, einer sehr schweren Probleminstanz, Tabelle A.2 zeigt Ergebnisse des Dienstplanes 701, einer eher typischen Probleminstanz im Bezug darauf, wie schwierig sie zu lösen ist.

In den nun folgenden Vergleichen dienen die Tabellen aus dem Anhang als ständige Referenz. Nur an einigen Stellen sind Auszüge aus den Tabellen in den laufenden Text eingebunden.

Name der Konfiguration	Definition und Beschreibung
Basis 1	COKE-Original COKE in seiner bisherigen Implementierung
Basis 2	Standard-Extensions verwendet die Standardkonfiguration
Variante 1	[3.1.2.3] verletzungsabhängige Störenfriede
Variante 2	[3.1.4.2] Veränderungsbewertung am Störenden
Variante 3	[3.1.5.3] exakterer Zyklustest
Variante 4	[3.1.2.1] alle Nachfrager sind Störenfriede
Variante 5	[3.1.3.1] Störenfriede gemeinsam zurückziehen
Variante 6	[3.3.2] kein Pruning von Ästen
Variante 7	[3.2.2] Tiefensuche
Variante 8	[3.2.3] Hillclimbing
Kombination 1	[3.1.3.1 3.1.2.3] Störenfriede gemeinsam zurückziehen verletzungsabhängige Störenfriede
Kombination 2	[3.2.2 3.3.2] Tiefensuche kein Pruning von Ästen
Kombination 2	[3.3.2 3.1.5.3 3.1.2.3] ohne Pruning exakterer Zyklustest verletzungsabhängige Störenfriede

Tabelle 8.2: Die Tabelle zeigt die verwendeten Testkonfigurationen zur Untersuchung der lokalen Optimierung.

8.2.1 Vergleich der Originalimplementierung mit der Reimplementierung

Die Testkonfiguration „Basis 2“ entspricht einer Reimplementierung der Konfiguration „Basis 1“, dem COKE-Original. Sie unterscheiden sich grundsätzlich nicht in den algorithmischen Varianten, d.h. sie haben dasselbe Konfigurationstupel, die Ergebnisse fallen jedoch aufgrund einiger Fehler in der alten Implementierung unterschiedlich aus. Zum Vergleich und um das Ergebnis der neuen Evaluation in Relation zu bisherigen Evaluationen zu stellen, wurden deshalb auch die Ergebnisse der früheren Optimierung ermittelt.

Sowohl bei PEPSI als auch bei NURSE zeigt sich eine drastische Verbesserung der Verletzungsbewertung. Bei der Berechnung der PEPSI-Pläne läßt sich außerdem noch eine Verkürzung der Berechnungszeit feststellen. Viele Pläne, die

zuvor globale Optimierungsläufe benötigten, um eine brauchbare Lösung zu erhalten, sind jetzt bereits nach der Konstruktion und lokalen Optimierung frei von schweren Restriktionsverletzungen. Dies zeigte sich auch bei weiteren Versuchläufen mit anderen Dienstplänen.

8.2.2 Bewertung der Variante „verletzungsabhängige Störenfriede“

Bisher wurden in COKE die Störenfriede der ersten Vertauschungsstufe abhängig von der Verletzung ermittelt. Jede Verletzung hat dazu eine Startkorrekturfunktion, die störende Nachfrager angibt. In allen folgenden Schritten werden die Störenfriede nur zuteilungsabhängig ermittelt d.h. eine Störenfriedfunktion, die pauschal für alle Verletzungen gilt, gibt unabhängig von der Verletzung die Störenfriede an.

Da das Wissen bei den verletzungsabhängigen Störenfrieden viel spezieller ist, ist zu erwarten, daß ihre zusätzliche Nutzung bei mehrstufigen Vertauschungen zu einer Verbesserung des Ergebnisses führt.

Bei der Zugbegleiterplanung hat sich diese Erwartung erfüllt. Das Optimierungsergebnis dieser Konfiguration bei Dienstplan 714 (siehe Tabelle A.1) ist die Beste aller Testkonfigurationen. An der maximalen Stufe der Korrekturen, erkennt man, daß gerade die Fähigkeit mehrstufige Korrekturen zu finden zugenommen hat. Bei der Versuchsreihe mit dem Dienstplan 701 ergab sich eine geringere Verschlechterung. Diese betraf jedoch nur Optimierungskriterien und keine harten Restriktionen.

Auch bei der Pflegepersonalplanung (Tabelle A.3) verbessert sich das Ergebnis und ist das Beste unter allen Konfigurationen. Hier ist der Vergleich zu den zuteilungsabhängigen Störenfrieden allerdings nicht so fair, denn die Wissenskomponente des Nurse-Schedulers enthält keine Definition von zuteilungsabhängigen Störenfrieden, sondern nur von verletzungsabhängigen Startkorrekturen. Das ist auch der Grund, warum in dieser Versuchsreihe bei den meisten anderen Varianten maximal 1-stufige Korrekturen gefunden werden.

Einen offensichtlichen Grund für die Unterscheidung der angewandten Konzepte bei der ersten Stufe der Vertauschungen und den späteren Stufen gibt es nicht. Die neue Funktionalität, auch in späteren Stufen Störenfriede verletzungsabhängig zu ermitteln, ermöglicht zum einen bessere Ergebnisse (PEPSI) und erspart dem Wissensingenieur zum anderen die Eingabe eines weiteren Wissenselementes (NURSE).

8.2.3 Bewertung der Variante „Veränderungsbewertung am Störenden“

In Abschnitt 3.4.3 wurde schon vermutet, daß die Variante „Veränderungsbewertung am Störenden“ weniger zielstrebig und damit ineffizienter optimiert als die Variante „Veränderungsbewertung am Initiator einer Veränderung“.

Sowohl die Ergebnisse des Nurse-Schedulers als auch die Ergebnisse von PEPSI bestätigen dies. In allen Versuchsreihen ergeben sich mit der Veränderungsbewertung am Störenden deutlich höhere Verletzungsbewertungen.

PEPSI DP 714	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
Standard Extensions, einfacher Zyklustest						
Basis 2	1721,60	07:15	23940	0	2	22660
exakterer Zyklustest						
Variante 3	1721,60	12:39	38286	2	2	16530

Tabelle 8.3: Auszug aus der Tabelle A.1. Gegenübergestellt sind die beiden implementierten Variante von Zyklustests. Der exaktere Zyklustest schneidet weniger Äste hinter vermeintlichen Zyklen ab und durchsucht damit einen größeren Bereich des Suchraumes. Auf die Güte der Optimierung hat dies jedoch in diesem Fall keine Auswirkung.

Die bisher eingesetzte Veränderungsbewertung am Initiator der Vertauschung ist zwar auch kein absolutes Qualitätsmaß für eine Verbesserung, sie ist jedoch ausreichend gut und, wie die Evaluation gezeigt hat, die bessere Alternative.

8.2.4 Bewertung der Variante „exakterer Zyklustest“

Durch zeiteffektive Zyklustests wird die Suche an manchen Knoten bzw. Zuständen im Suchbaum abgebrochen, bei denen man eigentlich auf noch keinen Zyklus gestoßen ist. In Abschnitt 4.1.3 wurde vermutet, daß dadurch auch Suchäste mit potentiellen Lösungen abgeschnitten werden.

Bei der Versuchsreihe mit dem Nurse-Scheduler treten bei den Tests keine Zyklen auf, da aufgrund der Störenfrieddefinition nur einstufige Vertauschungen möglich sind. In PEPSI (vergleiche Tabelle 8.3) zeigt der Vergleich des exakteren Zyklustestes mit der Standardkonfiguration, daß weniger Zyklen erkannt und damit mehr Zustände erzeugt werden. Die bessere Durchsuchung des Suchraumes führt in diesem Fall, wie auch in den meisten untersuchten Fällen, zu keinem besseren Ergebnis.

Wenn man den geringen Nutzen bei den Versuchen betrachtet, scheint es wenig wahrscheinlich, daß man mit dem exakterem Zyklustest eine starke Verbesserung der Ergebnisse erzielen kann. Solange Zeitfaktoren keine Rolle spielen und die lokale Optimierung nicht durch Zeitschranken unterbrochen wird, ist ein exakterer Zyklustest in jedem Falle empfehlenswert, da in dem nicht durchsuchtem Suchraum potentielle Lösungszustände liegen können.

8.2.5 Bewertung der Variante „alle Nachfrager sind Störenfriede“

Ein einfacher Weg zur Intensivierung ist, jeden bereits zugeteilten Nachfrager als Störenfried zu betrachten. In diesem Falle wird keine Wissenskomponente zur Ermittlung der Störenfriede benötigt.

Bei Problemen mit wenigen Nachfragern kann dies zu sehr guten Erfolgen führen, wie auch das Beispiel der Zugbegleiterplanung in Tabelle A.1 zeigt. Das Ergebnis erfüllt alle harten Restriktionen, nur einige Optimierungskriterien bleiben unerfüllt. Allerdings ist auch die Berechnungszeit am höchsten. Beim Nurse-Scheduling zeigt sich, was bei Problemen mit mehr Nachfragern passieren kann. Der Problemlöser sucht sehr lange, jedoch in die falsche Richtung, da die Korrekturen nicht mehr wissensbasiert ablaufen. So werden über 3 Million Zustände

erzeugt und die lokale Optimierung dennoch häufig ohne Ergebnis nach Überschreiten der Zeitschranke unterbrochen.

In Problemdomänen mit wenigen Nachfragern, kann die Vorgehensweise, pauschal alle zugeteilten Nachfrager als Störenfriede anzusehen, sehr wirksam sein. Es wird zwar sehr viel Zeit benötigt, die Ergebnisse sind jedoch sehr gut. Bei einer großen Anzahl von Nachfragern führt die Optimierung jedoch nicht zum Erfolg, sondern wird häufig vorzeitig durch die Zeitbegrenzung unterbrochen.

8.2.6 Bewertung der Variante „alle Störenfriede gemeinsam zurückziehen“

Das Problem, daß an einer verletzten Zuteilung auch mehrere Störenfriede beteiligt sein können, wurde in Abschnitt 3.4.2 beschrieben. COKE in der Standardkonfiguration zieht immer nur jeweils einen Störenfried zurück, man kann jedoch Fälle konstruieren, in denen eine Verletzung damit nicht aufgelöst werden kann.

Da sich aufgrund der Wissensbasis bei NURSE die Auswirkungen der mehrfachen Störenfriede in der Testkonfiguration Variante 5 nicht zeigen, wird stattdessen die Kombination 1 mit Variante 1 verglichen. Das gemeinsame Zurückziehen bewirkt hier eine Erhöhung der Verletzungsbewertung im Vergleich zum einzelnen Zurückziehen. Ein Blick auf die anderen Werte zeigt, daß sehr viele Zustände erzeugt werden und die Optimierung häufig das Zeitlimit erreicht. Die Suche mit Kombination 1 ist also zu aufwendig für die im Testszenario einheitlich verwendete Zeit. Die Beschränkung der Zeit pro Verletzungsgewichteinheit auf 18 Sekunden resultierte in einer Gesamtzeitlaufzeit von ca. $1\frac{1}{2}$ Stunden.

In PEPSI dagegen bringt das gemeinsame Zurückziehen (Variante 5) verglichen mit der Standardkonfiguration einen Gewinn, sowohl in Bezug auf die Güte, als auch in Bezug auf die Laufzeit. Eine Kombination mit der Variante der verletzungsabhängigen Störenfriede (Kombination 1) bringt dagegen wieder eine Verschlechterung. Die Suche wird wieder zu aufwendig, wie die Anzahl der Zeitüberschreitungen schließen läßt.

Bei kleinen Problemen mit wenigen Nachfragern und wenigen Störenfrieden kann das gemeinsame Zurückziehen der Störenfriede zu Verbesserungen des Ergebnisses führen. In anderen Fällen werden die Nachbarschaften zu stark erweitert, als daß der Suchraum noch effektiv durchsucht werden könnte.

8.2.7 Bewertung der Variante „Kein Abschneiden von schlecht bewerteten Ästen“

In Abschnitt 4.5.4 wurde gezeigt, daß das Abschneiden von schlecht bewerteten Ästen nur korrekt ist, wenn sich unter den zu berücksichtigenden Restriktionen ausschließlich monotone Restriktionen befinden.

In PEPSI wurden bei der Ermittlung der Daten der Tabelle A.1 nur monotone Restriktionen aktiv gesetzt. Da das Pruning der Äste also korrekt ist, ist keine Verbesserung des Ergebnisses zu erwarten. Die Verletzungsbewertung der Variante 6 bestätigt dies.

NURSE 1.1.2000	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
Verletzungsabhängige Störenfriede						
Variante 1	1467,80	44:59	743104	14	5	4392
Verletzungsabhängige Störenfriede, ohne Pruning						
[3.1.2.3 3.3.2]	1692,80	02:13:14	1114785	105	5	16939

Tabelle 8.4: Für die Bewertung des Prunings wurde bei diesen Versuchen zur Erstellung des Pflegepersonaldienstplanes eine neue Kombination getestet.

PEPSI DP 714	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
DP 714, Standard Extensions						
Basis 2	1721,60	07:15	23940	0	2	22660
DP 714, Tiefensuche						
Variante 7	1721,60	14:23	25092	1	3	24060
DP 701, Standard Extensions						
Basis 2	287,00*	01:12	4131	0	3	1314
DP 701, Tiefensuche						
Variante 7	537,60*	01:27	4416	0	4	1368

Tabelle 8.5: Auszug der Optimierungsergebnisse aus den Tabellen A.1 und A.2. Erstellt wurden der Dienstplan 714 und der Dienstplan 701 mit PEPSI.

Die Wissenskomponente des Nurse-Schedulers enthält zwar nichtmonotone Restriktionen, die während der lokalen Optimierung aktiv sind, die fehlende Definition der zuteilungsabhängigen Störenfriede in der Wissenskomponente verhindert jedoch wieder mehrstufige Vertauschungen und macht damit auch das Schonen von Ästen in Variante 6 der Tabelle A.3 unwirksam.

Um die Nützlichkeit der neuen Option, die lokale Optimierung ohne Pruning durchzuführen, festzustellen, wurde deshalb noch ein Versuch mit einer weiteren Konfiguration durchgeführt, bei der sowohl die Option „ohne Pruning“, als auch die Option „verletzungsabhängige Störenfriede“ aktiviert ist. Diese kann in Tabelle 8.4 mit Variante 1 verglichen werden.

Der Vergleich zeigt statt einer Verbesserung durch die korrektere Vorgehensweise eine Verschlechterung der Verletzungsbewertung. Durch das Verfolgen der schlecht bewerteten Äste werden soviele Zustände erzeugt, daß die Optimierung häufiger als nötig durch die Zeitschranke unterbrochen wird.

Das Abschneiden der Äste stellt sich also als gute Heuristik heraus. Man findet schneller eine gute Lösung, auch wenn man dabei die Vollständigkeit der Suche verliert.

8.2.8 Bewertung der Varianten „Hillclimbing und Tiefensuche“

Die Tiefensuche stellte sich in den Versuchen überraschenderweise als brauchbare Alternative zur Bestensuche heraus. Wie Tabelle 8.5 zeigt, sind die Optimierungsergebnisse nur geringfügig schlechter, als die der Bestensuche. Die Laufzeit ist im allgemeinen größer, jedoch blieb sie bei den untersuchten Beispielen immer in einem vertretbaren Rahmen.

Hillclimbing ist wie erwartet sehr schnell. Nie erreicht die Suche das Zeitlimit, da vorher schon immer lokale Minima der Bewertungsfunktion erreicht werden. In aller Regel sind die Optimierungsergebnisse schlechter als bei der Bestensuche, manchmal jedoch, wie z.B. bei der Versuchsreihe mit Dienstplan 714 der Zugbegleiterplanung erreicht die Suche zufällig ein sehr gutes Ergebnis. Der Einsatz von Hillclimbing ist nicht sehr effizient[1]. Sinnvolle Einsatzmöglichkeiten können sich aber zum Beispiel bieten, wenn man schnell eine Ausgangszuordnung für die globale Optimierung erzeugen möchte.

8.2.9 Fazit

Die aufwendigeren Varianten und insbesondere Kombinationen benötigen massiv mehr Zeit. Ein deutlicher Hinweis sind die verstärkt auftretenden Zeitüberschreitungen bei den Kombinationen 1, 2 und 3 in beiden Domänen. Kombination 3 bei der Pflegepersonalplanung zeigt, daß es damit nicht nur bei einem Knappwerden der Ressource Zeit bleibt, sondern auch an die Grenzen des Speichers gegangen wird.

Einzelne Varianten bringen jedoch durchaus einen Effizienzgewinn und Kombinationen können bei Problemen mit weniger Nachfragern, wie z.B. der Zugbegleiterplanung eine Verbesserung der Optimierungsleistung bringen. Die erfolgreichste Änderung ist, auch bei mehrstufigen Veränderungen die Störenfriede verletzungsabhängig zu ermitteln.

8.3 Evaluation der globalen Optimierung

Es stehen zwei implementierte Konzepte bei der globalen Optimierung zur Auswahl. Eine Hillclimbing-Suche über die verletzten Zuteilungen und eine Hillclimbing-Suche über jede einzelne Verletzung. Um die Ergebnisse der globalen Optimierung unabhängig von der eingebetteten lokalen Suche zu beurteilen, wurde für je zwei zu vergleichende Läufe dieselbe Ausgangszuordnung erzeugt und dasselbe Konfigurationstupel für die lokale Suche verwendet. Jede Konfiguration wurde mit zwei Ausgangszuordnungen getestet. Ein Ausgangszustand war dabei das Ergebnis einer guten lokalen Optimierung mit einem geringen Verletzungsgewicht. Der zweite Ausgangszustand war das Ergebnis einer schlechteren lokalen Optimierung mit einem hohen Verletzungsgewicht.

In den Tabellen B.2 und B.1 sind die Ergebnisse der Versuchsreihen zur globalen Optimierung aufgelistet. Die Korrektur pro Verletzung erzeugt allgemein mehr Zustände, da die lokale Korrekturfunktion häufiger (nämlich für jede Verletzung statt jeder verletzten Zuteilung) aufgerufen wird. Sie benötigt damit etwas mehr Zeit. Es läßt sich kein signifikanter Unterschied in der Bewertung feststellen; der Ausreißer bei den Testdaten des Nurse-Schedulers mit einer extrem besseren Verletzungsbewertung beim Hillclimbing über Zuteilungen läßt sich damit nicht erklären und ist wohl zufällig durch eine veränderte Reihenfolge der Zuteilungen entstanden.

Aufgrund dieser Evaluation kann man die Effizienz der Alternativen nicht bewerten. Zu divers sind die in den Beispielen erhaltenen Daten. Auch mit weiteren Versuchsläufen in den untersuchten Domänen konnte kein eindeutigeres

[1] Es ist zwar schnell, aber zur Effizienz gehört auch die Qualität des Ergebnisses

Ergebnis erhalten werden. Keine der Varianten ist wesentlich schlechter als die andere.

8.4 Evaluation der Zeitbegrenzungskomponente

In Abschnitt 5.2 wurden verschiedene Contract-Modelle vorgestellt. Die Modelle „Zeit pro Zuordnung“ und „Zeit pro Zuteilung“ unterscheiden sich nur in der Eingabe, müssen also in der Wirkung nicht getrennt untersucht werden. Die Evaluation beschränkt sich im folgenden also auf einen Vergleich der Zeitangabe pro Verletzungsgewichtseinheit und der Angabe pro Zuteilung.

Im folgenden sollen zwei Dinge untersucht werden. Zum einen, wie sich das Ergebnis in Abhängigkeit von der Rechenzeit entwickelt, zum anderen ob das Contract-Modell „Zeit pro Zuteilung“ die verwendete Rechenzeit wirklich kontrollierbarer macht.

Tabelle C.1 zeigt die durchgeführten Versuche. Beim Versuch mit der Standardkonfiguration erkennt man, daß auch bei starker Zeitreduzierung, die Qualität der gefundenen Lösung nicht abnimmt. Anhand der Anzahl der generierten Zustände kann man feststellen, daß aber ein wesentlich kleinerer Teil des Suchraumes durchsucht wird. Die bei der Standardkonfiguration vorkommenenden zweistufigen Vertauschungen werden also sehr schnell gefunden. Eine Verkürzung der Zeit macht scheinbar nicht sehr viel aus. Um dies zu überprüfen wurde derselbe Test mit Kombination 3 durchgeführt. Diese Konfiguration enthält bei einem Zeitkontingent von 60 Sekunden pro Zuteilung 7-stufige Vertauschungen. Erst bei einer Reduktion dieser Zeit auf 15 Sekunden verschlechtert sich die Zuordnungsbewertung. Gleichzeitig reduziert sich die maximale Stufe der Vertauschungen. Eine mehrstufige Korrektur, die zuvor gefunden wurde, wird nun bei der Suche aus Zeitmangel nicht mehr erreicht.

Die Versuchsreihe mit dem Nurse-Scheduler in Tabelle C.2 zeigt, daß mehr Zeit für Korrekturen nicht notwendigerweise ein besseres Ergebnis bedeutet, auch wenn dies im allgemeinen der Fall ist. Dort steigt mit dem Zuwachs des Zeitkontingents von 30 auf 60 Sekunden auch die Verletzungsbewertung an. Eine genauere Untersuchung der Läufe ergab, daß der Lauf mit dem größeren Zeitkontingent bei den frühen Zuteilungen besser optimierte, damit jedoch einen eingeschränkteren Zustand erreichte und bei späteren Zuteilungen weniger Verbesserungen fand. Zusätzliche strategische Restriktionen, die in frühen Zuteilungen die Suche lenken, könnten hier gegebenenfalls für eine Verbesserung der Optimierung sorgen.

Bei der Zeitangabe pro Zuteilung kennt man zwar im Gegensatz zur Zeitangabe pro Verletzungseinheit eine Maximalzeit; im Regelfall wird die Berechnungszeit jedoch weit darunter liegen, da einige Zuteilungen nicht korrigiert werden müssen und die Korrektur anderer in weniger als der Maximalzeit durchgeführt werden kann. Es ist mit den neuen Contract-Modellen ebenfalls notwendig, je nach Anwendung und Problem ein Gespür dafür zu entwickeln, wie sich die Zeitangabe auf die Maximalzeit auswirkt.

Ob eine bessere Kontrolle der Laufzeit durch eines der beiden anderen Modelle erreicht werden kann, bleibt eine interessante Frage. Die Versuchsreihe mit dem Nurse-Scheduler zeigt ein mögliches Problem, das bei der Zeitkontingentermittlung durch einen Vorlauf auftreten kann. Wird bei einer der ersten Zuteilungen

durch ein größeres Zeitkontingent eine Verbesserung erreicht, kann dadurch möglicherweise die gesamte Situation für die späteren Zuteilungen verändert sein. Eine Abschätzung, wie dramatisch diese Auswirkungen sein können, ist jedoch theoretisch nicht zu ermitteln und muß in Versuchen bestimmt werden.

8.5 Evaluation der Plausibilitätstestkomponente

Die Plausibilitätstestkomponente ist keine algorithmische Verbesserung in dem Sinne, daß sie bessere Ergebnisse ermöglicht oder den Zuordnungsvorgang bezüglich der Laufzeit effizienter macht. Sie dient lediglich dem Benutzer als Unterstützung, herauszufinden, wie plausibel eine Problemstellung ist.

Dementsprechend muß auch die Evaluation anders gestaltet sein. Die Kriterien zur Evaluation sind:

- Kann eine unlösbare Problemstellung erkannt werden ?
- Können hinderliche Restriktionen oder Restriktionsgruppen erkannt werden ?

Als Problemdomäne zur Evaluation wurde die Zugbegleiterplanung ausgesucht. Die Restriktionen dieser Problemdomäne kann man nach ihrer Funktion in folgende Gruppen einteilen.

1. Weiche strategische Restriktionen
   Eine strategische Restriktion ist z.B., die Übergänge zwischen Schichten bei frühen Zuteilungen möglichst kurz zu halten, um mehr Freiraum für spätere Schicht-Zuteilungen zu haben. Diese Restriktion verwirklicht die Auswahl eines „least constraining value“. (siehe Abschnitt 3.2.3).

2. Weiche und härtere Optimierungsrestriktionen
   Die von diesen Restriktionen gestellten Bedingungen sind nicht gesetzlich vorgeschrieben, steigern aber die Qualität eines Dienstplanes. So soll z.B. ein vorgegebenes Ruheschema eingehalten werden oder möglichst nur eine Schicht am Tag zugeteilt sein.

3. Arbeitsrechtliche und gesetzliche harte Restriktionen
   In dieser Restriktionsgruppe befinden sich Regelungen der Dienstverordnung und gesetzliche Bestimmungen. Solche Regelungen sind beispielsweise das Einhalten eines zeitlichen Mindestabstandes zwischen zwei Schichten oder eine maximale Arbeitszeit pro Woche. Das Verletzen dieser Restriktionen liefert eine schwere Verletzung, denn ein Dienstplan darf nur verwendet werden, wenn er diese Restriktionen einhält.

4. Logische Bedingungen
   Die Problemstruktur hat einige logische Voraussetzungen, die eingehalten werden müssen, um einen sinnvollen Plan zu erhalten. So dürfen sich Schichten nicht überlappen, da ein Mitarbeiter nicht gleichzeitig in zwei Schichten arbeiten kann. Auch daß eine Montagsschicht nur an einem Montag zugeordnet werden kann, ist eine logische Bedingung und steht noch über den gesetzlichen Vorschriften.

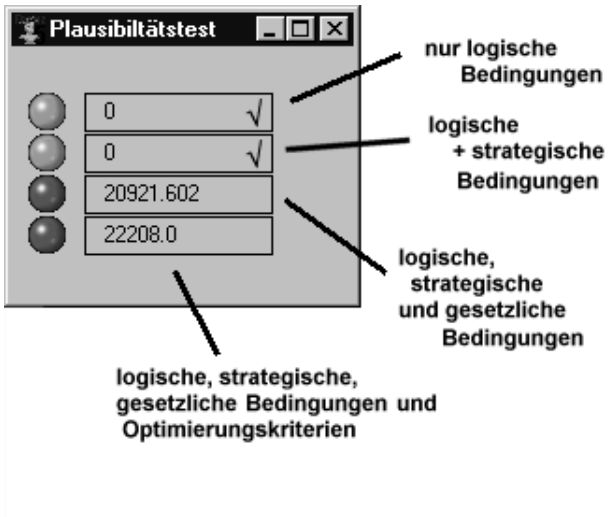

Abbildung 8.2: Ergebnis des Plausibilitätstests für Dienstplan 714, wenn er statt 11 Wochen auf 10 Wochen geplant wird. Während die logischen Bedingungen noch eingehalten werden können, scheitert die Zuordnung bereits bei der zusätzlichen Berücksichtigung der gesetzlichen Vorgaben.

Für den Versuch werden die Profile nicht generisch aus den Verletzungsgewichten ermittelt, sondern es werden Profile orientiert an diesen Gruppen angelegt. Im ersten Profil befinden sich nur die logischen Bedingungen, im zweiten die logischen und strategischen, im dritten werden die gesetzlichen Restriktionen hinzugenommen und im letzten Profil sind schließlich alle Restriktionen aktiviert.

8.5.1 Erkennen einer unlösbaren Problemstellung

Die vorhandenen Testdaten stammen aus der reallen Dienstplanung und die Schichten können jeweils ohne schwere Verletzungen zugeordnet werden. Eine unlösbare Situation kann z.B. dadurch erzeugt werden, daß eine Schichtenmenge, die für einen 11-Wochen-Dienstplan vorgesehen ist, in einen 10-Wochen-Dienstplan eingeplant wird.

Die Abbildung 8.2 zeigt das Ergebnis eines Plausibilitätstests für den Dienstplan 714, wenn er statt 11 Wochen auf 10 Wochen geplant wird. Während die logischen Bedingungen (z.B. die Überlappungsfreiheit) noch eingehalten werden können, scheitert die Zuordnung bereits bei der zusätzlichen Berücksichtigung der gesetzlichen Vorgaben. Die Zuordnung scheitert deshalb umso mehr, wenn sie versucht, die Optimierungskriterien zu berücksichtigen. Mit großer Wahrscheinlichkeit kann bei diesem Ergebnis davon ausgegangen werden, daß ein Zuordnen dieser Schichtmenge auf 10 Wochen nicht plausibel ist. Um absolute Gewißheit zu erhalten, müßte man ein vollständiges Verfahren benutzen. Ein Approximationsverfahren wie V&V, kann ein sicheres Ergebnis nicht garantieren.

Als Gegenprobe wird in der Standardkonfiguration ein Plausibilitätstest mit 11 Wochen durchgeführt. Diese Konfiguration führt bei Aktivierung aller Restriktionen nicht zu einer Lösung, obwohl es eine gibt. Der Plausibilitätstest bzw. un-

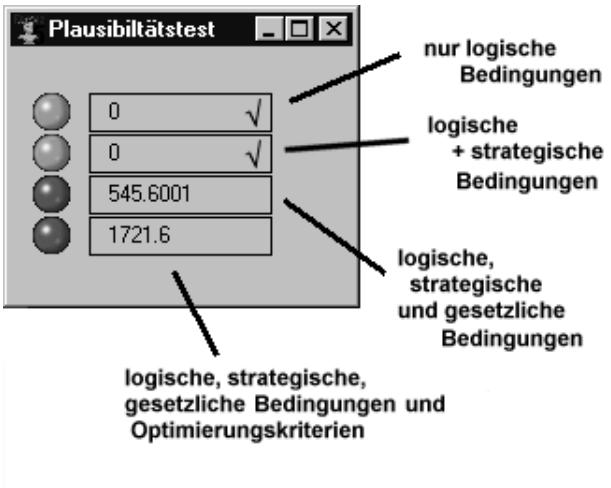

Abbildung 8.3: Ergebnis des Plausibilitätstests für Dienstplan 714, wenn er - wie gedacht - auf 11 Wochen geplant wird. Die Planung scheitert nicht allein an den Optimierungskriterien.

sere Interpretation des Ergebnisses sollte also als Antwort liefern, daß die Problemstellung plausibel ist. Abbildung 8.3 zeigt die Ergebnisse des Durchlaufs. Die Planung scheitert nicht allein an den Optimierungskriterien. V&V findet in dieser Konfiguration der lokalen Optimierung keine Lösung, die **alle** gesetzlichen Bestimmungen einhält. Das Verletzungsgewicht beim dritten Durchlauf ist jedoch bereits so gering, daß man vermuten kann, daß man das Ergebnis vielleicht noch weiter optimieren kann.

Eine nicht lösbare Problemstellung ist mit dem Plausibilitätstest also nur bedingt identifizierbar. Es ist notwendig, die Ergebnisse des Plausibilitätstest zu interpretieren und das ist manchmal sehr schwierig. Mit speziellem Wissen über die Problemdomäne kann man effektivere Plausibilitätstests erstellen. So warnt PEPSI vor der Zuordnung, wenn die durchschnittliche Arbeitszeit pro Woche stark von einem Erfahrungswert abweicht. Für den Fall, daß kein solches Wissen zur Verfügung steht, kann die Plausibilitätsprüfung jedoch ein zusätzliches Mittel zur Entscheidung des weiteren Vorgehens sein: Wenn das Problem plausibel scheint, kann man dem Algorithmus mehr Rechenzeit zum Optimieren zur Verfügung stellen oder nach einer alternativen Problemlösemethode suchen. Wenn die Problemstellung nicht plausibel scheint, ist es aussichtsreicher, die Problemstellung zu modifizieren.

8.5.2 Erkennen hinderlicher Restriktionen

Der zweite zu evaluierende Teil des Plausibilitätststestes, das Erkennen hinderlicher Restriktionen, soll helfen, herauszufinden, wie die Problemstellung modifiziert werden kann. Eine Modifikation der Problemstellung kann beispielsweise das Deaktivieren von Restriktionen sein, aber auch eine Veränderung der Nachfrager oder Anbietermengen.

Im vorherigen Beispiel ist die Menge der Restriktionen, an der die Zuteilung scheitert, die Menge der gesetzlichen Vorschriften, darunter auch die maximale Wochenarbeitszeit. Die überschrittene Wochenarbeitszeit hat ihre Ursache unter

anderem darin, daß zuviele Schichten auf einen zu kurzen Zeitraum zugeteilt wurden.

Wenn man eine Restriktionsgruppe als „lösungsverhindernd“ identifiziert, kann man in einem zweiten Plausibilitätstest Profile durch Teilung dieser Restriktionsgruppe erzeugen. So kann man genauere Aufschlüße über verhindernde Restriktionen bekommen. Ein generischer Ansatz dazu scheitert an der Anzahl der möglichen Aufteilungen in Gruppen.

Noch schwieriger als eine Interpretation der Plausibilität ist der Schluß auf die Ursache des Scheiterns. Eine allgemeine Aussage kann jedoch nicht getroffen werden, da dies stark von der Problemdomäne und den realisierten Restriktionen abhängt.

8.5.3 Fazit

Man kann mit diesem Werkzeug die Plausibilität einer Problemstellung nicht nachweisen oder widerlegen sondern nur abschätzen. Dafür ist es notwendig, die erhaltenen Ergebnisse abhängig von der Problemdomäne zu interpretieren.

Nützlich wird die Plausibilitätstestkomponente, wenn in einer Problemdomäne schon Restriktionen als „kritisch“ bekannt sind. Sehr einfach können dann mit Hilfe der Profile verschiedene Restriktionssets durchprobiert werden. In den vorhandenen Problemdomänen, bzw. deren bestehender COKE-Modellierung sind leider keine solchen kritischen Restriktionen bekannt.

Da auch die Gewichtungen der Restriktionen verändert werden können, kann man das Werkzeug auch als Unterstützung beim Wissenserwerb benutzten. So kann der Entwickler zum Beispiel herausfinden, mit welchen Restriktionsgewichtungen die Planung besonders erfolgreich verläuft. Der Grundrahmen für die Gewichtung wird dabei üblicherweise feststehen. Eine Konsistenverletzung bleibt eine Konsistenzverletzung, aber das Gewicht verschiedener Optimierungskriterien untereinander kann durchaus variabel sein und unterschiedliche Ergebnisse liefern.

Kapitel 9

Zusammenfassung und Ausblick

9.1 Zusammenfassung

Den Ausgangspunkt für die vorliegende Arbeit bildete die Beobachtung, daß Anwendungen, die auf dem Shellbaukasten COKE basieren, scheinbar noch nicht das Maximum an möglicher Optimierungsleistung erreichten. COKE implementiert „Vorschlagen und Vertauschen“, eine leicht verständliche, der menschlichen Vorgehensweise nachempfundene Problemlösungsmethode. Doch so einfach sie dadurch auch zu verstehen ist, gleicht sie der menschlichen Vorgehensweise auch in den unzähligen Möglichkeiten zur Variation und Optimierung. Dadurch läßt sich die Methode nur sehr schwer im Detail schematisieren.

Nach der Klärung der Grundlagen von Zuordnungsproblemen in Kapitel 2 wurde deshalb in Kapitel 3 die Problemlösemethode analysiert. Unumstößliche Schritte wurden dabei spezifiziert, bei Schritten, die Variationen zulassen, wurde nach Alternativen gesucht. Probleme und Wechselwirkungen der Varianten wurden beschrieben und daraus eine Bewertung der neuen Ideen in Bezug auf die Effizienz abgeschätzt.

In Kapitel 4 wurden verschiedene Suchalgorithmen für den Einsatz in „Vorschlagen und Vertauschen“ untersucht. Zuerst mußten dazu die Beschaffenheit und die Eigenschaften der zugrundeliegenden Suchraumstrukturen beschrieben werden. Nach einer Klassifikation der Suchraumstrukturen in drei Haupttypen wurden die klassischen Suchverfahren Bestensuche, Tiefensuche, Hillclimbing und Simulated Annealing im Hinblick auf Anwendbarkeit und Wirkung in V&V untersucht.

Das Kapitel 5 behandelte weitere Erweiterungsmöglichkeiten der „Vorschlagen und Vertauschen“-Methoden, die speziell einen Effizienzgewinn im Bezug auf die Mensch-Maschine-Kommunikation bewirken. Vorgestellt wurde die Spezifikation eines Werkzeuges zur Plausibilitätsprüfung der Aufgabenstellung sowie verschiedene Modelle zur Benutzerkontrolle der Optimierungslaufzeit.

In Kapitel 6 wurden die theoretisch erarbeiteten Variationen zusammengefaßt und in Abschnitt 8 die zur Implementation ausgesuchten Neuerungen evaluiert. Die verschiedenen Varianten zur Optimierung und Zeitkontingentierung wurden an zwei Anwendungen, dem Nurse-Scheduler und PEPSI evaluiert. Das Werkzeug zur Plausibilitätsprüfung konnte mangels geeigneten Beispielen in den untersuchten Anwendungen nur schwach evaluiert werden.

9.2 Bewertung

Die in dieser Arbeit erstellten Erweiterungen und die teilweise Reimplementierung des COKE-Kernels können deutlich bessere Optimierungsergebnisse bei früher schon bestehenden Problemdomänen erreichen. Vor allem hat durch die Verbesserungen die Fähigkeit, mehrstufige Veränderungen zu finden, sehr zugenommen. Durch das neue Konzept der verletzungsabhängigen Störenfriede, konnte das die Verletzungsbewertung in einigen Fällen auf weniger als 1/3 der ürsprunglichen Bewertung reduziert werden.

Bei der Zugbegleiterplanung, bei der früher gelegentlich interaktiv eingegriffen werden mußte, können jetzt problemlos ohne Benutzereingriff Lösungen gefunden werden. Die Erfolge bei der Dienstplanerstellung waren nicht ganz so groß; zwar konnten Verbesserungen bei der Optimierungsleistung erzielt werden, jedoch noch nicht ein brauchbarer Dienstplan erzeugt werden. Dies liegt jedoch an dem frühen Entwicklungsstadium, in dem sich die Wissenskomponente des Nurse-Schedulers befindet. Weitere Arbeit an der Wissensbasis kann hier die Planung verbessern und so auch den Unterschied zwischen den Varianten deutlich machen.

Aufgrund der Komplexität der Zusammenhänge zwischen Problemlöserelementen und Wissenskomponente, den vielfältigen Wechselwirkungen untereinander, ist ein optimales Zusammenspiel dieser Komponenten eine schwierige Aufgabe und erfordert vom Wissensingenieur wie vom Experten viel Hintergrundwissen über die Funktionsweise der Problemlösemethode und der Zuordnungsshell. Es gelang in dieser Arbeit einige dieser Zusammenhänge festzustellen und zu erklären. So wurden bei der Evaluation z.B. Relationen zwischen der Ausführlichkeit der Nachbarschaften und den Zeitüberschreitungen untersucht und Auswirkungen von Restriktionseigenschaften (Monotonie) auf die Exaktheit der Suche theoretisch erklärt.

Die gesammelten Erfahrungen aus der Evaluation und das Wissen über die Zusammenhänge können künftig nutzbringend beim Wissenserwerb zu anderen Problemdomänen eingesetzt werden. Problem bei der Entwicklung früherer COKE-Anwendungen war mehrfach, daß die genaue Funktion und Verwendung der Wissenselemente nicht bekannt und dokumentiert war. Dadurch wurden beim Wissenserwerb Fehler gemacht, die die eigentliche Optimimierungsleistung von COKE reduzierten.

9.3 Ausblick

Der nächste Schritt ist nun, die bisher nur als Aufsatz realisierten COKE-Extensions als festen Bestandteil in COKE zu integrieren. Dort können sie problemspezifisch konfiguriert werden. Da der kommerzielle Einsatz der COKE-Anwendungen Nurse-Scheduler und WIZARD geplant ist, kann so bald auch eine Evaluation in einer realen Problemumgebung stattfinden.

Einige der in dieser Arbeit vorgestellten und nur theoretisch vorgestellten Varianten versprechen weiterhin einen zusätzlichen Effizienzgewinn. Sie können in Fortsetzung dieser Arbeit, gekoppelt mit zusätzlichen Wissenserwerb in den einzelnen Domänen, zu einer Verbesserung der Optimierungsleistungen beitragen.

Literaturverzeichnis

[Burkard & Derigs 1980] R.E. Burkard and U. Derigs: *Assignment and Matching Problems — Solution Methods with FORTRAN Programs*, volume 184 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Heidelberg, Berlin, 1980.

[Busacker & Goven 1961] R. G. Busacker and P.J. Goven: *A Procedure for Determining a Family of Minimal Cost Network Flow Patterns*. ORO Technical Report 15, John Hopkins University, Operations Research Office, 1961.

[Busch 1997] Ciske Busch: *Spezifikation eines kooperativen Werkzeugs zur Lösung komplexer Zuordnungsprobleme*. Diplomarbeit, Universität Würzburg, 1997.

[Duden 1988] Duden: *Duden - Informatik*. Dudenverlag, 1988.

[Forster 1999] Frank Forster: *Automatische Generierung von Oberflächen zur interaktiven Lösung von Zuordnungsproblemen*. Diplomarbeit, Universität Würzburg, 1999.

[Grosche & Ziegler 1990] Grosche und Ziegler: *Taschenbuch der Mathematik*. Teubner, 1990.

[Guy & Steele 1990] L. Guy and Jr. Steele: *Common Lisp*. Digital Press, 1990.

[Herrler 1999] Rainer Herrler: *PEPSI ein Tool zur interaktiven Dienstplanung des BORD-Personals der Deuschten Bahn AG*. Studienarbeit, Universität Würzburg, 1999.

[Hestermann 1996] Christian Hestermann: *Wissensbasierte Any-Time-Suche bei diskreter und Kontinuierlicher Zuordnung*. internes Papier, page 01, 1996.

[Hestermann 1997] Christian Hestermann: *Wissensrepräsentation in Wizard*. internes Papier, page 01, 1997.

[Inkad 1999] Inkad: *Wizard-Einführung*. Universität Würzburg, veröffentlicht im Rahmen des INKAD-Projektes, 1999.

[Klügl 1994] Franziska Klügl: *Optimierung komplexer Zuordnungsprobleme durch Simuliertes Ausglühen und Genetische Algorithmen*. Diplomarbeit, Universität Würzburg, 1994. Diplomarbeit.

[Mitchell 1997] Tom Mitchell: *Machine Learning*. McGraw-Hill, Inc, 1997.

[Nilsson 1980] Nils J. Nilsson: *Principles of Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, New York, 1980.

[Norvig 1992] Peter Norvig: *Artificial Intelligence Programming*. Morgan Kaufmann Publishers, 1992.

[Poeck & Puppe 1992] Poeck and Puppe: *COKE : Effizient solving of complex assignment problems with the propose-and-exchange method*. 5th International Conference on Tools with Artificial Intelligence, 1992.

[Poeck & Gappa 1993] Karsten Poeck and Ute Gappa: *Making role limiting shells more flexible*. In Nathalie Aussenac, Guy Boy, Brian Gaines, Marc Linster, Jean-Gabriel Ganascia, and Yves Kodratoff (eds.): *Knowledge Acquisition for Knowledge-Based Systems; Proc. of EKAW '93*, volume 723 of *Lecture Notes in Artificial Intelligence*, pages 103–122, Springer Verlag, Berlin, 1993.

[Poeck 1994] Karsten Poeck: *Konfigurierbare Problemlösungsmethoden am Beispiel der Problemklassen Zuordnung und Diagnostik*. Doktorarbeit, Universität Würzburg, 1994. erscheint demnächst im infix-Verlag.

[Poeck 1995] Karsten Poeck: *Konfigurierbare Problemlösungsmethoden am Beispiel der Problemklassen Zuordnung und Diagnostik*, Band 86 der Reihe *DISKI*. infix, Sankt Augustin, 1995.

[Puppe 1988] Frank Puppe: *Einführung in Expertensysteme*. Springer, 1988.

[Puppe 1990] Frank Puppe: *Problemlösungsmethoden in Expertensystemen*. Springer, 1990.

[Rich & Knight 1991] E. Rich and K. Knight: *Artificial Intelligence*. McGraw-Hill, Inc, 1991.

[Russell & Zilberstein 1991] Stuart Russell and Sholomo Zilberstein: *Composing real-time systems*. Proceedings of the IJCAI, 01:212–217, 1991.

[Russell & Norvig 1995] Stuart Russell and Peter Norvig: *Artificial Intelligence — A Modern Approach*. Prentice Hall, 1995.

[Schiffer 1998] Stefan Schiffer: *Visuelle Programmierung*. Addison-Wesley-Longman, Bonn, 1998.

Anhang A

Tabellen zur lokalen Optimierung

PEPSI DP 714	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
COKE-Original						
Basis 1	3539,20	16:12				
Standard Extensions						
Basis 2	1721,60	07:15	23940	0	2	22660
Verletzungsabhängige Störenfriede						
Variante 1	452,00*	06:26	18339	0	7	4009
Veränderung am Störenden						
Variante 2	3660,00	07:25	41489	0	6	94840
exakterer Zyklustest						
Variante 3	1721,60	12:39	38286	2	2	16530
alle Störenfriede						
Variante 4	541,60*	22:20	62509	6	3	2610
Störenfriede gemeinsam zurückziehen						
Variante 5	1331,20	04:02	24796	1	3	1559
Kein Abschneiden von schlecht bewerteten Ästen						
Variante 6	1721,60	19:30	61880	6	2	76850
Tiefensuche						
Variante 7	1721,60	14:23	25092	1	3	24060
Hillclimbing						
Variante 8	1235,00	0:35	1595	0	2	220
Störenfriede gemeinsam zurückziehen, verletzungsabhängige Störenfriede						
Kombination 1	3539,20	26:22	92900	10	0	2159
Tiefensuche, ohne Pruning						
Kombination 2	1235,20	24:09	9135	12	2	17380
verletzungsabhängige Störenfriede, exakterer Zyklustest, ohne Pruning						
Kombination 3	678,40*	19:32	49419	9	7	5093

Tabelle A.1: Gemessene Werte bei der Erstellung des PEPSI-Plans 714. Dieser Dienstplan enthält 55 Nachfrager und ist einer der schwierigsten Dienstpläne der zur Verfügung stehenden Testdaten. Bei einigen Testkonfigurationen dieser Versuchsreihe gelingt es dennoch, ohne anschließende globale Optimierung einen brauchbaren Dienstplan zu erstellen.

PEPSI DP 701	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
COKE-Original						
Basis 1	2994,40	10:54				
Standard Extensions						
Basis 2	287,00*	01:12	4131	0	3	1314
Verletzungsabhängige Störenfriede						
Variante 1	438,00*	02:46	8661	0	3	2593
Veränderung am Störenden						
Variante 2	21820,80	10:00	28143	2	4	119223
exakterer Zyklustest						
Variante 3	287,00*	05:45	21052	3	3	1727
alle Störenfriede						
Variante 4	308,00*	17:44	54791	6	5	2826
Störenfriede gemeinsam zurückziehen						
Variante 5	17328,00	07:33	43967	3	7	2353
Kein Abschneiden von schlecht bewerteten Ästen						
Variante 6	287,00*	09:31	39065	5	3	53037
Tiefensuche						
Variante 7	537,60*	01:27	4416	0	4	1368
Hillclimbing						
Variante 8	2073,60	00:25	1338	0	3	403
Störenfriede gemeinsam zurückziehen, verletzungsabhängige Störenfriede						
Kombination 1	2352,80	17:56	82758	10	7	3774
Tiefensuche, ohne Pruning						
Kombination 2	432,00*	36:35	7277	9	5	2842
verletzungsabhängige Störenfriede, exakterer Zyklustest, ohne Pruning						
Kombination 3	416,00*	12:53	34718	7	3	2044

Tabelle A.2: Gemessene Werte bei der Erstellung des PEPSI-Plans 701. Dieser Dienstplan hat 44 Schichten, die von 10 Zugbegleitern geleistet werden müssen.

NURSE 1.1.2000	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
COKE-Original						
Basis 1	8616,80	00:19				
Standard Extensions						
Basis 2	4708,80	00:34	5491	0	1	0
Verletzungsabhängige Störenfriede						
Variante 1	1467,80	44:59	743104	14	5	4392
Veränderung am Störenden						
Variante 2	10382,04	00:30	5147	0	1	0
exakterer Zyklustest						
Variante 3	5524,00	04:02:04	3454421	107	2	14538
alle Störenfriede						
Variante 4	4708,80	01:12:34	1362032	0	1	5144
Störenfriede gemeinsam zurückziehen						
Variante 5	4708,80	00:44	10635	0	1	0
Kein Abschneiden von schlecht bewerteten Ästen						
Variante 6	4708,80	00:41	5491	0	1	0
Tiefensuche						
Variante 7	4708,80	00:72	5491	0	1	0
Hillclimbing						
Variante 8	13650,77	03:54	3485	0	0	0
Störenfriede gemeinsam zurückziehen, verletzungsabhängige Störenfriede						
Kombination 1	8617,71	01:37:33	1257026	41	3	380
Tiefensuche, ohne Pruning						
Kombination 2	4708,80	02:20	5491	0	1	0
verletzungsabhängige Störenfriede, exakterer Zyklustest, ohne Pruning						
Kombination 3	Der Speicher von 256 MB wurde überschritten.					

Tabelle A.3: Verschiedene Konfigurationen bei der lokalen Optimierung eines Nurse-Scheduling-Planes.

Anhang B

Tabellen zur globalen Optimierung

PEPSI DP 714	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
Ausgangsverletzung 1721,60 , Hillclimbing über Zuteilungen						
über Zuteilungen	1721,60	07:41	23940	0	0	22660
Ausgangsverletzung 1721,60 , Hillclimbing über Verletzungen						
über Verletzungen	1715,20	13:42	42503	0	2	20840
Ausgangsverletzung 2529.20 , Hillclimbing über Zuteilungen						
über Zuteilungen	1322,40	07:11	22988	1	2	9630
Ausgangsverletzung 2529.20 , Hillclimbing über Verletzungen						
über Verletzungen	1322,40	07:38	24829	1	2	9970

Tabelle B.1: Ergebnisse nach der globalen Optimierung des Dienstplanes 714 in PEPSI. Es wurde mit zwei unterschiedlich hohen Ausgangsverletzungen optimiert.

NURSE 1.1.2000	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
Ausgangsverletzung 4708.80 , Hillclimbing über Zuteilungen						
über Zuteilungen	1824,80	00:09	2599	0	1	0
Ausgangsverletzung 4708.80 , Hillclimbing über Verletzungen						
über Verletzungen	1804,80	00:08	2660	0	1	0
Ausgangsverletzung 13650.77 , Hillclimbing über Zuteilungen						
über Zuteilungen	995,40	00:29	4544	0	1	0
Ausgangsverletzung 13650.77 , Hillclimbing über Verletzungen						
über Verletzungen	4283,68	00:15	5125	0	1	0

Tabelle B.2: Ergebnisse nach der globalen Optimierung im Nurse-Scheduler. Es wurde mit zwei unterschiedlich hohen Ausgangsverletzungen optimiert.

Anhang C

Tabellen zur Zeitbegrenzung

PEPSI DP 714	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
Standard	18 Sekunden pro Verletzungsgewichtseinheit					
	1721,60	07:15	23940	0	2	22660
	60 Sekunden pro Korrektur					
	1721,60	03:29	11130	2	2	7030
	30 Sekunden pro Korrektur					
	1721,60	02:17	7090	3	2	2750
	15 Sekunden pro Korrektur					
	1721,60	01:32	4670	3	2	1140
Kombination 3	18 Sekunden pro Verletzungsgewichtseinheit					
	678,40*	19:32	49419	9	7	5093
	60 Sekunden pro Korrektur					
	678,40*	09:41	25755	9	7	1866
	30 Sekunden pro Korrektur					
	678,40*	05:08	14120	9	7	649
	15 Sekunden pro Korrektur					
	1049,60	02:55	8081	9	3	242

Tabelle C.1: Die Ergebnisse von Versuchsläufen mit dem Contract-Modell „Zeit pro Zuteilung“. Erstellt wurde der Dienstplan 714 mit dem PEPSI-System in den Konfigurationen „Standard“ und „Kombination 3“.

NURSE 1.1.2000	Bewertung	Laufzeit hh:mm:ss	#Zustände	#Zeitüber-schreitungen	max. Stufe Korrekturen	Zyklen
Variante 1	18 Sekunden pro Verletzungsgewichtseinheit					
	1467,80	44:59	743104	14	5	4392
	60 Sekunden pro Korrektur					
	1768,80	40:21	658511	17	5	3346
	30 Sekunden pro Korrektur					
	1018,40	27:11	442506	32	4	2149
	15 Sekunden pro Korrektur					
	1256,80	17:19	274221	42	4	795

Tabelle C.2: Die Ergebnisse von Versuchsläufen mit dem Contract-Modell „Zeit pro Zuteilung“. Erstellt wurde ein Dienstplan 30 Tage ab dem 1.1.2000. Gewählt wurde die Testkonfiguration „Variante 1“, da die Standardkonfiguration schon vor dem Ablauf des Zeitlimits terminiert.