Technische Universität Chemnitz–Zwickau
Fakultät für Informatik
Lehrstuhl Künstliche Intelligenz

Diplomarbeit

Verbesserung von Stundenplänen durch ein Multiagentensystem

eingereicht von:	Kay Anke
	geboren am 27.12.1971 in Leisnig
betreuender Hochschullehrer:	Prof. Dr. rer. nat. habil. W. Dilger
Betreuer:	Dr. R. Staudte

Chemnitz, den 16. Dezember 1996

Technische Universität Chemnitz–Zwickau
Fakultät für Informatik

Aufgabenstellung für die Diplomarbeit

Name, Vorname des Diplomanden:	A n k e, Kay
Immatrikulations–Nr.:	08088

Thema:

Verbesserung von Stundenplänen durch ein Multiagentensystem

Zielstellung:

Die typischen Beziehungen zwischen den beteiligten Objekten in einem Stundenplan lassen sich als Netz arithmetischer Constraints formulieren. Jede Lösung dieser Constraints ergibt einen Ausgangsplan, in welchem für gegebene Klassen, Lehrer, Räume und Zeiten sämtliche Zuordnungen zu den entsprechenden Lehrveranstaltungen getroffen wurden.
Ausgehend von einem solchen vollständig erstellten Plan ist in einer zweiten Phase dieser Plan bezüglich einiger ausgewählter Güteattribute (wie z.B. Minimierung der Anzahl von Freistunden für Lehrer und/oder Klassen, Minimierung von Raumwechsel für Klassen usw.) zu verbessern. Dazu sollte ein Multiagentenansatz, wie er etwa in [Dilg95] beschrieben wurde, dienen.
Für die erste Phase sind die in der vorangegangenen Projektarbeit erzielten Resultate zu nutzen. Die Ergebnisse sind anhand geeigneter Beispiele zu demonstrieren und mit ähnlichen aus der Literatur bekannten Arbeiten der Stundenplanung (z.B. [HeWü94]) zu vergleichen. Die Implementation in Oz ist zu dokumentieren.

Betreuender Hochschullehrer:	Prof. Dr. W. Dilger
Fakultät:	Informatik
Professur:	Künstliche Intelligenz
Betreuer:	Dr. R. Staudte
Beginn am:	17. Juni 1996
Eingereicht am:	16. Dezember 1996
Verteidigt am:	06. Februar 1997

Kurzreferat

Bibliographische Bezeichnung

Verbesserung von Stundenplänen durch ein Multiagentensystem

Kay Anke, 1996 — 92 S., 9 Abb., 16 Tab., 9 Alg., 13 Lit.

Technische Universität Chemnitz–Zwickau, Fakultät für Informatik, Diplomarbeit.

Stichworte: Constraintsystem, Multiagentensystem, Programmiersprache Oz, Simulated Annealing, Stundenplanerstellung, Optimierung

Zusammenfassung

Gegenstand der Arbeit ist, ausgehend von einem durch ein Constraintsystem erzeugten Stundenplan diesen hinsichtlich der wichtigen und praxisrelevanten Kriterien zu optimieren. Dazu wird ein Multiagentensystem implementiert, in dem Lehrer und Klassen als Agenten fungieren.

Für die Implementierung des Systemes wird die Programmiersprache Oz verwendet. Neben dem Problem der Stundenplanerstellung werden vorrangig Constraintprogrammierung und Multiagentensysteme beschrieben.
Das vorgestellte Constraintsystem dient weiterhin innerhalb des implementierten Multiagentensystemes als Problemlösungskomponente. Das Multiagentensystem zeichnet sich darüberhinaus hauptsächlich durch die Verhandlungen der Agenten über Planveränderungen aus.

Am Beispiel einer Mittelschule sind die Ergebnisse der Optimierung veranschaulicht. Die Stundenpläne ausgewählter Lehrer und Klassen vor und nach der Optimierung werden gegenübergestellt und diskutiert.

Zum besseren Verständnis der Problematik werden zwei weitere Stundenplanungssysteme vorgestellt. Das Programm Oz College Timetabling wird dabei ausführlicher behandelt. Es wurde auch in der Programmiersprache Oz implementiert, weist jedoch fundamentale Unterschiede zu dem hier entwickelten System auf. Diese erstrecken sich sowohl auf den Aufbau und die Arbeitsweise der beiden Programme als auch auf die Optimierung der Stundenpläne.

Inhalt

Abbildungen

Tabellen und Stundenpläne

Algorithmen und Programmausschnitte

Kapitel 1

Motivation und Ziel der Arbeit

Mit der Gründung des Forschungsbereiches Programmiersysteme am DFKI Saarbrücken begann 1990 die Entwicklung von Oz. Die KI–Sprache wurde u.a. mit dem Anspruch entworfen, Nachfolger von Prolog, Lisp und Smalltalk zu werden.
Im Januar 1994 wurde die erste offizielle Version der Programmiersprache DFKI Oz freigegeben. Am Lehrstuhl Künstliche Intelligenz der TU Chemnitz–Zwickau entstand der Wunsch, die neue Sprache auf ihre Tauglichkeit bezüglich des Lösens von Problemen aus dem Bereich der Künstlichen Intelligenz zu testen.

Als praxisrelevante, leicht zu verstehende, jedoch in ihrer Komplexität nichttriviale Aufgabe wurde das Problem der Planung von Stundenplänen als Testanwendung gewählt. Smolka und Müller schreiben zu diesem Problem [KI3/96]:

> Die Erstellung von Stundenplänen z.B. für Schulen ist ein schwieriges kombinatorisches Problem, das komplexe Constraints und anwendungsspezifische Suchstrategien erfordert. Mit Techniken des Operations Research (OR) und traditionellen Verfahren der Constraintlogikprogrammierung sind beide jedoch nicht auf hinreichend hoher Abstraktionsebene ausdrückbar. Insbesondere sind die OR–Methoden oft nicht ausreichend flexibel, und ihre Anpassung im Einzelfall zu teuer. Hier kann ein programmiersprachlicher Ansatz durch erhöhte Flexibilität von Vorteil sein.

Als naheliegender Lösungsansatz bietet sich ein Multiagentensystem an. Lehrer und Klassen einer Schule sind Partner, die gemeinsam einen guten Gesamtstundenplan anstreben. Durch Kommunikation verhandeln sie untereinander über Veränderungen des Planes. Bei jeder Veränderung werden sich Pläne einiger Partner verbessern, andere können sich verschlechtern. Letztere vertrauen jedoch auf kooperatives Verhalten der anderen, durch welches eine lokale Verschlechterung nur temporär auftreten sollte. Im Verlauf der Verhandlungen wird sich der Gesamtplan der Schule i.allg. verbessern.

In der Praxis werden die unterschiedlichen Rollen (Lehrer, Klassen, Räume) nur von wenigen Lehrern eingenommen. Die Erstellung des Stundenplanes dauert nach Angaben von Lehrern bei manueller Bearbeitung meist mehrere Tage (siehe hierzu Tabelle 3.1). Es bietet sich an, Lehrer, Klassen und eventuell auch Räume als Agenten anzusehen, die ihre Interessen selbständig in einem Multiagentensystem vertreten.

Nach einer allgemeinen Einarbeitungsphase in Oz wurden im November 1995 erste Gedanken zur Thematik auf dem Internationalen Workshop zur Oz–Programmierung in der Schweiz vorgestellt.
Die Grundidee bestand darin, durch ein Constraintsystem einen vollständig gesetzten Plan zu erzeugen, der durch Lehrer– und Klassen–Agenten verbessert wird. Die Ausarbeitung dieser Idee ist in [Dilg95] nachzulesen.
Der Prototyp eines solchen Constraintsystemes, welches einen vollständig gesetzten Stundenplan erzeugt, entstand im Rahmen einer Projektarbeit ([Anke96]).

Das Ziel dieser Diplomarbeit ist die Erstellung eines Systemes, mit dessen Hilfe Stundenpläne erstellt und verbessert werden können. Dazu sind Kriterien zu untersuchen, die für eine Optimierung in Frage kommen.
Für die Verbesserung eines Stundenplanes wird ein Multiagentensystem programmiert, in welchem Lehrer und Gruppen (Schüler einer Klasse mit gleichem Lehrplan) verhandelnde Agenten darstellen.
Basierend auf [Anke96] wird das Constraintsystem für die Erstellung eines vollständig gesetzten Stundenplanes überarbeitet und angepaßt.
Die Implementation des Prototyps erfolgt in DFKI Oz 2.0.

Die Ergebnisse werden abschließend anhand eines realen Stundenplanes einer Chemnitzer Mittelschule veranschaulicht.

Kapitel 2

Grundlagen der Arbeit

Im folgenden werden das Konzept der Problembeschreibung durch Constraints, Multiagentensysteme und der Algorithmus *Simulated Annealing* beschrieben. Desweiteren wird eine Einführung in die Programmiersprache Oz gegeben, in der das zu erstellende Planungssystem implementiert ist.
Das in Abschnitt 2.3 beschriebene Simulated Annealing wird innerhalb des Multiagentensystemes eingesetzt. Der Algorithmus bietet einen Optimierungsansatz, der u.a. zu besseren Testergebnissen führt.

2.1 Das Constraintkonzept

Für die Berechnung eines vollständig gesetzten Stundenplanes soll ein Constraintsystem eingesetzt werden. Während der Multiagentensystemphase wird dadurch auch die Konsistenz in den optimierten Plänen sichergestellt.
Für das weitere Verständnis der Problematik wird an dieser Stelle eine kurze Einführung in die Funktionsweise von Constraintsystemen gegeben.

2.1.1 Beziehungen zwischen Objekten

Im Alltag findet man überall Objekte (Gegenstände, Personen), die zwangsläufig miteinander in Beziehung treten. Wirkt ein Objekt auf ein anderes oder sich selbst ein, kann sich dieses verändern. Eine Veränderung kann nur dann eintreten, wenn auf ein Objekt eingewirkt wird. Ein von der Außenwelt abgeschlossenes Objekt kann durch diese nicht verändert werden.
Ohne diese Beziehungen zwischen Objekten gäbe es somit kein Ursache–Wirkung–Prinzip. Alle Objekte würden unabhängig voneinander und unveränderbar bestehen. Um die Wirkung, die ein Objekt auf andere hat, beschreiben zu können, muß man die geltenden Beziehungen beschreiben. Das setzt das Wissen um und über diese Beziehungen voraus.

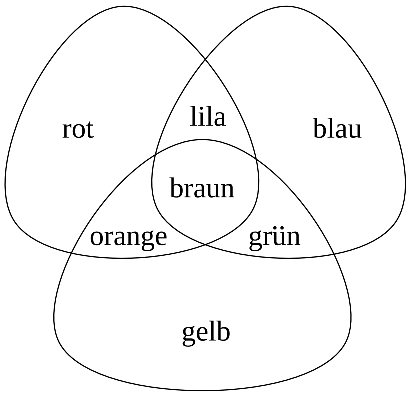

Abbildung 2.1: Das natürliche Mischverhalten von Farben

Constraints

Constraints dienen zur Repräsentation von Abhängigkeiten zwischen Objekten bzw. Eigenschaften von Objekten. Dazu werden Relationen beschrieben, die Beziehungen zwischen Variablen ausdrücken. Eine Relation beschreibt, in welcher Art und Weise sich die Eigenschaften von Objekten gegenseitig beeinflussen und verändern. Die Eigenschaften werden dabei mit Constraintvariablen beschrieben. Ihnen sind jeweils eine Menge zulässiger Werte (Wertebereich, Domäne) zugeordnet. Jeder einzelne Wert einer solchen Menge wird solange als potentiell möglicher Wert der Variable angesehen, bis die Nichterfüllbarkeit einer Relation bzgl. dieses Wertes nachgewiesen wurde und er aus der Domäne entfernt wird.

2.1 Definition (Constraint)
Ein Constraint C ist ein Paar $(\mathcal{V}, \mathcal{R})$. $\mathcal{V}$ ist eine n–elementige Menge von Variablen $\{V_1, \ldots V_n\}$ über den Wertebereichen $D_1, \ldots D_n$ und $\mathcal{R}$ eine entscheidbare Relation $\mathcal{R} \subseteq D_1 \times \cdots \times D_n$. □

2.2 Beispiel (Constraint)
Drei Behälter seien Objekte mit der Eigenschaft, Farbe zu enthalten (F_1, F_2, F_3). Es gelten weiterhin Beziehungen zwischen den Farbeigenschaften der Behälter: die Farben in den drei Behältern sind verschieden voneinander und das Mischen der Farben aus Behälter 1 und 2 ergibt die Farbe Grün (das Mischverhalten von Farben ist durch Abbildung 2.1 beschrieben). Die entsprechenden Constraints lassen sich wie folgt formulieren:

$$(\{F_1, F_2\}, F_1 \neq F_2) \tag{2.1}$$
$$(\{F_2, F_3\}, F_2 \neq F_3) \tag{2.2}$$
$$(\{F_1, F_3\}, F_3 \neq F_1) \tag{2.3}$$
$$(\{F_1, F_2\}, F_1 \oplus F_2 = grün) \tag{2.4}$$

□

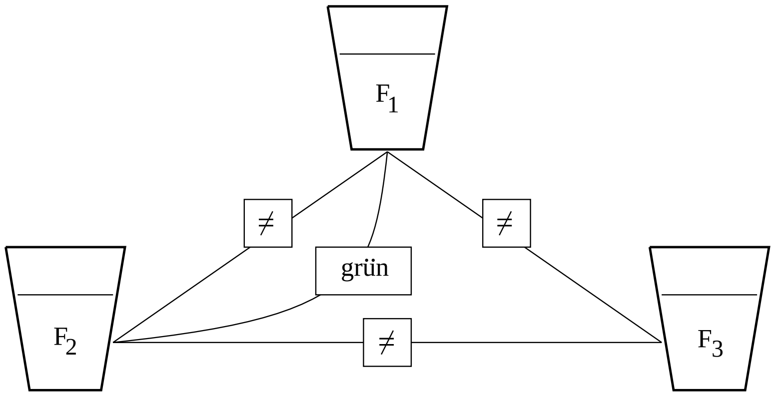

Abbildung 2.2: Constraintnetz zu Beispiel 2.4

Constraintnetz

Jedes Objekt kann durch Beziehungen, die über seine Eigenschaften gelten, mehrere andere Objekte beeinflussen, die wiederum mit weiteren Objekten verknüpft sein können. Ein solcher Verbund wird Constraintnetz genannt.
Dabei werden Variablen aus verschiedenen Constraints zu einer Variable des Constraintnetzes zusammengefaßt.

2.3 Definition (Constraintnetz)
Ein Constraintnetz CN ist ein Paar $(\mathcal{V}, \mathcal{C})$. $\mathcal{V}$ ist eine m–elementige Menge von Variablen $\{V_1, \ldots V_m\}$ und $\mathcal{C}$ eine k–elementige Menge von Constraints $\{C_1, \ldots C_k\}$. Die Variablenmengen der Constraints $C_1, \ldots C_k$ sind Teilmengen von $\mathcal{V}$. □

2.4 Beispiel (Constraintnetz)
Bezugnehmend auf Beispiel 2.2 kann folgendes Constraintnetz angegeben werden:

$$\begin{aligned}(\{F_1, F_2, F_3\}, \{(\{F_1, F_2\}, F_1 \neq F_2)\\ (\{F_2, F_3\}, F_2 \neq F_3)\\ (\{F_1, F_3\}, F_3 \neq F_1)\\ (\{F_1, F_2\}, F_1 \oplus F_2 = gr\ddot{u}n)\})\end{aligned}$$

□

Constraintproblem

Initialisiert man ein Constraintnetz mit Grundwertebereichen für die Constraintvariablen, so entsteht ein Constraintproblem.
Das Constraintnetz allein kann als Beschreibung eines Systems angesehen werden. Erst mit einer konkreten Aufgabenstellung (durch die Grundwertebereiche gegeben) wird das eigentliche Problem beschrieben. Beispiel 2.6 verdeutlicht das an zwei unterschiedlichen Grundwertebereichen für das Constraintnetz aus Beispiel 2.4.

2.5 Definition (Constraintproblem)
Ein Constraintproblem CP ist ein Paar $(init, CN)$. $init$ ist eine Funktion, die die initialisierenden Grundwertebereiche der Constraintvariablen des Constraintnetzes CN beschreibt. □

2.6 Beispiel (Constraintproblem)
Die Initialisierungsfunktion zu der in Beispiel 2.2 beschriebenen Problematik lautet

$$init(F_i) = \{blau, gelb, rot\} \quad (i = 1, 2, 3)$$

Ein anderes Problem kann durch

$$init(F_i) = \{blau, rot\} \quad (i = 1, 2, 3)$$

initialisiert sein, hätte jedoch keine Lösung, da die Farben in den drei Behältern nicht verschiedenen voneinander sein könnten. □

2.1.2 Propagierung

Die Auswertung der Constraintrelationen eines Constraintnetzes kann die Wertebereiche der Constraintvariablen dynamisch verändern. Dabei können aus einer Domäne nur Werte ausgeschlossen werden; man spricht von einer Einschränkung des Wertebereiches. Zu Beginn der Auswertung wird jede Variable mit ihrem Grundwertebereich initialisiert.

Einer Domäne kann kein Wert hinzugefügt werden. Eine Variable, deren Wertebereich leer ist, kann diesen somit nicht wieder „auffüllen"; ihr kann damit kein Wert zugewiesen werden. Jede Lösung eines Constraintproblemes benötigt für jede Constraintvariable einen nichtleeren Wertebereich. Ein Problem ist nicht lösbar, sobald nur eine einzige Variable keinen Wert (leere Domäne) besitzt.

Algorithmus 2.1 Einfache Propagierung

1. Initialisierung aller Constraintvariablen mit deren Grundwertebereich
2. Aktivierung aller Constraints
3. Wiederholung bis kein Constraint mehr aktiv ist → Ende der Propagierung; die Constraintvariablen können nicht weiter eingeschränkt werden
   3.1 Wahl eines aktiven Constraints C
   3.2 Neuberechnung und evtl. Einschränkung aller Constraintvariablen, die in C vorkommen
   3.3 ist der Wertebereich wenigstens einer Constraintvariablen leer, endet die Propagierung; das Constraintproblem ist nicht lösbar
   3.4 Aktivierung aller Constraints, die eine durch 3.2 eingeschränkte Variable enthalten
   3.5 Deaktivierung des Constraints C

Die Einschränkung der Wertebereiche wird Propagierung genannt. Ein einfacher Propagierungsalgorithmus ist in Algorithmus 2.1 dargestellt.

Nachdem alle Constraintvariablen mit ihrem Grundwertebereich initialisiert wurden, wird jede einzeln betrachtet. Existieren Werte in der Domäne der Variable, die eine Relation, an der die Variable beteiligt ist, nicht erfüllen, so werden diese aus der Domäne entfernt; der Wertebereich der Variable wird eingeschränkt. Wird er durch diese Einschränkung leer, endet die Propagierung. Es existiert dann keine Lösung für das durch die Constraints formulierte System mit den gegebenen problembeschreibenden Grundwertebereichen.

Nach der ersten Einschränkung jeder Constraintvariablen wird die Prozedur wiederholt. Dazu werden jedoch nur noch die Variablen erneut zur Betrachtung herangezogen, die über Constraintrelationen von Variablen abhängig sind, die im letzten Schritt eingeschränkt wurden. Dieser Vorgang wird solange wiederholt, bis keine Variable weiter eingeschränkt werden kann. Der Propagierungsalgorithmus endet hier.

2.7 Beispiel (Propagierung 1)
Bezugnehmend auf Beispiel 2.2 kann folgende Propagierung durchgeführt werden:

1. die Constraintvariablen werden mit ihren Grundwertebereichen initialisiert:

$$F_1 = F_2 = F_3 = \{blau, gelb, rot\}$$

2. Anwendung von Gleichung 2.4 und Einschränkung von F_1 und F_2:

$$\begin{aligned} F_1 \oplus F_2 &= grün \\ \{blau, gelb, rot\} \oplus \{blau, gelb, rot\} &= grün \\ \{blau, gelb\} \oplus \{blau, gelb, rot\} &= grün \\ \{blau, gelb\} \oplus \{blau, gelb\} &= grün \\ &\Rightarrow F_1 = \{blau, gelb\} \\ &\quad\ F_2 = \{blau, gelb\} \end{aligned}$$

□

Beispiel 2.7 zeigt deutlich, daß für die Lösung eines Problemes die Wahl der Constraints von großer Bedeutung ist. Nur von zwei Variablen konnten die Wertebereiche eingeschränkt werden. Der Bereich der dritten Variable, deren Wert eigentlich feststeht, konnte nicht verändert werden. Das liegt an der geringen Ausdruckskraft der Ungleichungen 2.1 bis 2.3, die lediglich unterschiedlichen Farben in den Behältern sicherstellen. Für die Beschreibung eines Problemes ist es wichtig, durch welche Constraints es beschrieben wird.
Beschreibt man die Farben in den drei Behältern noch durch

$$F_1 \oplus F_2 \oplus F_3 = braun, \tag{2.5}$$

so kann das Problem besser eingeschränkt werden (siehe Beispiel 2.8).

2.8 Beispiel (Propagierung 2)

- Anwendung von Gleichung 2.5 und Einschränkung von F_3:

$$\begin{aligned} F_1 \oplus F_2 \oplus F_3 &= braun \\ \{blau, gelb\} \oplus \{blau, gelb\} \oplus \{blau, gelb, rot\} &= braun \\ grün \oplus \{blau, gelb, rot\} &= braun \\ grün \oplus rot &= braun \\ &\Rightarrow F_3 = rot \end{aligned}$$

□

Domänen können nach Beendigung der Propagierung in Abhängigkeit von der Wahl der Constraints sowie der Grundwertebereiche der Constraintvariablen mehrere Werte enthalten. In diesem Fall kann nicht entschieden werden, ob das Problem überhaupt lösbar ist. Er tritt meist auf, wenn für ein Problem mehrere Lösungen existieren.

Eine Möglichkeit, diese Unzulänglichkeit zu beheben, bietet die Fallunterscheidung. Zu diesem Zweck wird das Problem in zwei getrennt zu betrachtende Probleme zerlegt. In beiden gelten die gegebenen Constraints und Einschränkungen weiter, jedoch wird dem einen Problem ein weiteres Constraint C und dem anderen dessen Negation ($\neg C$) hinzugefügt.
Sinnvoll ist es, wenn sowohl C als auch $\neg C$ eine Variable mit einem Wertebereich aus mehreren Werten einschränkt. Danach erfolgt für die beiden neuen Probleme eine weitere Propagierung. Innerhalb dieser Probleme kann eine Fallunterscheidung erneut notwendig werden, wenn sich noch keine Lösung (bzw. die Nichtlösbarkeit) des Problems ergeben hat.

2.9 Beispiel (Propagierung nach Fallunterscheidung)

- Bildung der Constraints C und $\neg C$; z.B.:

$$\begin{aligned} C: &\quad F_1 = blau \\ \neg C: &\quad F_1 \neq blau \end{aligned}$$

- Fall 1 (C gilt):

$$\begin{aligned} F_1 &\neq F_2 \\ blau &\neq \{blau, gelb\} \\ blau &\neq gelb \\ &\Rightarrow F_2 = gelb \end{aligned}$$

- Fall 2 ($\neg C$ gilt):

$$\begin{aligned} F_1 &\neq F_2 \\ gelb &\neq \{blau, gelb\} \\ gelb &\neq blau \\ &\Rightarrow F_2 = blau \end{aligned}$$

- Lösungen:

$$F_1 = blau, F_2 = gelb, F_3 = rot \tag{2.6}$$
$$F_1 = gelb, F_2 = blau, F_3 = rot \tag{2.7}$$

□

2.1.3 Relaxation

Bei der Aufstellung der Constraints eines Problemes muß man zwischen normalen und weichen (*soft*) Constraints unterscheiden. Normale Constraints gehören zur Problemformulierung und müssen deshalb unbedingt erfüllt werden. Weiche Constraints dienen hauptsächlich der Klassifizierung der Problemlösungen, sofern mehrere existieren.
D.h., normale Constraints beschreiben, welche Anforderungen an eine (beliebige) Lösung des Problemes gestellt werden. Mit weichen Constraints gibt man an, welche Anforderungen zusätzlich an eine optimale Lösung gestellt werden.

Existiert für ein Constraintnetz aus normalen und weichen Constraints keine Lösung, so nennt man es überbestimmt, wenn es ohne die weichen Constraints eine Lösung gibt. Deshalb ist es sinnvoll, bestimmte Constraints in einem Constraintnetz ein– bzw. ausblenden zu können. Dieser Vorgang wird Relaxation genannt (vgl. [Schw95]). Ein eingeblendetes Constraint ist in einer Lösung des Problemes erfüllt, ein ausgeblendetes jedoch nicht.

Bei der Aufweichung eines Constraintsystems ist jedoch folgendes zu beachten: Das Ausblenden von Constraints stellt eine Abschwächung der Problembeschreibung dar. Es existiert bei ausgeblendeten Constraints eine mindestens gleiche Anzahl von Lösungen des Problemes wie bei einer Problembeschreibung ohne Ausblendung. Es kann jedoch vorkommen, daß die Wertebereiche der Constraintvariablen durch Propagierung weniger eingeschränkt werden und der verbleibende Suchraum groß bleibt.

Jedem weichen Constraint kann z.B. ein weiteres Attribut (Relaxationswert) für die Wichtung hinzugefügt werden. Meist stehen dafür Werte aus dem Intervall [0, 1] zur Verfügung; 0 wird für Constraints verwendet, bei denen es egal ist, ob sie erfüllt sind oder nicht. Je näher der Relaxationsfaktor eines Constraints an 1 liegt, um so wichtiger ist es, das Constraint zu erfüllen.
Betrachtet man z.B. die Summe der Relaxationswerte der nichterfüllbaren weichen Constraints einer Lösung als Kosten der Problemlösung, so geben diese an, wie weit die Lösung von der Netzkonsistenz (alle Constraints sind erfüllt) entfernt ist. Eine zweite Lösung, deren Kosten geringer sind, stellt damit eine bessere Lösung des Problemes dar.

2.2 Multiagentensysteme

2.2.1 Agenten

Agenten können als reale Objekte (natürliche Agenten) bzw. als deren programmierte Vertreter (künstliche Agenten) betrachtet werden. Letztere sollen sich ähnlich ihren Vorbildern verhalten und deren Verhaltensmuster in Computersystemen abbilden. Diese Systeme werden Multiagentensysteme genannt. Natürliche Agenten (Individuen, Gegenstände) können Bestandteil von solchen Systemen sein[1].

Ein Agent zeichnet sich dadurch aus, daß er drei Aufgaben zu bewältigen hat, die in Konflikt zueinander stehen können:

1. Ein Agent besitzt individuelle Ziele, die er innerhalb der Gruppe so gut wie möglich durchsetzen möchte.

2. Die Gruppe, der der Agent angehört, besitzt globale Gruppenziele. Stehen diese in Konflikt zu den Zielen eines Agenten, muß dieser zwischen den eigenen Absichten und den Gruppenzielen dynamisch abwägen.

3. Ein Agent ist für die Lösung einer Aufgabe bereit, mit anderen Agenten der Gruppe zu verhandeln. Die Kommunikation dient dabei der Kooperation der Agenten, um gemeinsam eine Lösung zu erreichen, die sowohl hinsichtlich der Gruppen– als auch der individuellen Ziele als optimal angesehen werden kann.

2.10 Beispiel (Gefangenendilemma [Fisch93])
Zwei Straftäter sind in einem Gefängnis so untergebracht, daß sie sich nicht miteinander verständigen können. Beide Täter (Agenten) haben zwei Handlungsalternativen, sie können entweder ein Geständnis ablegen oder schweigen. Beide wissen, daß das Strafmaß auf ein Jahr festgelegt wird, wenn sie ein Geständnis ablegen und der andere schweigt; dieser erhält für sein Schweigen 4 Jahre Gefängnis. Legen beide ein Geständnis ab, so wird das Strafmaß auf 3 Jahre festgelegt. Schweigen hingegen beide, beträgt die Strafe jeweils zwei Jahre. Das Dilemma besteht darin, daß beide Täter ein Geständnis ablegen müssen, damit sie eine Chance auf das günstigste Strafmaß von einem Jahr erhalten und das schlimmste von 4 Jahren für sich ausschließen.
Das führt dazu, daß für beide das Strafmaß auf 3 Jahre festgelegt wird. Würden beide schweigen, wäre das Ziel der Strafminimierung besser erfüllt. □

Beispiel 2.10 verdeutlicht, daß eine sinnvolle Arbeit von Agenten untrennbar mit Kommunikation untereinander verbunden ist.

2.2.2 Einteilung von Multiagentensystemen

Zahlreiche Modelle sind auf dem Gebiet der Multiagentensysteme aus verschiedenen Aufgabenstellungen heraus entstanden. Dementsprechend wurden Versuche unternommen, eine Einteilung der Systeme vorzunehmen. Eine gebräuchliche Klassifizierung schlug Sridharan vor, die in [Brot94] um weitere Kriterien ergänzt wurde.

[1] im weiteren Verlauf werden nur künstliche Agenten betrachtet

Systemmodell	Individuum		Team	Gesellschaft
Agentenzahl	klein		mittel	groß
Interaktionsschema	einfach			komplex
Anpassungsfähigkeit	starr	programmierbar	lernfähig	autodidaktisch
Kontrollverteilung	kontrolliert		abhängig	unabhängig
Ressourcen	eingeschränkt			reichlich
Granularität	feinkörnig		mittel	grobkörnig
Problemlösung	Synthese			Zerlegung
Kooperativität	zuvorkommend		konkurrierend	antagonistisch
Kopplung	end			lose

Tabelle 2.1: Klassifikation von Multiagentensystemen

Tabelle 2.1 zeigt die Kriterien, nach denen Multiagentensysteme eingeteilt werden können.

Ausgehend von der Aufgabenstellung und der erforderlichen Spezialisierung der Agenten kann ein System aus (sehr) wenigen Agenten bestehen (Individuum, Team) oder aus vielen Agenten (Gesellschaft). Im letzteren Fall geht man meist von einer geringeren Spezialisierung der einzelnen Agenten aus. Die gestellten Aufgaben sind hier meist anspruchslose „Fleiß"–Arbeiten, die schnell erfüllt werden sollen, jedoch keine hohen Ansprüche an die Agenten stellen. Das Schnitzeljagdmodell ist dafür ein typischer Vertreter (siehe Seite 26).

Neben den Kriterien Systemmodell, Anzahl der Agenten und Interaktionsschema beeinflussen sich weitere Eigenschaften der Multiagentensysteme. Die Kriterien Anpassungsfähigkeit und Kontrollverteilung beschreiben z.B., wie autonom sich ein Agent verhalten kann.

Die Lösung eines Problemes ist meist eine Kombination aus Synthese und Zerlegung. Bei der reinen Synthese werden von den einzelnen Agenten Lösungen erarbeitet, die nicht vollständig sein müssen. Diese werden zu einer kompletten Problemlösung zusammengestellt. Bei Zerlegung wird ein gegebenes Problem in Unterprobleme aufgeteilt, die von Agenten gelöst werden sollen. Unterprobleme können dabei wieder unterteilt werden oder durch Synthese gelöst werden. Wie stark ein Problem zerlegt werden kann, wird durch die Granularität bestimmt.

Interaktionsschema, Kooperativität und Kopplung beschreiben die Art und den Ausprägungsgrad der Kommunikation unter den Agenten.
Sind sie darauf angewiesen, häufig und über sehr detaillierte Abläufe zu diskutieren, besteht eine enge Kopplung zwischen ihnen. In dem Fall sollte das Interaktionsschema eher komplex aufgebaut sein.
Wie kooperativ sich die Agenten verhalten, ist probemabhängig. Meist liegt die Kooperativität zwischen zuvorkommend und konkurrierend, wobei unvermutete Wechsel zwischen den Ausprägungen möglich und von der Erfüllung der individuellen Ziele abhängig sind.

2.2.3 Kooperationsmodelle

Viele Kooperationsmodelle von Multiagentensystemen entstanden aus realen oder fiktiven Aufgabenstellungen heraus. In [Dilg94] werden die bekanntesten vorgestellt. Systeme mit indirekter Kommunikation sind:

- Blackboard–Modell,
- Markt–Modell und
- Schnitzeljagd–Modell.

Diesen stehen Modelle mit direkter Kommunikation gegenüber:

- Contract–Net–Modell und
- Shared Plans.

Aus jeder Gruppe von Kooperationsmodellen wird im folgenden ein Vertreter näher beschrieben.

Schnitzeljagd–Modell

Auf einem topologisch unbekannten Gelände liegen verstreut Gesteinsproben. Die Lage der Proben ist unbekannt, man nimmt jedoch an, daß sie gehäuft auftreten. Ein Transportfahrzeug bewegt sich durch das Gelände und sendet ein Positionierungssignal aus. Eine Menge von primitiven Agenten kann sich im Empfangsbereich des Signales frei bewegen und Gesteinsproben einsammeln, die sie zufällig finden. Anhand des Signales kann das Transportfahrzeug aufgesucht werden, um eine gefundene Probe abzuliefern.
Da die Bewegung der Agenten zufällig ist, werden die Proben auch nur zufällig gefunden. Auf ein gehäuftes Vorkommen an einer Stelle wird keine Rücksicht genommen. Aus diesem Grund existiert eine recht einfache Kommunikation zwischen den Agenten. Finden sie eine Gesteinsprobe und bringen sie zum Transportfahrzeug zurück, legen sie auf ihrem Weg Papierschnitzel aus. Treffen Agenten während ihrer Suche nach Proben auf solche Schnitzel, bewegen sie sich entlang des markierten Weges zu einer potentiellen Fundstelle.
Werden keine Proben mehr gefunden, finden sich die Suchagenten beim Transportfahrzeug ein, um das Gelände zu verlassen.

Das System verhält sich äußerst robust und wirtschaftlich. Die Agenten besitzen kaum Wissen, können nicht lernen und folgen einem sehr einfachen Schema. Fällt ein Suchagent aus, hat das kaum Folgen auf die Lösung der Aufgabe. Es ist lediglich ein geringer Leistungsabfall zu bemerken. Nur wenn der Transportagent ausfällt, wird das System signifikant gestört.
Eine dynamische Veränderung der Umgebung stellt keine Probleme dar, da die Agenten über kein Weltbild verfügen und das Gelände bei jeder Bewegung erneut untersucht wird.

Die individuellen Vorzüge und das Gruppenziel sind gleich: Sammeln von Proben. Ein konkurrierendes Verhalten der Agenten untereinander wäre hier also hinderlich.

Das Besondere des Modells liegt in der Primitivität der Agenten und der Kommunikation, die das gestellte Problem sicher bewältigen. Für komplexere Aufgaben ist es jedoch nicht geeignet.

Die Kriterien des Modells können wie folgt beschrieben werden:

- Systemmodell: Gesellschaft (gleichartige Agenten ohne individuelle Spezialisierung)
- Agentenzahl: (sehr) groß
- Interaktionsschema: einfach
- Anpassungsfähigkeit: starr (aus wirtschaftlichen Gründen)
- Kontrollverteilung: voneinander unabhängige Suchagenten
- Ressourcen: reichlich (Gesteinsproben, Energie für Agenten, Suchzeit)
- Granularität: grobkörnig (keine Zerlegung in Unterprobleme)
- Problemlösung: Synthese (Sammeln der Proben und Zusammentragen)
- Kooperativität: zuvorkommend (alle arbeiten am gleichen Ziel)
- Kopplung: lose (das Auslegen von Schnitzeln erfolgt nicht für einen bestimmten Agenten)

Contract–Net–Modell

Eine schon vorhandene Menge von Wissensquellen (Prozeduren, Mengen von Produktionsregeln) ist verteilt und lose gekoppelt. Die Wissensquellen sind auf verschiedene Knoten (Agenten) verteilt, wobei mehrere in einem Knoten plaziert sein können. Eine Wissensquelle ist meist nicht in der Lage, eine Aufgabe allein zu lösen. Sie kann ein Teilproblem lösen oder zu einer solchen Lösung beitragen. Für die Verteilung der Teilprobleme und das Sammeln der Ergebnisse bedarf es einer komplexen Kommunikation. Die Wissensquellen als auch Agenten sollen wenig Zeit mit Kommunikation verbringen und sich mehr dem Lösen der ihnen zugeteilten Aufgaben widmen. In diesem Sinne sind sie lose gekoppelt.

Jeder Agent kann zwei Rollen ausführen: er kann als *Vertragsanbieter* (Manager) ein ihm gestelltes Problem in Teilprobleme zerlegen und andere Agenten mit deren Lösung beauftragen oder als *Vertagsnehmer* (Contractor) ein Problem annehmen, lösen und das Ergebnis an den Vertragsanbieter zurückliefern. Ist das Problem, welches ein Agent als Vertragsanbieter unterteilt und weiterreicht, auch nur ein Teilproblem, so spielt der Agent beide Rollen gleichzeitig.

Die Ausschreibung und Abwicklung eines Auftrages erfolgt in den im Algorithmus 2.2 angegebenen Phasen. Ist ein Agent in der Lage, ein Teilproblem selbständig zu lösen, unterteilt er es nicht weiter.
Die Agenten zeichnen sich durch das individuelle Ziel aus, eine hohe Arbeitsauslastung anzustreben. Deshalb können sie ihre Arbeitskraft auch gezielt anbieten und

so andere Agenten veranlassen, Arbeit aufzuteilen und damit zu einer kürzeren Bearbeitungszeit zu gelangen.
Gruppenziel sollte es sein, Aufträge günstig abzuwickeln (geringe Bearbeitungszeit, niedrige Kosten).

Auch bei diesem Modell stellt die Robustheit des Systems einen entscheidenden Vorteil dar. Fällt ein einzelner Knoten aus, können andere Wissensbasen seine Aufgaben übernehmen. Voraussetzung ist jedoch, daß die Funktionalität des ausgefallenen Knotens durch andere Knoten abgedeckt werden kann. Die Funktionalität kann dabei auch verteilt vorkommen.
Darüberhinaus ist das Modell auf Grund der Vielseitigkeit der Wissensquellen für die verschiedensten Anwendungen einsetzbar.

Als Nachteil ist der hohe Kommunikationsaufwand anzusehen. Um die Aufgaben detailliert zu beschreiben, ist ein umfangreiches Protokoll sowie eine große Anzahl von Nachrichten notwendig. Werden für einen wirtschaftlichen Betrieb des Systems verschiedene Aufgaben gleichzeitig bearbeitet, wird die Kommunikation noch komplexer, da ein Knoten an mehreren Aufträgen mitwirken kann.

Die Kriterien des Modells können wie folgt beschrieben werden:

- Systemmodell: Team (auf bestimmte Gebiete spezialisierte Agenten)
- Agentenzahl: mittel
- Interaktionsschema: komplex

Algorithmus 2.2 Einfache Auftragsabwicklung im Contract–Net–Modell

1. Agent A unterteilt ein ihm gestelltes Problem P in n Teilprobleme P_i mit $i = 1, \dots n$
2. Wiederholung für alle P_i:
   - 2.1. A ist Manager und gibt P_i bekannt
   - 2.2. Agenten, die sich in der Lage sehen, die Aufgabe zu erfüllen, geben ein Angebot ab. Es enthält z.B. benötigte Bearbeitungszeit, Kosten der Bearbeitung usw.
   - 2.3. A wertet die Angebote der potentiellen Contractor–Agenten aus und übergibt die Aufgabe an den günstigsten Anbieter A_i
   - 2.4. A_i bearbeitet die Aufgabe und gibt das Ergebnis an A zurück oder die Information, daß er die gestellte Aufgabe nicht lösen kann; im zweiten Fall erfolgt eine Neuausschreibung (2.1.)
3. A stellt die Ergebnisse aller P_i zur Lösung von P zusammen
4. A übermittelt die Lösung (respektive die Nichtlösbarkeit) an seinen Auftraggeber (anderer Agent, Benutzer). Konnte wenigstens ein P_i nicht gelöst werden, informiert A, daß er die gestellte Aufgabe wider Erwarten nicht erfüllen kann.

Message					
Header		Content			
Sender	Empfänger	Typ	Descriptor	Agent	Text

Tabelle 2.2: Syntaktischer Aufbau einer Nachricht

- Anpassungsfähigkeit: mindestens programmierbar
- Kontrollverteilung: voneinander unabhängige Wissensquellen
- Ressourcen: eingeschränkt (Anzahl der Knoten)
- Granularität: feinkörnig bis mittel (Zerlegeung nur, wenn wirklich nötig)
- Problemlösung: Zerlegung
- Kooperativität: konkurrierend (jeder Agent will eine hohe Arbeitsauslastung erreichen)
- Kopplung: lose

2.2.4 Kommunikationsprotokolle

Die Kommunikation zwischen Agenten unterliegt Regeln, die in einem Multiagentensystem eingehalten werden müssen. Das garantiert ein konfliktfreies Verstehen der Agenten untereinander. Ein Schema, nach dem der Aufbau von Nachrichten erfolgt, wird Protokoll genannt.

Ein allgemein gehaltenes und deshalb ausdrucksstarkes Protokollmodell stellen *Generisch Konfigurierbare Kooperationsprotokolle* dar:

Generisch Konfigurierbare Kooperationsprotokolle

Das kleinste Bestandteil einer Kommunikation ist die Nachricht. Sie besteht aus einem *Header*, der den sendenden und empfangenden Agenten der Nachricht enthält und einem Inhaltsteil. Dieser besteht aus:

- Nachrichtentyp (ein Wert aus: Inform, Query, Command, Offer)
- Descriptor (ein Wert aus: Verhalten, Ressource, Intention)
- Agent (Name des Agenten, auf den sich die Textangabe bezieht)
- Text (eigentlicher Inhalt der Nachricht)

Protokolle bestehen aus einer Folge von Nachrichten, die sich zwei Agenten abwechselnd zuschicken. Der Protokollname ist dabei der Typ der ersten Nachricht mit der Endung „*ing*". Will z.B. ein Agent einem anderen ein Angebot unterbreiten, nutzt er dafür das Protokoll Offering, welches mit einer Nachricht vom Typ Offer beginnt.

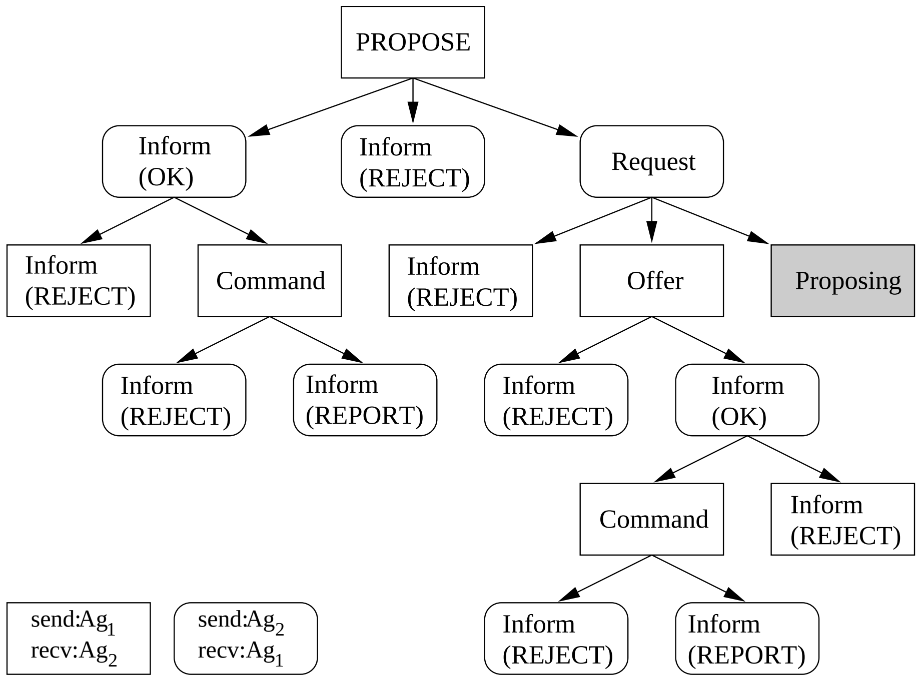

Abbildung 2.3: Protokoll PROPOSING

Auf eine erhaltene Nachricht kann ein Agent oft mit verschiedenen Nachrichten reagieren. Dies ist abhängig von seinem durch die Ziele spezifizierten Verhalten, welches u.a. durch die empfangenen Nachricht bestimmt wird. Die verschiedenen Nachrichten, die in einem Protokoll vorkommen können, werden in Form eines Baumes dargestellt. Die Nachrichten einer Ebene stellen die verschiedenen Antwortmöglichkeiten dar, die auf die Nachricht der darüberliegenden Ebene erfolgen können. Von Ebene zu Ebene wechseln sich die beiden Agenten als Sender und Empfänger ab.

Die vier Nachrichtentypen INFORM (Information), QUERY (Anfrage), COMMAND (Befehl) und OFFER (Angebot) können beliebig erweitert werden. Abbildung 2.3 zeigt stellvertretend für weitere das Protokoll PROPOSING (Vorschlag).

2.11 Beispiel (Protokoll PROPOSING**)**
Agent Ag_1 möchte Agent Ag_2 einen Vorschlag unterbreiten. Er sendet ihm die Nachricht vom Typ PROPOSE mit den entsprechenden Spezifikationen. Ag_2 kann

1. dem Vorschlag zustimmen (INFORM (OK)) und damit das Protokoll COMMANDING starten,

2. den Vorschlag ablehnen (INFORM (REJECT)) und damit die Kommunikation zu diesem Vorschlag abbrechen oder

3. einen Wunsch äußern (REQUEST) und damit das Protokoll REQUESTING starten; z.B. Anforderung genauer beschreiben, den Vorschlag hinsichtlich eines Parameters ändern usw.

Ist Ag_2 mit dem Vorschlag einverstanden, wird die erste Alternative gewählt. Will Ag_1 die Ausführung seines Vorschlages durch Ag_2, so weist er ihn an, den Vorschlag auszuführen. Hat er den Vorschlag weiteren Agenten unterbreitet und gewährt einem solchen den Zuschlag, so ist auch eine Ablehnung möglich. Im Falle eines Befehls an Ag_2 kann dieser seinerseits ablehnen (z.B. auf Grund von Veränderungen, die im Verlauf der bisherigen Kommunikation auftraten) oder das Ergebnis des Auftrages an Ag_1 übermitteln (INFORM (REPORT)).
Im Fall, daß Ag_2 zum Vorschlag eine Bitte äußert, kann Ag_1 die Kommunikation abbrechen, wenn er die Bitte nicht erfüllen kann bzw. will. Zum zweiten kann er das Protokoll (OFFERING) starten, wenn er aus seinem Vorschlag und der vorgetragenen Bitte ein Angebot ableiten kann. Ist dies nicht der Fall, kann er Agent Ag_2 einen neuen Vorschlag unterbreiten.
Das Angebot kann abgelehnt oder angenommen werden. Letzteres führt zur Auftragsanweisung durch das Protokoll COMMANDING. □

2.2.5 Agentenmodelle

Die verschiedenen Kooperationsmodelle bedingen meist einen bestimmten Agententyp. Die Art und Weise, wie sie sich verhalten, kann dennoch in einem Modell variieren. Hierzu wird Tabelle 2.1 ein weiteres Kriterium zur Beschreibung von Multiagentensystemen hinzugefügt:

Agententyp	reaktiv	hybrid	reflexiv

Reaktive Agenten

Reaktive Agenten ähneln Tieren, die auf bestimmte Reize nach einem bestimmten Verhaltensmuster reagieren. Da sie über kein Weltbild verfügen, sind sie auch nicht in der Lage, selbständig logische Schlüsse zu ziehen.

Ihre einfache Struktur erlaubt nur einfache Handlungen, jedoch sind diese nur wenig fehleranfällig und für Anwendungen, die auf ein abgegrenztes Problem ausgerichtet sind, ausreichend.

Reflexive Agenten

Reflexive Agenten sind in ihrer Handlungsweise am Menschen orientiert. Man kann sagen, sie verhalten sich intelligenter als reaktive Agenten.

Sie verfügen über internes Wissen, welches ihre Umgebung beschreibt. Damit verfügen sie über ein eingeschränktes, evtl. problemübergreifendes Weltbild. Auf Grund von logischen Schlüssen können reflexive Agenten ihre Sichtweise der Umwelt anpassen.
Desweiteren verfügen reflexive Agenten über lang- und kurzfristige Ziele. Diese werden Intentionen genannt. Mit der Zeit können sie sich ändern und bestimmen damit maßgeblich das Verhalten eines Agenten. Zieländerungen treten vor allem auf, wenn

ein Agent Schlüsse bezüglich seiner Umwelt (eigenes Wissen und empfangene Nachrichten) zieht.

Hybride Agenten

Hybride Agenten besitzen sowohl reaktives als auch reflexives Verhalten. Die meisten Agenten sind hybrid und nutzen somit die Vorteile beider Modelle. Abhängig von der Aufgabenstellung schwankt der Ausprägungsgrad zwischen den beiden Kriterien.

Sinnvoll ist es, wenn Agenten auf unbekannte Situationen reaktiv handeln und in bekanntem Terrain auf der Grundlage ihres Weltbildes.

2.12 Beispiel (Hybrides Agentenverhalten)
Ein Transportagent bewegt sich in den Gängen eines Lagers, dessen Regalanordnung ihm bekannt ist. Stößt er während seiner Fahrt durch die Regalstraßen auf einen anderen Transportagenten (bekanntes Hindernis), so kommt zu seinen kurzfristigen Zielen hinzu, dem Hindernis auszuweichen. Auf Grund seines Wissens über die Lage der Regalstraßen und der Nachricht des anderen Agenten, daß dieser nach rechts ausweicht, kann er eine neue optimale Fahrstrecke berechnen.
Wird der Agent durch zwei unbekannte Hindernisse, mit denen er nicht kommunizieren kann, in einer Regalstraße eingesperrt, reagiert er mit Stehenbleiben und Warten. Obwohl sein Ziel darin besteht, eine Ware an einen bestimmten Ort zu transportieren, fährt er nicht permanent zwischen den Hindernissen hin und her. □

2.3 Simulated Annealing

Simulated Annealing (simuliertes Abkühlen) ist ein Algorithmus zur Lösung komplexer Optimierungsprobleme ([KMMP91]). In Anlehnung an die Züchtung von Kristallen versucht er, ein globales Minimum an Kosten (hier: Energie) zu finden.

Es existiert eine Energiefunktion, die für die Lösungen eines Problemes die Energiezustände ermittelt. Eine Lösung mit geringer Energie ist besser als eine mit höherer Energie.
Vergleicht man die Energiezustände zweier Lösungen, die sich nur geringfügig unterscheiden, kann wenigstens ein Energiezustand als Minimum angesehen werden. Benachbarte Lösungen unterscheiden sich idealerweise nur in einer der n Variablen, die das Problem enthält. Jede Lösung liegt in einem $(n+1)$–dimensionalen Raum, auf dessen $(n+1)$. Dimension die Energiezustände der Lösungen abgebildet sind. Indem man von einer Lösung ausgehend die Energiezustände der jeweils benachbarten Lösungen im Raum untersucht, kann man durch ein $(n+1)$–dimensionales Energiegebirge „wandern".

Auf der Suche nach einem lokalen Minimum beginnt man mit einer beliebigen Lösung und bestimmt deren Energie. Danach sucht man eine benachbarte Lösung, deren Energiezustand kleiner ist. Von dieser Lösung ausgehend wiederholt man die Suche. Weisen sämtliche Nachbarlösungen einen jeweils größeren Energiezustand auf, stellt die aktuelle Lösung ein lokales Minimum dar. I.allg. ist sie jedoch kein globales Minimum. Durch sukzessive Abwärtsbewegung im Energiegebirge findet sich bald ein solches lokales Minimum.

Ein solches Minimum stellt nicht zwangsläufig ein globales Minimum dar. Auf der Suche nach einer optimalen Lösung befindet man sich in einer Sackgasse.
Um diese wieder verlassen zu können, wird ein Wert namens *Temperatur* eingeführt. Er gibt an, wieviel Energie dem System zugeführt werden kann. D.h., auf dem Weg zu einer benachbarten Lösung kann eine solche einen höheren Energiezustand als die aktuelle Lösung besitzen. Die Differenz der beiden Energiezustände darf den Wert der Temperatur jedoch nicht übersteigen. Dadurch können lokale Minima wieder

Algorithmus 2.3 Simulated Annealing

1. suche beliebige Lösung L_{akt} mit den Kosten $E(L_{\text{akt}})$
2. Menge der betrachteten Lösungen $\mathcal{L} = L_{\text{akt}}$
3. Wiederholung bis $\mathcal{N}$ leer
   3.1. $\mathcal{N}$ ist Menge der benachbarten Lösungen zu L_{akt} mit
   $\forall\ N_i \in \mathcal{N} \quad E(N_i) \leq E(L_{\text{akt}}) + T$
   3.2. $L_{\text{akt}} = L_{\text{next}}$, $\mathcal{L} = \mathcal{L} \cup L_{\text{next}}$ mit $L_{\text{next}} \in \mathcal{N}$
   3.3. verringere die Temperatur T (z.B. linear)
4. $L_{\text{min}} \in \mathcal{L}$ mit $\forall\ L_i \in \mathcal{L} \quad E(L_{\text{min}}) \leq E(L_i)$ ist (globales) Minimum

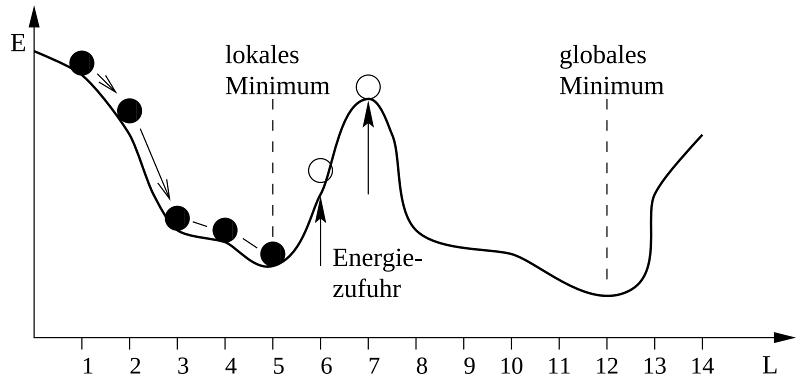

Abbildung 2.4: Zweidimensionales Energiegebirge

verlassen werden.

Die Temperatur nimmt jedoch mit fortschreitender Zeit ab, so daß es immer schwerer wird, ein lokales Minimum zu verlassen. Spätestens, wenn die Temperatur bei Null liegt, kann man ein gefundenes Minimum nicht mehr verlassen. Das kleinste gefundene Minimum entspricht idealerweise dem globalen Minimum.

Abbildung 2.4 zeigt ein zweidimensionales Energiegebirge mit zwei Minima. Beginnend bei Lösung L_1 wird das lokale Minimum bei L_5 schnell gefunden. Ist der Temperaturwert groß genug, um über L_6 zu L_7 zu gelangen, kann das zweite Minimum gefunden werden. Reicht die Temperatur nicht aus, dem System die benötigte Energie zuzuführen, um das lokale Minimum zu verlassen, muß L_5 zwangsläufig als globales Minimum angesehen werden.

Kristallzüchtung ([SHG90]):

Um künstliche Kristalle mit wenigen Fehlern und somit perfekter Struktur herzustellen, müssen diese sehr langsam abgekühlt werden. Die Atome haben dadurch genügend Zeit, um die Gitterplätze zu finden, bei denen eine minimale Gesamtenergie besteht. Ist die Temperatur groß genug, können einzelne Atome ihre Position ändern, so daß die Gesamtenergie des Systems zunimmt. Lokale Minima können so wieder verlassen werden. Die Wahrscheinlichkeit dafür verringert sich jedoch mit sinkender Temperatur.

2.4 Die Programmiersprache Oz

2.4.1 Überblick

Dieses Kapitel behandelt in Anlehnung an [KI3/96] die Programmiersprache Oz (in der StuPlan implementiert wurde) und das ihr zugrundeliegende Programmiermodell. Oz ist eine nebenläufige Programmiersprache, die Constraintprogrammierung mit funktionaler und objektorientierter Programmierung verbindet und als Nachfolger von Hochsprachen wie Lisp, Prolog und Smalltalk entwickelt wurde.

Die Entwicklung von Oz begann 1990 mit der Gründung des Forschungsbereiches Programmiersysteme am DFKI Saarbrücken. Ein erster Prototyp der Sprache war im September 1992 fertig. Das erste Oz–System (DFKI Oz 1.0) wurde im Januar 1994 freigegeben; 1997 folgt die erweiterte und verbesserte Version DFKI Oz 2.0. Die weiteren Ausführungen beziehen sich auf diese Version.

2.4.2 Das Oz Programmiermodell

Das Berechnungsmodell

Im Programmiermodell von Oz finden Berechnungen in sogenannten Berechnungsräumen statt. Ein solcher Berechnungsraum enthält Berechnungseinheiten, die über einen gemeinsamen (abstrakten) Speicher miteinander kommunizieren. Diese Einheiten werden Aktoren genannt.
Als Berechnung wird die Reduktion von Aktoren bezeichnet. Existieren mehrere Aktoren in einem Berechnungsraum, so werden diese nacheinander und fair reduziert. Dabei kann einerseits der Speicher verändert werden, andererseits können neue Aktoren entstehen. Der reduzierte Aktor wird aus dem Berechnungsraum entfernt.

Das Programmfragment

```
local X Y in
   Y=if X==0 then error else X end  ...  X=3
end
```

installiert als erstes die gesamte `if`–Anweisung im aktuellen Berechnungsraum. Danach können weitere Aktoren aufgenommen werden. Da die Berechnungen fair voranschreiten, wird die `if`–Anweisung früher oder später reduziert. Dabei entstehen die neuen Aktoren

$$\texttt{if X==0 then Y=error end} \tag{2.8}$$

$$\texttt{if X\textbackslash=0 then Y=X end} \tag{2.9}$$

Der Aktor für die erste `if`–Anweisung wird gelöscht. Nachdem X an den Wert 3 gebunden wurde, erzeugt Aktor 2.9 den neuen Aktor $Y = X$, der dann wiederum den Aktor $Y = 3$ im Berechnungsraum installiert. Wird dieser reduziert, verändert sich der Speicher; der Variable Y wird der Wert 3 zugewiesen. Aktor 2.8 wird ohne weitere Installation von Aktoren oder Speicherveränderung aus dem Berechnungsraum entfernt.

Constraints

Constraints sind in Oz logische Formeln, die die Werte von Variablen beschreiben. Eine Variable ist entweder (noch) ungebunden oder an einen beliebigen Wert gebunden. Im zweiten Fall kann sie keinen neuen Wert annehmen.
Elementare Constraints sind Gleichungen zwischen Variablen[2] oder zwischen Variablen und Strukturen (Zahlen, Tupel, Verbunde) wie:

```
X=38 Y=X
Z=mitarbeiter(name:'Meyer, Peter' alter:X)
```

Darüberhinaus besitzt Oz Constraints über endliche Bereiche ganzer Zahlen (finite domains). Die Anweisung

```
X::1#10
```

beschränkt die Variable `X` auf ganze Zahlen zwischen 1 und 10.

Konditionale

Konditionale wie

```
case {Member 2 Xs} then X=yes else X=no end
```

bestehen aus einem Bedingungsteil (guard) und jeweils einem Anweisungsteil für den Erfüllungs– bzw. Nichterfüllungsfall der Bedingung; die Angabe des `else`–Anweisungsteiles kann entfallen, wenn die Bedingung immer erfüllt wird. Solange keine Aussage über die Erfüllbarkeit der Bedingung getroffen werden kann, wartet das Konditional mit der Ausführung eines seiner Zweige.
Die Erfüllung von Bedingungen im Speicher wird durch logische Implikation (entailment) modelliert.

Prozeduren

Prozedurale Programmierung wird von Oz voll unterstützt. Prozeduren können an Variablen gebunden werden, so daß die Argumente (Ein– und Rückgabewerte) einer Prozedur wiederum Prozeduren sein können.

Funktionen sind Spezialfälle von Prozeduren, bei denen der Aufruf der Funktion den Rückgabewert realisiert. Das entsprechende Argument wird dabei nicht in der Argumentenliste aufgeführt. Diese syntaktisch vereinfachte Schreibweise ist besonders dann zu empfehlen, wenn eine Prozedur genau einen Rückgabewert besitzt.

Die Funktion

```
fun {FilterMap Xs FilterF MapF}
   case Xs
   of nil then nil
   [] X|Xr then
      case {FilterF X}
      then {MapF X}|{FilterMap Xr FilterF MapF}
      else {FilterMap Xr FilterF MapF}
      end
   end
end
```

[2] Variablen werden in Oz durch Bezeichner definiert, die mit einem Großbuchstaben beginnen

durchsucht eine Liste `Xs` und filtert die Elemente heraus, die die Funktion `FilterF` erfüllen. Auf diese Elemente wird die Funktion `MapF` angewandt. Die Anweisung

```
Ys={FilterMap [1 2 3 4 5] Odd fun {$ X} X+2 end}
```

schränkt `Ys` zu `[3 7 9]` ein. (Die ungeraden Zahlen der Eingabeliste werden um 2 erhöht.)
Als Filter dient die Standardfunktion `Odd`; die Abbildungsfunktion wird durch eine lokale Funktion beschrieben, deren Name durch die anonyme Variable `$` dargestellt wird.

Objekte

Objekte sind abstrakte Datenstrukturen mit synchronisiertem Zustand. Dieser wird durch Attribute gekennzeichnet, die ihren Wert verändern können. Durch die Methoden eines Objektes kann dessen Zustand verändert werden.
Die Ausführung einer Methode wird durch das Senden einer Nachricht realisiert. Eintreffende Nachrichten werden vom Objekt serialisiert, so daß zu jedem Zeitpunkt maximal eine Methode Zugriff auf den Zustand des Objektes besitzt.

Objekte können in Oz durch Prozeduren (Methoden) und Zellen (Attribute) definiert werden. Zellen[3] sind Referenzen auf Datenstrukturen, die umgesetzt werden können. Der Vorgang des Umsetzens ist eine atomare Operation, d.h., der Inhalt einer Zelle wird gelesen und durch einen neuen Wert ersetzt.

Klassenhierarchien können durch Mehrfachvererbung aufgebaut werden, d.h., eine Klasse kann mehrere Elternklassen besitzen. Die Klasse `UrObject` definiert die grundlegende Funktionalität jeder weiteren Klasse. Alle anderen sind direkt oder indirekt von ihr abgeleitet.

2.4.3 Constraintprogrammierung

Propagierer

Neben den elementaren existieren weiterhin komplexe Constraints, die durch sogenannte Propagierer ausgedrückt werden. Arithmetische Gleichungen und Ungleichungen werden z.B. durch Propagierer dargestellt:

```
X >: Y
X + Y =: 3
X*X + Y =: 5
```

Propagierer können durch Prozeduren implementiert werden. Ein Propagierer kann oft durch verschiedene Implementationen beschrieben werden. Deren operationale Verhalten können jedoch zu unterschiedlichen Komplexitäten des Problemes führen. Komplexe Probleme lassen sich häufig erst durch eine geeignete Implementierung der Propagierer lösen.

Propagierung ist ein deterministisches, nichtsuchendes Verfahren. Nach Abschluß der Propagierung ist jedoch nicht garantiert, daß eine Lösung bzw. die Nichtlösbarkeit des Problemes erkannt wurde.

[3] Zellen werden in Sprachen wie Pascal oder C als Pointer bezeichnet

Distribuierung

Wird ein Problem P durch Propagierung nicht vollständig gelöst (inklusive Erkennen der Nichtlösbarkeit), muß es distribuiert werden. Es wird eine Fallunterscheidung vorgenommen, bei der zwei Teilprobleme P_1 und P_2 unter Zuhilfenahme eines neuen Constraints C folgendermaßen generiert werden:

$$\begin{aligned} P = P_1 \vee P_2 &= P \wedge 1 \\ &= P \wedge (C \vee \neg C) \\ &= (P \wedge C) \vee (P \wedge \neg C) \\ &\Rightarrow \quad P_1 = P \wedge C \\ &\Rightarrow \quad P_2 = P \wedge \neg C \end{aligned}$$

C sollte dabei so gewählt werden, daß in den Problemen P_1 und P_2 wieder propagiert werden kann. Um die Teilprobleme unabhängig voneinander bearbeiten zu können, wird für jedes ein eigener Berechnungsraum erzeugt.

Durch die Aufteilung eines durch Propagierung nicht vollständig lösbaren Problemes in zwei Teilprobleme (und deren eventuelle weitere Aufteilung) baut sich ein Suchbaum auf. Dieser kann eine, mehrere oder keine Lösung enthalten; ein Problem mit mehreren Lösungen muß auf jeden Fall distribuiert werden.
Für die Suche nach einer Lösung im Suchbaum können verschiedene Strategien angewandt werden. Neben vordefinierten wie Tiefensuche (nach erster oder allen Lösungen) oder lösungsoptimierender Suche (Branch–and–Bound) können eigene Algorithmen implementiert werden.

2.13 Beispiel (Distribuierung)
Das Problem P schränkt die Variable `X` durch Propagierung auf $X \in \{1, 2, 3\}$ ein. C könnte durch $X = 1$ und $\neg C$ durch $X \neq 1$ beschrieben werden. In P_1 gilt dann $X \in \{1\}$, in P_2 dagegen $X \in \{2, 3\}$. Beide Probleme werden getrennt voneinander nocheinmal propagiert (siehe Abbildung 2.5). □

Reified Constraints

Das in Kapitel 2.1.3 beschriebene Ein- und Ausblenden von Constraints wird von Oz unterstützt. Dafür stehen sogenannte *reified*[4] Constraints zur Verfügung.

Jedes Constraint C ist an eine Constraintvariable B gekoppelt, die nur die boolschen Werte 0 und 1 annehmen kann.
Ist C im Berechnungsraum enthalten, d.h., es kann erfüllt werden, wird B an den Wert 1 gebunden. C ist im Constraintnetz eingeblendet. Im anderen Fall gilt $B = 0$, das Constraint ist ausgeblendet. Das bedeutet, daß C im Berechnungsraum nicht erfüllt werden kann. Es gilt somit $\neg C$.
Umgekehrt kann aber auch B auf einen Wert festgelegt werden. Dieser bestimmt dann, ob C oder $\neg C$ im Constraintnetz erfüllt werden muß.

[4] Da keine geeignete Übersetzung gefunden wurde, wird hier der englische Begriff verwendet.

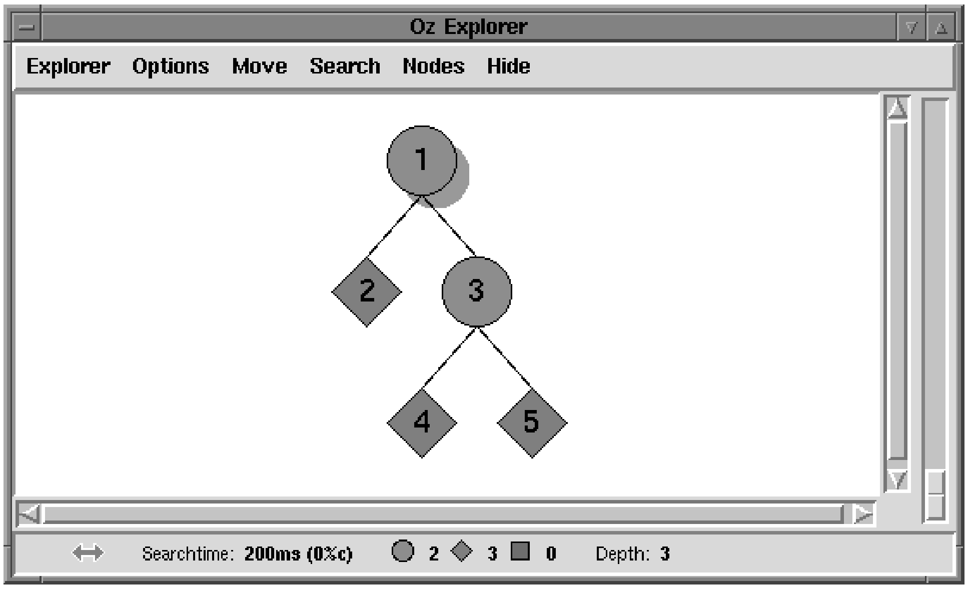

Abbildung 2.5: durch Distribuierung erzeugter Suchbaum zum Problem aus Beispiel 2.13 (Knoten 1: $X \in \{1, 2, 3\}$, Knoten 2: $X = 1$, Knoten 3: $X \in \{2, 3\}$, Knoten 4: $X = 2$, Knoten 5: $X = 3$)

2.4.4 Nebenläufigkeit

Nebenläufigkeit und Parallelität

Das Programmiermodell von Oz ist nebenläufig, jedoch nicht parallel. D.h., Berechnungen können unabhängig voneinander voranschreiten, über den gemeinsamen Speicher miteinander kommunizieren und sich synchronisieren.
Parallelität besagt dagegen, daß sich Operationen der Hardware zeitlich überlagern.

Reduktion von Aktoren

In einem Berechnungsraum sind meist mehrere Aktoren gleichzeitig in der Lage, reduziert zu werden. Um hohe Fairness, Reaktivität und Effizienz bei der Reduktion der Aktoren zu erzielen, ist ein geeigneter Reduktionsalgorithmus notwendig.

Oz greift deshalb auf sequentielle Komposition von Aktoren zurück.

```
<Ausdruck1> <Ausdruck2>
```

legt fest, daß `Ausdruck1` vor `Ausdruck2` reduziert wird. Sequentielle Komposition gruppiert Aktoren zu sogenannten sequentiellen Threads.
Um die beiden Ausdrücke nebenläufig zu reduzieren, muß `Ausdruck1` explizit als eigenständiger Thread deklariert werden:

```
thread <Ausdruck1> end <Ausdruck2>
```

2.4.5 DFKI Oz

DFKI Oz ist eine interaktive und leistungsfähige Implementierung von Oz. Die Programmierumgebung basiert auf GNU Emacs. Das System bietet weiterhin:

- einen nebenläufigen Browser,
- ein objektorientiertes Graphiksystem basierend auf Tcl/Tk,
- Interoperabilitätsmechanismen (Sockets, C, C++),
- automatische Speicherbereinigung (garbage collection),
- ein interaktives, graphisches Werkzeug zur Analyse von Suchproblemen (Oz Explorer, siehe Abbildung 2.5),
- einen inkrementellen Übersetzer sowie
- Unterstützung für stand–alone–Anwendungen.

Kapitel 3

Erstellung von Stundenplänen

3.1 Das Stundenplanungsproblem

3.1.1 Einführung

Das Problem der Stundenplanerstellung ist auf Grund seiner Individualität und Komplexität schwer zu lösen. Um den Lehrern diese Arbeit zu vereinfachen, existieren einige Softwareprodukte unterschiedlicher Qualität. Trotzdem ergab eine Umfrage[1] in Grund– und Mittelschulen sowie Gymnasien, daß viele Schulen weiterhin ihre Pläne manuell gestalten. Bei kleineren Grundschulen stellt das kein so gravierendes Problem dar wie bei Gymnasien, deren Schüler der 11. und 12. Klasse auf Grund des Kurssystems jeweils einen eigenen Stundenplan besitzen. Tabelle 3.1 soll verdeutlichen, welcher zeitliche Aufwand bei der manuellen Planung (in Abhängigkeit von der Anzahl der zu planenden Stunden) existiert.

Im Durchschnitt kann ein Lehrer, der an der Aufstellung des Stundenplanes beteiligt ist, in einer Stunde acht Unterrichtseinheiten planen. Das bedeutet, daß auf der einen Seite mit Hilfe von Software ein großes Potential an Arbeitszeit eingespart werden kann; andererseits ist es aber auch ein Indiz dafür, daß das Erzeugen von optimalen Plänen weder manuell noch rechentechnisch trivial ist.
Die Abhängigkeiten für die Stundenplanung sind somit so genau wie möglich zu erfassen. Desweiteren ist eine Optimierungsstrategie zu entwickeln, die fähig ist, gute Ergebnisse zu erzeugen.

3.1.2 Begriffe

Ab diesem Kapitel werden einige Begriffe benutzt, die an die Problematik gebunden sind, jedoch verschieden interpretiert werden können. Aus diesem Grund besteht die Notwendigkeit, ihre Bedeutung kurz zu erläutern.

[1] Die Umfrage fand im Oktober 1996 an ca. 80 Schulen der Chemnitzer Region statt.

	Grundschulen			Mittelschulen			Gymnasien		
	min	⊘	max	min	⊘	max	min	⊘	max
UE	88	199,1	299	270	411,8	396	1088	1074,8	1764
T (h)	20	23,1	14	84	52,8	24	260	146,7	120
UE/T	4,4	8,6	21,4	3,2	7,8	16,5	4,2	7,3	14,7

Tabelle 3.1: Planungszeiten bei manueller Stundenplanerstellung (UE: Anzahl aller Unterrichtseinheiten einer Schule, T: Zeiterfordernis für das manuelle Setzen der Einheiten, UE/T: Verhältnis zwischen der Anzahl der Unterrichtseinheiten und der dafür benötigten Planungszeit)

Zeiteinheit, Zeit

Stundenpläne bestehen aus mehreren Wochentagen, an denen jeweils eine bestimmte Anzahl von Unterrichtsstunden stattfinden kann. Diese Stunden sind numeriert und jeder zu planenden Unterrichtseinheit wird genau eine zugewiesen. Da sie selten genau 60 Minuten andauern, wird eine solche Stunde hier Zeiteinheit (kurz: Zeit) genannt.
Jedes Objekt (Lehrer, Gruppe, Raum) verfügt über einen Plan, der aus n Zeiteinheiten besteht. n ist das Produkt aus der Anzahl der Tage und der maximalen Anzahl von Zeiteinheiten eines Tages. Jedes Objekt kann Zeiteinheiten sperren, so daß keiner Unterrichtseinheit, an der das Objekt beteiligt ist, eine solche Zeit zugewiesen werden kann. Zeiteinheiten, die global — also von allen Objekten — gesperrt sind, belegen auch eine ganze Zahl aus dem Intervall $[1, n]$ (siehe Beispiel 3.1).

3.1 Beispiel (Zeiteinheiten)
In einer Schule findet von Montag bis Sonnabend Unterricht an maximal 6 Stunden statt; es existieren somit 36 Zeiteinheiten. Die 1. Einheit montags und die beiden letzten am Sonnabend stehen jedoch nicht zur Verfügung.
Nebenstehendes Schema zeigt die Zeiteinheiten der Schule und deren Numerierung. (Jedes einzelne Objekt kann für sich weitere Einschränkungen vornehmen.) □

	Mo	Di	Mi	Do	Fr	Sa
I		7	13	19	25	31
II	2	8	14	20	26	32
III	3	9	15	21	27	33
IV	4	10	16	22	28	34
V	5	11	17	23	29	
VI	6	12	18	24	30	

Unterrichtseinheit, Einheit

Unterrichtseinheiten sind Objekte, denen eine Zeit und ein Raum zugewiesen wird. Jede Einheit wird durch einen Namen, je eine Gruppen–, Lehrer–, Raum– und Zeitmenge definiert. Darüberhinaus können weitere Eigenschaften das Verhältnis zu anderen Unterrichtseinheiten beschreiben.

Ein Stundenplanungssystem besteht aus endlich vielen Einheiten und ist gelöst, wenn jeder eine Zeit und ein Raum zugewiesen ist. Durch Veränderung der Zeit bzw. des Raums einer Einheit kann der Gesamtplan in Abhängigkeit von den Gütekriterien optimiert werden.

3.2 Beispiel (Unterrichtseinheit)
Das vorliegende Datenobjekt beschreibt die zweite (von fünf) Deutschstunden. Aus zwölf zur Verfügung stehenden Räumen wurde der Raum mit der Identifikationsnummer 4 ausgewählt; von den 50 möglichen Zeiteinheiten wurde die 14. geplant (bei maximal 10 Zeiteinheiten pro Tag ist das die 4. Stunde am zweiten Tag).

```
o(all:5                 % Anzahl der Einheiten der Veranstaltung
  groups:[1 2]          % Menge der beteiligten Gruppen (Gruppen–IDs)
  id:3                  % eindeutige ID der Einheit
  name:2                % Name der Einheit (Fach–ID)
  number:2              % lfd. Nummer innerhalb der Veranstaltung
                        % speziellere Beschreibung der Ordnungsrelation
  order:order(distributed:nil sorted:[2 4#6]²together:nil)
  room:4                % zugewiesener Raum
  rooms:[3#14]          % Menge der verfügbaren Räume (Raum–IDs, eingeschränkt)
  teachers:[5]          % Menge der beteiligten Lehrer (Lehrer–IDs)
  time:14               % zugewiesene Zeit
                        % Menge der verfügbaren Zeiten (Zeit–Nr., eingeschränkt)
  times:[1#7 11#17 21#27 31#37 41#47])
```

□

Veranstaltung

Eine Veranstaltung ist eine Menge von Unterrichtseinheiten, deren Namen, Gruppen–, Lehrer– sowie initialisierten Raum– und Zeitmengen identisch sind. Zwischen diesen Einheiten gilt eine Ordnungsrelation, die in Kapitel 3.1.3 Punkt 2. beschrieben wird.

3.3 Beispiel (Veranstaltung)
Die hier dargestellte Veranstaltung wird in 5 Einerblöcke (Einzelstunden) geteilt. Die zweite dieser Einheiten ist in Beispiel 3.2 zu sehen.

```
o(all:5                 % Anzahl der Einheiten
  groups:[1 2]          % Menge der beteiligten Gruppen (Gruppen–IDs)
  name:2                % Name der Einheit (Fach–ID)
  order:[1 1 1 1 1]     % Ordnung der Einheiten untereinander
  rooms:[1#19]          % Menge der verfügbaren Räume (Raum–IDs)
  teachers:[5]          % Menge der beteiligten Lehrer (Lehrer–IDs)
  times:[1#55])         % Menge der verfügbaren Zeiten (Zeit–Nr.)
```

□

[2] diese Schreibweise steht abkürzend für `[2 4 5 6]`; sie ist jedoch syntaktisch nicht korrekt

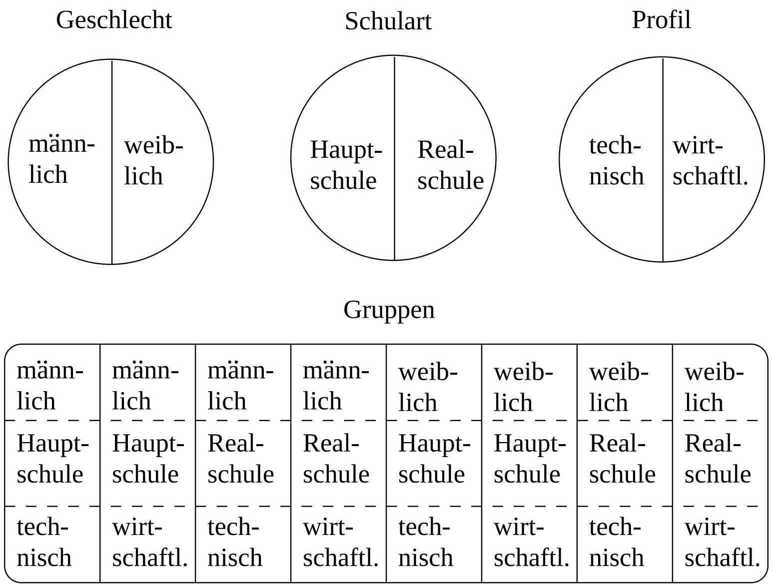

Abbildung 3.1: Klasse 7wt des Beispieles aus Anhang A und ihre 8 Gruppen

Block, Blockeinheiten

Unterrichtseinheiten einer Veranstaltung, deren Zeiteinheiten aufeinanderfolgend und am gleichen Tag stattfinden, heißen Blockeinheiten. Meist treten sie in Form einer Doppelstunde auf. Je nachdem, wieviele Einheiten zu einem solchen Block gehören, nennt man diesen n–Block (z.B. Zweierblock). Da $n = 1$ zulässig ist, kann eine einzelne Unterrichtseinheit als (Einer–)Block bezeichnet werden.

Klassen und Gruppen

Klassen sind Mengen von Schülern, die in den Unterrichtsfächern gemeinsam unterrichtet werden. Dabei kann es jedoch Ausnahmen geben. Fächer, die nur von einigen Schülern einer Klasse besucht werden, können nicht für die gesamte Klasse geplant werden. Mehrere verschiedene Unterrichtsfächer können demzufolge für eine Klasse gleichzeitig stattfinden, wenn jeder Schüler maximal eines dieser Fächer besucht.

Aus diesem Grund werden Klassen in Gruppen eingeteilt. Jeder Schüler gehört dabei genau einer Gruppe an. Werden alle Schüler einer Klasse in allen Unterrichtsfächern gemeinsam unterrichtet, existiert genau eine Gruppe, die mit der Klasse identisch ist.

Gruppenbildung tritt z.B. auf:

- bei Sportunterricht, den Mädchen und Jungen getrennt voneinander haben;
- bei unterschiedlichem Lehrangebot für Schüler verschiedener Schularten (z.B. Haupt–, Realschule);
- bei unterschiedlichem Unterrichtsangebot für Schüler mit verschiedenen Unterrichtsprofilen (z.B. technisches, wirtschaftliches, hauswirtschaftliches, na-

turwissenschaftliches, sprachliches Profil);

- bei Klassenaufteilungen zur Minimierung der Schüleranzahl auf Grund von knappen Ressourcen an Lehrmitteln (z.B. Computerarbeitsplätze).

Abbildung 3.1 stellt die Bildung von 8 Gruppen dar, die auf Grund verschiedener Aufteilungen einer Klasse entstehen.

Vollständiger Plan

Der Gesamtplan einer Schule, der alle zwingend vorgeschriebenen Kriterien erfüllt, heißt vollständiger Plan. Diese Kriterien können z.B. effektiv durch harte Constraints beschrieben werden. Kriterien zur Optimierung eines Planes können dabei erfüllt sein, jedoch ist dies zufällig.

3.1.3 Problemanalyse

Die Problematik der Stundenplanerstellung läßt sich auf teilweise komplexe Beziehungen zurückführen, die immer erfüllt sein müssen. Ein Plan, der diese Beziehungen für alle Unterrichtseinheiten erfüllt, heißt vollständiger Plan.

Neben den grundlegenden Beziehungen existieren weitere Spezialfälle. Sie sind zwar in den allgemeingültigen Beziehungen eines vollständigen Planes enthalten, die Formulierung einer Beziehung durch verschiedene Constraints vereinfacht jedoch die Einschränkung der Constraintvariablen und verkleinert damit den Suchraum. Das kann die Berechnungszeit einer Lösung verringern (vgl. hierzu Abschnitt 2.1.2).

Beziehungen eines vollständigen Plans

1. Eine Unterrichtseinheit kann nur zu einer Zeit stattfinden. Alle Lehrer und Gruppen, die an der Einheit beteiligt sind, sowie mindestens ein ihr zur Auswahl stehender Raum müssen diese Zeiteinheit zur Verfügung haben. D.h., die einer Unterrichtseinheit zur Verfügung stehenden Zeiten gelten sowohl für alle beteiligten Lehrer und Gruppen als auch für mindestens je einen Raum, in dem die Einheit stattfinden kann.

2. Eine Veranstaltung besteht aus Blöcken. (Trivialerweise gibt es somit auch Blöcke bestehend aus einer einzelnen Unterrichtseinheit. Besteht eine Veranstaltung aus einer einzelnen Unterrichtseinheit, so besitzt sie genau einen Block der Größe 1.) Tauscht man in einem Plan die Tupel aus Zeit und Raum mehrerer gleich großer Blöcke untereinander aus, so entsteht ein anderer Plan, der jedoch die gleichen Eigenschaften wie der Ausgangsplan besitzt. Um die Entstehung äquivalenter Pläne/redundante Lösungen zu vermeiden und damit Kosten zu sparen (Rechenzeit), existiert eine Relation zwischen den Unterrichtseinheiten einer Veranstaltung:

    (a) Zwischen gleich großen Blöcken besteht eine einfache „Kleiner–als“–Beziehung, d.h., ist die Positionierung einer Unterrichtseinheit innerhalb der Veranstaltung kleiner als die einer anderen Einheit der selben Veranstaltung innerhalb eines anderen, gleich großen Blocks, so ist auch der Tag, an dem sich deren Zeiteinheit befindet, kleiner als der der zweiten Einheit.

(b) Blöcke werden dadurch erzeugt, daß Unterrichtseinheiten einer Veranstaltung zeitlich direkt hintereinander geplant werden. Zwischen den Unterrichtseinheiten eines Blocks gilt, daß die Differenz zwischen den Positionierungen zweier Einheiten auch zwischen den Zeiten der beiden Einheiten besteht. Zusätzlich müssen diese am gleichen Tag stattfinden.

(c) Blöcke unterschiedlicher Größe einer Veranstaltung finden an verschiedenen Tagen statt.

3. Da Lehrer und Gruppen zur gleichen Zeit nicht an mehreren Unterrichtseinheiten teilnehmen können, sind die Zeiten aller Unterrichtseinheiten eines Lehrers bzw. einer Gruppe verschieden.

4. In einem Raum können zu einer Zeit nicht mehrere Unterrichtseinheiten stattfinden. Das bedeutet, die Räume aller Unterrichtseinheiten, die zur gleichen Zeit stattfinden, sind verschieden.

Abgeleitete Spezialfälle

5. Eine Zeiteinheit kann maximal so oft verplant werden, wie Räume diese Zeit verfügbar haben.

6. In Schulen findet Fachunterricht häufig in einem dafür vorgesehenen Fachraum statt (z.B. Sport in der Turnhalle). Die Zeiten der Unterrichtseinheiten, die nur in einem solchen Fachraum durchgeführt werden, sind verschieden.

3.1.4 Kriterien für die Verbesserung eines Planes

Die Erzeugung eines vollständigen Stundenplanes für eine Schule mit vielen Unterrichtseinheiten ist eine komplexe und zeitaufwendige Aufgabe. Erschwerend kommt hinzu, daß der Plan nicht nur vollständig, sondern auch gewissen (teilweise subjektiven) Gütekriterien genügen muß. Im allg. wird in den Schulen bei der schrittweisen manuellen Erstellung bei jedem Setzen einer Unterrichtseinheit darauf geachtet, die Kriterien einzuhalten. Der erste vollständige Plan ist daher meist schon ein guter, der nur noch wenig optimiert werden kann.

Als oberstes Kriterium neben der Vollständigkeit eines Planes (die immer erfüllt sein muß) gilt eine minimale Anzahl von Freistunden für die Schüler. In Schulen, die über genügend Ressourcen (Räume, Lehrer) verfügen, gilt sogar die strenge Vorgabe, keine Freistunden zu erzeugen. Andere Schulen lassen nur bei älteren Klassenstufen Freistunden zu, da sie jüngere Schüler während Freistunden nicht unbeaufsichtigt lassen dürfen.

Weiterhin gibt es eine Vielzahl von Kriterien, die einen Plan für Lehrer und Schüler optimal erscheinen lassen. Meist sind sie auf die Bedürfnisse einer Schule abgestimmt, teilweise variieren sie von Schuljahr zu Schuljahr. Einige der gebräuchlichsten Kriterien neben der Vollständigkeit des Planes und einer minimalen Anzahl von Freistunden für Schüler sind:

- Unterrichtseinheiten finden vorzugsweise morgens und am frühen Nachmittag statt und sind für Schüler gleichmäßig über die Wochentage verteilt.
- Für Lehrer können freie Zwischenstunden existieren, jedoch sollen sie eine individuell festzulegende Anzahl nicht übersteigen. Die Anzahl der Unterrichtstunden pro Tag wird meist auch festgelegt, um eine ausgewogene Aufteilung der Einheiten zu erreichen.
- Raumwechsel sind für Schüler gering zu halten. Ein Wechsel zwischen entfernten Gebäudeteilen sollte nur einmal pro Tag erfolgen.
- Anstrengende bzw. ermüdende Unterrichtseinheiten wie z.B. Sport sollten wenn möglich für Schüler nur am Anfang oder am Ende eines Unterrichtstages geplant werden.
- Nach Möglichkeit soll eine Klasse so viel Unterricht wie möglich bei ihrem Klassenlehrer bzw. in ihrem Klassenzimmer verbringen.

3.2 Das Planungssystem StuPlan

3.2.1 Überblick

Gegenstand der Arbeit ist es, ein Planungssystem zu implementieren. Das System ist ausschließlich zur Planung von Stundenplänen konzipiert, wie sie in Schulen verwendet werden. Im weiteren wird Stundenplanungssystem durch SPS abgekürzt; das implementierte System heißt StuPlan.

Das System StuPlan ist ein Prototyp, der sich in vier Module gliedert. Sie erfüllen die folgenden Aufgaben:

1. Erstellung eines vollständigen Planes durch ein Constraintsystem (`plan.oz`),
2. Optimierung eines Planes hinsichtlich wichtiger Gütekriterien (`mas.oz`),
3. graphische Anzeige von Plänen (`ausgabe.oz`),
4. Portierung von Plänen in LaTeX–Tabellen (`tabelle.oz`).

Jedes Modul lädt die problemspezifische Struktur `problem`. Die ersten beiden Module modifizieren und speichern sie. Darüberhinaus stehen weitere Dateien zur Verfügung:

Datei	Beschreibung
`general.oz`	allgemeingültige Prozeduren für die einzelnen Module
`control.oz`	globale Variablen zur Steuerung der Optimierung
`save.oz`	Speicherung der Datenstruktur
`schule.oz`	`problem`–Struktur, Ausgangsdaten der Schule
`schule1.oz`	`problem`–Struktur, erster Plan
`schule2.oz`	`problem`–Struktur, optimierter Plan

3.2.2 Datenstrukturen

Ein zu lösendes Problem wird durch die Datenstruktur `problem` beschrieben. Der Verbund (Record) besteht aus acht Listen. In diesen werden Informationen über

- Lehrer (Slot `teachers`),
- Klassen (Slot `forms`),
- Gruppen (Slot `groups`),
- Räume (Slot `rooms`),
- Unterrichtsfächer im allgemeinen (Slot `subjectnames`) und
- im speziellen (Slot `subjects`) sowie
- Planungstage (Slot `days`) und
- Planungseinheiten (Slot `times`)

Slot	Typ	Beschreibung
`account`	Integer	Kontostand eines Agenten (Lehrer, Gruppe)
`all`	Integer	Anzahl der Einheiten einer Veranstaltung
`endTime`	String	Angabe des Endes einer Zeiteinheit
`groups`	Liste	Menge der Gruppen (ID) einer Unterrichtseinheit
`id`	Integer	eindeutige Referenznummer, ID
`lastTime`	Integer	aktuellen Zeit einer Unterrichtseinheit
`name`	String	verbale Beschreibung des Elementes
`number`	Integer	Nummer der Unterrichtseinheit einer Veranstaltung
`order`	Liste, Record	Bei Veranstaltungen bestimmen die Elemente der Liste die Größe der einzelnen Blöcke. Bei Unterrichtseinheiten bestimmt der Verbund `order` über die drei Slots `together`, `sorted` und `distributed`, welche Beziehungen zwischen den Einheiten einer Veranstaltung bestehen. Sie werden als 5. Constraint im folgenden Abschnitt beschrieben.
`parts`	Liste	Menge der Gruppen (ID) einer Klasse oder eines Klassenteiles
`room`	FD	einschränkbare Raummenge einer Unterrichtseinheit
`rooms`	Liste	Menge der Räume (ID) einer Unterrichtseinheit
`startTime`	String	Angabe des Beginns einer Zeiteinheit
`subjects`	Liste	Menge von Fächern (ID), die in einem Raum stattfinden können
`teacher`	Integer	Klassenlehrer (ID)
`teachers`	Liste	Menge der Lehrer (ID) einer Unterrichtseinheit
`time`	FD	einschränkbare Zeitmenge einer Unterrichtseinheit
`times`	Liste	Menge von Zeiten einer Unterrichtseinheit

Tabelle 3.2: Die Slots der Datenelemente und ihre Bedeutung (FD: Finite Domain)

zusammengefaßt. Jede Liste besteht aus einer Menge von Records, die die relevanten Daten eines Listenelementes enthalten. Die Elemente werden durch eine Menge von Slots definiert (siehe Tabelle 3.2). Tabelle 3.3 stellt die Slots der unterschiedlichen Listenelemente dar.

3.2.3 Das Constraintsystem

Aufbauend auf die in der Projektarbeit [Anke96] untersuchten Beziehungen, die in einem SPS gelten, sind die Constraints von StuPlan überarbeitet und teilweise stark verändert worden. Aus diesem Grund werden die allg. Beziehungen und die daraus resultierenden Constraints ausführlich vorgestellt.

Slots	subjects	teachers	groups	subjectnames	rooms	forms	days	times
id	A	E	E	E	E	E	E	E
name	E	E	E	E	E	E	E	
times	E/B	E	E	E	E			
time	B							
lastTime	A							
rooms	E/B							
room	B							
teachers	E							
groups	E							
order	E/B							
all	E							
number	A							
account		A	A					
teacher			E					
subjects					E			
parts						E		
startTime								E
endTime								E

Tabelle 3.3: Datenstrukturen und ihre Slots (A: automatische Erzeugung der Werte, E: manuelle Eingabe, B: Berechnung der Werte durch Constraintsystem, E/B: manuelle Eingabe und automatische Veränderung der Werte)

Generell existieren für jede Unterrichtseinheit zwei Constraintvariablen, deren Wertebereiche einzuschränken sind. Die Variablen repräsentieren die Zeiteinheit und den Raum, die für die Unterrichtseinheit geplant werden und stehen über die Slots `time` und `room` zur Verfügung.
Die Räume können selbst über eingeschränkte Zeiten verfügen, so daß Raum und Zeit einer Einheit voneinander abhängig sind. Die entsprechenden Variablen einer der beiden Komponenten sind vor denen der anderen zu planen. Da bei der Suche nach einer Lösung der Zeiten auf Grund der größeren Komplexität der Suchraum stärker eingeschränkt wird, ist es sinnvoll, die Zeiten vor den Räumen zu planen.

Algorithmus 3.1 verdeutlicht die nacheinander stattfindenden Berechnungen der beiden Werte. Gibt es für wenigstens eine Unterrichtseinheit keine Zeit, so ist das Problem bezüglich der Zeitkomponete überbestimmt. Die Berechnung der Räume, in denen die Einheiten stattfinden, entfällt deshalb. Ist jedoch für wenigstens eine Raumvariable keine Raumzuteilung möglich, ist eine weitere Lösung für die Zeiten der Einheiten zu berechnen.

Die Constraints

Die folgenden Constraints beschreiben einen vollständigen Plan, wie er duch das System StuPlan erzeugt wird. Die Spezifikation der zugrundeliegenden Beziehungen erfolgte in Abschnitt 3.1.3:

1. Eine Unterrichtseinheit kann nur zu genau einer Zeiteinheit stattfinden. Diese besitzt sie selbst sowie jeder Lehrer und jede Gruppe, die an dieser Einheit beteiligt sind, als verfügbare Zeit. Weiterhin gibt es mindestens einen der Einheit zur Auswahl stehenden Raum, der diese Zeit verfügbar hat.
   Die verfügbaren Zeiten der Lehrer, Gruppen, Räume und Unterrichtseinheiten werden vorgegeben; die der Einheiten schränken sich wie folgt ein:

   $$S_i.\mathtt{times} = \left(\bigcap_{L_j \in \mathcal{L}_i} L_j.\mathtt{times}\right) \cap \left(\bigcap_{G_j \in \mathcal{G}_i} G_j.\mathtt{times}\right) \cap \left(\bigcup_{R_j \in \mathcal{R}_i} R_j.\mathtt{times}\right) \cap S_i.\mathtt{times}$$

   mit

   S_i ... Unterrichtseinheit i
   $\mathcal{L}_i$ ... Menge der Lehrer der Einheit i
   $\mathcal{G}_i$ ... Menge der Gruppen der Einheit i
   $\mathcal{R}_i$ ... Menge der verfügbaren Räume der Einheit i

   Die verfügbaren Zeiten repräsentiert der Slot `times` des jeweiligen Elements. Eine Unterrichtseinheit findet zu einer Zeit statt, die für sie verfügbar ist:

   ```
   {ForAll Solution proc {$ S} S.time={FD.int S.times} end}
   ```

2. Da Lehrer und Gruppen zur selben Zeit nicht mehrere Unterrichtseinheiten besuchen können, sind die Zeiten aller Einheiten verschieden, denen wenigstens ein Lehrer bzw. eine Gruppe gleichermaßen zugeteilt ist.

   ```
   {ForAll {Append Problem.groups Problem.teachers}
   ```

Algorithmus 3.1 Berechnung der Gesamtlösung für Zeit– und Raumvariablen

1. Initialisierung und Propagierung der Zeitvariablen der Unterrichtseinheiten
2. Enumeration und Propagierung der Zeiten liefert die Teillösung sol_{time}
3. ist sol_{time} leer, existiert keine Lösung des Problemes
   sonst stellt sol_{time} eine Lösung für die Zeiten dar
4. Initialisierung und Propagierung jeder Raumvariable
5. Enumeration und Propagierung der Räume liefert sol_{room}
6. ist sol_{room} leer, dann weiter bei 2.
   sonst stellt sol_{room} eine Lösung des Problemes dar (Zeiten und Räme)

```
proc {$ X}
   {FD.distinct {FilterMap Solution
                 fun {$ S} {Member X.id (S.{Label X})} end
                 fun {$ S} S.time end}}
end}
```

3. Es können nicht mehr Unterrichtseinheiten zur gleichen Zeit geplant werden, wie Räume zu dieser Zeit zur Verfügung stehen. In StuPlan wird dies wie folgt implementiert:

```
{ForAll AllTimes
 proc {$ T}
    {FD.atMost
     {Length
      {Filter Problem.rooms fun {$ R} {Member T R.times} end}}
     {Map Solution fun {$ S} S.time end} T}
 end}
```

4. Alle Unterrichtseinheiten, die in einem ganz bestimmten Raum stattfinden, haben unterschiedliche Zeiten.

```
{ForAll Problem.rooms
 proc {$ X}
    {FD.distinct {FilterMap Solution
                  fun {$ S} S.rooms==[X.id] end
                  fun {$ S} S.time end}}
 end}
```

5. Eine Veranstaltung besteht aus einer Menge von Blöcken. Jeder Block besteht aus einer Menge von Unterrichtseinheiten. Zwischen allen Einheiten der Veranstaltung besteht eine Relation mit folgenden Eigenschaften:

   - Die Zeiten der Einheiten eines Blockes liegen alle am gleichen Tag. Die Differenz ihrer `number`–Werte bezeichnet ihren zeitlichen Abstand voneinander.
     Der Slot `order.together` einer Unterrichtseinheit enthält die Liste von ID's anderer Einheiten. Diese befinden sich zusammen mit der Einheit in einem Block.
   - Befinden sich zwei Einheiten in verschiedenen Blöcken, finden sie an unterschiedlichen Tagen statt (Slots: `order.distributed`, `order.sorted`).
   - Das Vertauschen der zeitlichen Reihenfolge von gleichgroßen Blöcken beinhaltet kein qualitativ anderes Ergebnis. Aus diesem Grund besteht eine Ordnung zwischen den Einheiten dieser Blöcke. Der Block mit kleineren `number`–Werten findet früher statt. Zur Bestimmung solcher Einheiten dient der Slot `order.sorted`.
   - Zwischen unterschiedlich großen Blöcken besteht keine eindeutige Reihenfolge (Slot: `order.distributed`).

```
{ForAll Solution
```

```
proc {$ S}
   case S.order.distributed\=nil
   then {ForAll
         {Filter Solution
          fun {$ X} {Member X.id S.order.distributed} end}
         proc {$ X} {Day X.time}\=:{Day S.time} end}
   else true
   end
   case S.order.sorted\=nil
   then {ForAll
         {Filter Solution
          fun {$ X} {Member X.id S.order.sorted} end}
         proc {$ X}
            case X.number<S.number
            then {Day X.time}<:{Day S.time}
            else {Day X.time}>:{Day S.time}
            end
         end}
   else true
   end
   case S.order.together\=nil
   then {ForAll
         {Filter Solution
          fun {$ X} {Member X.id S.order.together} end}
         proc {$ X}
            case X.number<S.number
            then {FD.plus X.time (S.number-X.number) S.time}
            else {FD.plus S.time (X.number-S.number) X.time}
            end
            {Day X.time}=:{Day S.time}
         end}
   else true
   end
end}
```

6. Unterrichtseinheiten können nur in den Räumen stattfinden, die ihnen zur Verfügung stehen. Die Liste der Räume befindet sich im Slot `rooms` der Einheiten.

```
{ForAll Solution proc {$ S} S.room={FD.int S.rooms} end}
```

7. Allen Unterrichtseinheiten, die zur gleichen Zeit geplant sind, müssen unterschiedlichen Räumen zugewiesen sein. Somit wird verhindert, daß mehrere Einheiten zur selben Zeit den gleichen Raum belegen.

```
{ForAll AllTimes
 proc {$ T} {FD.distinct {FilterMap Solution
                         fun {$ S} S.time==T end
                         fun {$ S} S.room end}}
 end}
```

8. Unterrichtseinheiten eines Blockes finden im gleichen Raum statt.

```
{ForAll Solution
 proc {$ S}
    case S.order.together\=nil
    then {ForAll {FilterMap Solution
                  fun {$ X} {Member X.id S.order.together} end
                  fun {$ X} X.room end}
          proc {$ X} X=:S.room end}
    else true
    end
 end}
```

9. Es können nicht mehr als n Unterrichtseinheiten, deren Mengen von verfügbaren Räumen exakt gleich sind, zur selben Zeit stattfinden. n ist dabei die Anzahl der Räume. Die Beziehung trägt nicht zur Konsistenzsicherung eines vollständigen Planes bei, kann aber den Suchraum des Problemes verkleinern. Die entsprechenden Constraints werden durch die folgende Funktion im Berechnungsraum installiert:

```
proc {SameRoomsDiffTimes Subs}
   case Subs
   of S|Sr then Ys Zs L={Length S.rooms} in
      {List.partition Sr fun {$ X} S.rooms==X.rooms end Ys Zs}
      case {Length Ys}>=L
      then Xs={Map S|Ys fun {$ Y} Y.time end} in
         {ForAll AllTimes proc {$ T} {FD.atMost L Xs T} end}
      else true
      end
      {SameRoomsDiffTimes Zs}
   else true
   end
end
```

Distribuierung

In Kapitel 2.4.3 wurde die Notwendigkeit der Problemaufteilung beschrieben. Da i.allg. mehrere Lösungen für das Stundenplanungsproblem existieren, ist eine Distribuierung des durch die Constraints gebildeten Berechnungsraumes vorzunehmen, wenn die Variablen noch nicht auf jeweils maximal einen Wert eingeschränkt sind.

Von dem Berechnungsraum, der das Problem enthält, wird eine Kopie erzeugt. Dem Originalberechnungsraum wird ein Constraint C und der Kopie des Berechnungsraumes $\neg C$ hinzugefügt. Die Distribuierungsstrategie bestimmt die Auswahl von C. Sie hat in Oz zwei Freiheitsgrade:

1. Wahl einer noch nicht auf einen Wert eingeschränkten Variable Var,

2. Wahl eines Wertes Val aus dem Wertebereich von Var.

C wird durch das Constraint $Var = Val$ gebildet, $\neg C$ durch $Var \neq Val$.

In StuPlan wurden verschiedene Distribuierungsstrategeien getestet. Erfolgreich bei der Suche nach einer Lösung in möglichst kurzer Zeit ist folgende modifizierte *first–fail*–Strategie:
Aus der Menge der Variablen, die noch mehrere Werte in ihrem Wertebereich haben, wird die Variable ausgewählt, deren Wertemenge am kleinsten ist. Von den Zeiten, die in dieser Menge enthalten sind, wird die ausgewählt, die an einem beliebigen Tag so früh wie möglich liegt. Gibt es mehrere Tage mit der gleichen frühesten Zeiteinheit, so wird die kleinste gewählt.

Es hat sich gezeigt, daß mit dieser Strategie das Finden einer ersten Lösung in relativ kurzer Zeit möglich ist. Sie setzt die Unterrichtseinheiten nach Möglichkeit gezielt eher morgens als nachmittags und annähernd gleichverteilt. Damit werden wichtige Gütekriterien schon beim Setzen der Einheiten beachtet.

3.4 Beispiel (Distribuierung)
In nebenstehendem Plan sind die noch verfügbaren Zeiten einer Unterrichtseinheit dargestellt. Die frühestmögliche Zeit ist die 2. Stunde donnerstags und sonnabends. Da die Zeiteinheit 20 kleiner als 32 ist, wird der Zeitvariablen der Einheit die Zeit 20 (2. Stunde am Donnerstag) im aktuellen Berechnungsraum zugewiesen. Im kopierten Berechnungsraum wird die Zeit 20 für die Unterrichtseinheit ausgeschlossen. □

	Mo	Di	Mi	Do	Fr	Sa
I						
II				20		32
III			15			33
IV	4	10	16	22	28	34
V		11		23		
VI	6	12		24	30	

3.2.4 Das Multiagentensystem

Optimierung

Als wichtiges Kriterium, welches einen guten Stundenplan auszeichnet, gilt eine minimale Anzahl von Freistunden (Lücken), die sich an einem Tag zwischen zwei Unterrichtseinheiten befinden.

Einige Schulen stellen die Anforderung, daß Klassen keine Lücken in ihren Plänen haben dürfen. Nach Aussagen von Lehrern ist das vor allem in Grundschulen wegen der geringen Komplexität der Plänen leicht zu realisieren. Ansonsten ist die Anzahl der Freistunden so gering wie möglich zu halten.
Lehrer sollen zwar auch wenig Freistunden besitzen, können diese aber z.B. für die Unterrichtsvorbereitung nutzen. Einige Freistunden sind hier sogar wünschenswert. Für Räume spielen Freistunden keine Rolle.

Das Stundenplanungssystem StuPlan versucht, den Ausgangsplan dahingehend zu optimieren, daß Gruppen und Lehrer so wenig wie möglich ungewollte Lücken in ihren Plänen haben. Dazu werden sie als Agenten angesehen, die ihre Unterrichtseinheiten umstellen und über den verändeten Plan mit den anderen Agenten verhandeln.

Als erster Ansatz für ein Multiagentensystem diente ein Contract–Net–Modell, wie es in [Dilg95] angedacht war. Dabei sollte ein Agent einige Unterrichtseinheiten seines Planes verschieben und so die Anzahl seiner Freistunden verringern. Über diese Verschiebungen hätte er mit den Agenten, deren Plan dabei auch verändert wird, verhandeln müssen. Diese hätten dann wiederum mit weiteren Agenten verhandeln sollen, um die Konsistenz ihrer Pläne zu gewährleisten. Bei der Implementierung zeigte es sich, daß dieser Herangehensweise auf Grund der Komplexität der Aufgabe zu keinen Lösungen innerhalb einer vertretbaren Zeit führte.
Für die Veränderung der Pläne bietet sich daher das beschriebene Constraintsystem als Problemlösungskomponente der Agenten an. Umfangreiche Tests haben gezeigt, daß damit relativ schnell ein leicht veränderter Plan erzeugt werden kann. Desweiteren sichert es ohne weiteren Aufwand die Konsistenz der Pläne. D.h., umfangreiche Algorithmen, die z.B. gewährleisten, daß eine Raumaufteilung im veränderten Plan möglich ist, entfallen.

Das Multiagentensystem arbeitet in sogenannten Runden. In jeder Runde ist jeder Agent maximal einmal damit beschäftigt, einen besseren Plan zu erzeugen. Dafür stehen ihm zwei Möglichkeiten zur Verfügung:

1. er verkleinert die Anzahl seiner Lücken oder
2. die Anzahl seiner Lücken bleibt konstant, jedoch verkleinert sich die Gesamtanzahl der Lücken im Plan.

Dazu wird wenigstens eine Zeiteinheit, die für den Agenten als Lücke besteht, mit einer Unterrichtseinheit besetzt. Die anderen Einheiten werden nach Möglichkeit an ihre vorherige Position gesetzt, so daß eine geringe Veränderung des Planes stattfindet.

Ist ein Agent mit der Verbesserung des Planes an der Reihe und er besitzt keine Lücke, dann wird der Plan durch diesen Agenten nicht verändert. Ist der Gesamtplan am Ende einer Runde identisch mit dem Plan am Anfang der Runde, so endet die Optimierung. Der optimierte Plan ist der beste Plan, der während des Optimierungsprozesses erzeugt wurde. Ein Plan ist gegenüber einem anderen als besser anzusehen, wenn er

1. weniger Lücken aufweist oder
2. die Anzahl der Lücken gleich ist, jedoch mehr Agenten lückenlos sind.

Kommunikationsprotokoll

Hat ein Agent A eine Planveränderung nach obigen Kriterien gefunden, so versucht er durch Kommunikation, diese bei den anderen Agenten durchzusetzen und damit den Plan zu verändern.
Um das zu erreichen, verhandelt A mit allen anderen Agenten über das Protokoll Querying, wie es in Abbildung 3.2 dargestellt ist.

Agent A fragt bei einem anderen Agenten B an, ob dieser mit dem veränderten Plan einverstanden ist. Tritt für B keine Verschlechterung seines Planes ein, antwortet

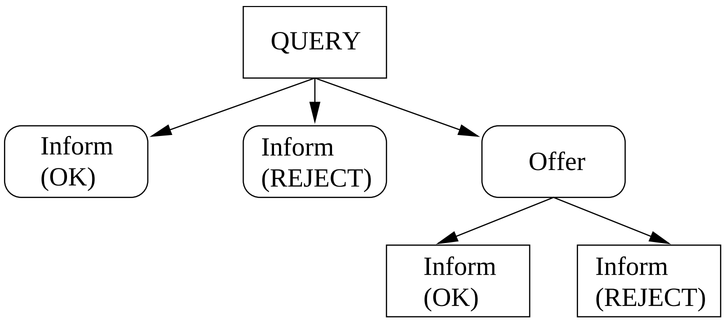

Abbildung 3.2: Protokoll Querying

er mit Ok und stimmt damit der Planveränderung zu. Das Kriterium, ob der Plan schlechter wird, ist eine Vergrößerung der Anzahl der Lücken für B.
Ist die Verschlechterung des Planes gravierend, so lehnt B die Anfrage ab. Die Veränderung des Planes wird damit nicht wirksam. Agent A kann versuchen, eine andere Planänderung vorzunehmen oder er gibt die Kontrolle an den nächsten Agenten ab.
Tritt für B nur eine geringfügige Planverschlechterung auf, so kann er zuvorkommend handeln und A ein Angebot unterbreiten: Wenn A bereit ist, B zu „belohnen" und damit die Verschlechterung auszugleichen, würde B der Veränderung zustimmen. Je nachdem, ob A mit diesem Angebot einverstanden ist, kann er annehmen oder ablehnen. Im letzteren Fall scheitert die Planveränderung.

Werteinheiten

Um anderen Agenten einen Anreiz für zuvorkommendes Verhalten zu bieten, besitzt jeder Agent ein Konto (Slot `account`). Es enthält eine begrenzte Anzahl von Werteinheiten, mit denen ein anderer Agent belohnt werden kann.
Am Anfang der Optimierung erhält jeder Agent so viele Einheiten, wie er Freistunden besitzt (zuzüglich eines Sockelbetrages). Verschlechtert sich ein Agent, bekommt er vom planverändernden Agenten eine Werteinheit von dessen Konto.
Ein Agent mit wenigen Werteinheiten kann den Plan nicht verändern, wenn sich mehr Agenten verschlechtern würden, als er entschädigen kann. I.allg. hat dieser Agent aber nur wenige Freistunden. Agenten mit vielen Freistunden und einer großen Anzahl an Werteinheiten können jedoch leichter Planveränderungen durchsetzen. Damit wird erreicht, daß der Abbau der Freistunden fair verläuft.

Zyklen

Wird ein Plan verändert, kann sich die Gesamtanzahl der Lücken vergrößern, wenn der planveränderde Agent seine Lückenanzahl minimiert. Dadurch kann es zu Zyklen kommen, bei denen ein Agent eine zuvor realisierte Veränderung zurücknimmt. Dieser Fall kann zwar als selten angesehen werden, jedoch könnte er auftreten und die Optimierung unendlich lange fortschreiten lassen, ohne daß eine weitere Verbesserung eintreten würde.

In StuPlan wird ein veränderter *Simulated–Annealing*–Algorithmus verwendet (siehe Kapitel 2.3). Neben seinen Vorteilen bei der Suche nach einem globalen Minimum zur Optimierung einer Lösung kann er unendlich oft wiederkehrende Zyklen vermeiden helfen.
Um eine nächste Lösung auszusuchen, werden hier jedoch nicht alle benachbarten Lösungen betrachtet, sondern nur eine ähnliche. Genügt die den Kriterien des Algorithmus, werden keine weiteren Lösungen zum Vergleich gesucht.

Die Variable *Temperatur* gibt dabei an, um wieviele Lücken ein veränderter Plan zunehmen darf. Verschlechtert sich ein Plan maximal um den Wert der Temperatur, so ist das gleichbedeutend mit der Möglichkeit der Agenten, ein Belohnungsangebot an den planverändernden Agenten abzugeben.
Die Temperatur bleibt in einer Runde konstant. Sie nimmt jedoch von Runde zu Runde ab, bis sie bei Null angelangt ist. Eine Temperatur von Null ist invariant.

Ist die Temperatur nach einigen Runden Null, kann sich ein veränderter Plan im gesamten nicht mehr verschlechtern. Es kann jedoch weiterhin ein Zyklus auftreten. Verbessert sich ein Agent und verschlechtert den Plan eines anderen Agenten um die gleiche Anzahl an Lücken, so kann das wieder rückgängig gemacht werden, wenn der zweite Agent an der Reihe ist, den Plan zu verändern. In jedem dieser Fälle zahlt der sich verbessernde Agent eine Belohnung an den sich verschlechternden Agenten von seinem Konto. Der Kontostand ist zwar begrenzt, jedoch ist er nach der Rücksetzung des Planes durch den zweiten Agenten wieder gleich groß. Aus diesem Grund muß ein Agent, der den Plan verändert, eine Werteinheit von seinem Konto abgeben, ohne daß ein anderer Agent diese erhält. Damit wird ein Verringerung des Kontostandes eines Agenten erreicht. Irgendwann ist er nicht mehr in der Lage, den Plan zu verändern, da sein Konto leer ist.

Einordnung des Multiagentensystemes

Das Modell wird wie folgt in das Klassifikationsschema von Seite 25 eingeordnet:

- Systemmodell: Team
- Agentenzahl: mittel
- Interaktionsschema: einfach bis leicht komplex
- Anpassungsfähigkeit: starr (aus Effizienzgründen)
- Kontrollverteilung: abhängig (Agenten aktivieren sich gegenseitig)
- Ressourcen: eingeschränkt (Zeit)
- Granularität: mittel
- Problemlösung: Zerlegung
- Kooperativität: zuvorkommend bis konkurrierend
- Kopplung: mittel
- Agententyp: reaktiv

Implementierung

Um einen Plan zu optimieren, wird jeder Agent in einer Runde genau einmal aufgefordert, den Plan zu verändern. Agenten ohne Lücke verändern den Plan nicht. Waren alle Agenten an der Reihe, wird der entstandene Plan mit dem Ausgangsplan der Runde verglichen. Sind beide gleich, endet die Optimierung.
Im anderen Fall wird der Wert der Temperatur verringert, der Vorgang wiederholt sich und jeder Agent ist wieder genau einmal an der Reihe, den Plan zu verändern. Die Reihenfolge, in der die Agenten arbeiten, wird vor jeder Runde festgelegt. Dabei kommen Agenten mit wenigen Lücken vor Agenten mit vielen Lücken an die Reihe. Algorithmus 3.2 veranschaulicht die Optimierung eines Stundenplanes.

Verändert sich ein Plan, wird er mit dem bisher besten Plan verglichen. Ist er besser, wird der optimierte Plan zum besten Plan.

Algorithmus 3.3 zeigt die Berechnung eines Planes. Nachdem eine Lösung für die Zeiten der Unterrichtseinheiten gefunden wurde, wird eine Lösung für die Raumzuteilung berechnet. Existiert keine Raumzuteilung, muß eine andere Lösung für die Zeiten gefunden werden. Gibt es keine solche, ist das Stundenplanproblem nicht lösbar.

Ist ein Agent an der Reihe, den Plan zu verändern, berechnet er ihn neu wie unter Abschnitt „Optimierung“ auf Seite 55 beschrieben. Schafft er dies auf Grund der Komplexität nicht innerhalb einer vorgegebenen Zeit, erfolgt ein *Timeout*. Es wird kein veränderter Plan berechnet. Das gleiche passiert, wenn die Nichterfüllbarkeit eines besseren Planes erkannt wird.

Gelangt der Agent zu einem besseren Plan, verhandelt er darüber mit den anderen Agenten (Algorithmus 3.5). Verläuft die Verhandlung mit jedem Agenten positiv, so wird der veränderte Plan der aktuelle. Ansonsten kann der Agent die Neuberechnung eines besseren Planes wieder vornehmen. Die Anzahl dieser Versuche wird über die Variable *LoopCounter* vorgegeben.

Hat ein veränderter Plan insgesamt mehr Lücken als der vorherige Plan und die Differnz der Lücken ist größer als der Wert der Temperaturvariablen, so wird der Plan abgelehnt.
Ansonsten muß der planverändernde Agent mehr Werteinheiten auf seinem Konto haben, als die Anzahl der sich verschlechternden Agenten. Diese erhalten jeweils eine Werteinheit als Ausgleich für ihre Verschlechterung. Eine weitere Werteinheit muß der Agent zusätzlich abgeben.

Algorithmus 3.2 Erstellung eines optimierten Planes

1. Berechnung eines Planes P_{best} (siehe Alg. 3.3); keine Lösung $\longrightarrow$ Problem ist nicht lösbar
2. Wiederholung bis $P_{\text{vorRunde}} = P_{\text{nachRunde}}$
   2.1. Wiederholung für alle Agenten
      2.1.1. Optimierung des Planes $\longrightarrow P_{\text{opt}}$ (siehe Alg. 3.4)
      2.1.2. wenn P_{opt} besser als P_{best}, dann $P_{\text{best}} = P_{\text{opt}}$
   2.2. Temperatur $\neq 0 \longrightarrow$ dec(Temperatur)
3. optimierter Plan ist P_{best}

Algorithmus 3.3 Berechnung eines Planes

1. Wiederholung bis Lösung (bzw. Nichterfüllbarkeit) des Problemes
   1.1. Berechnung bzgl. `time`–Komponente;
   keine Lösung $\longrightarrow$ Problem nicht lösbar
   1.2. Berechnung bzgl. `room`–Komponente;
   keine Lösung $\longrightarrow$ weiter bei 1.1.

Algorithmus 3.4 Optimierung eines Planes für Agent A

1. $P_{\text{neu}} = P_{\text{alt}}$
2. A hat Lücken in seinem Plan $\longrightarrow$ Wiederholung bis optimierter Plan, Nichterfüllbarkeit des Problemes oder Timeout
   2.1. Berechnung eines optimierten Planes P_{opt} (siehe Alg. 3.3)
   2.2. Verhandlung über P_{opt} mit anderen Agenten
   2.3. bei positiver Verhandlung $\longrightarrow P_{\text{neu}} = P_{\text{opt}}$

Algorithmus 3.5 Verhandlung über einen veränderten Plan

1. $P_{\text{neu}} = P_{\text{alt}}$, Zähler = 0
2. Lücke(P_{opt}) $\leq$ Lücke(P_{alt}) + Temperatur $\longrightarrow$ Wiederholung für alle Agenten A außer planverändernden Agenten A^*
   2.1. Lücke($A(P_{\text{opt}})$) $>$ Lücke($A(P_{\text{alt}})$) $\longrightarrow$ inc(Zähler)
3. Zähler $<$ Konto(A^*) $\longrightarrow$ Konto(A^*) = Konto(A^*) - Zähler - 1, $P_{\text{neu}} = P_{\text{opt}}$
4. wiederholen für alle Agenten A, die ihren Plan verschlechtert haben
   4.1. inc(Konto(A))

3.3 Analyse weiterer Stundenplanungssysteme

Für einen Vergleich des Stundenplanungssystems StuPlan mit anderen Systemen sollen an dieser Stelle zwei verschiedene Programme vorgestellt werden.
Das erste ist ein kommerzielles Programm zur Stundenplanverwaltung, welches die Erstellung von Plänen unterstützt; das zweite ist als Beispielprogramm zur Veranschaulichung der Constraintprogrammierung in Oz von den Entwicklern der Sprache programmiert worden.

3.3.1 SSE Stundenplan–Verwaltung für Windows

SSE ist ein typischer Vertreter der Programme, die Lehrer bei der Erstellung von Stundenplänen unterstützen, jedoch i.allg. keine selbständig und vollständig generierten Pläne erzeugen. Es dient vielmehr der Verwaltung der Pläne und bietet mehr Übersichtlichkeit als die rein manuelle Planerstellung.

Das Programm übernimmt die Verwaltung der relevanten Schuldaten. Es erfaßt

- Stammdaten (Lehrer, Klassen, Räume, Fächer),
- deren Zuordnung (welche Klasse hat welches Fach bei welchem Lehrer und in welchem Raum) und
- die zeitlichen Einschränkungen, denen Lehrer, Klassen und Räume unterliegen können.

Die Eingabe der Daten erfolgt übersichtlich in Form eines Stundenplanes, sodaß lange Einarbeitungszeiten entfallen. Unterstützt wird dies auch durch eine kontext–orientierte Hilfe.
Jede Eingabe wird automatisch auf Überschneidungen mit vorher getroffenen Festlegungen überprüft. Dadurch kann das Programm auch alle aktuellen Möglichkeiten anzeigen, zu der eine Unterrichtseinheit geplant werden kann. Existiert keine Möglichkeit, müssen schon geplante Einheiten aus dem Stundenplan entfernt werden. Dafür ist der Benutzer selbst verantwortlich.

Weitere Mermale des Programmes SSE:

- Planung von 14–tägig stattfindenden Unterrichtseinheiten ist möglich;
- mehrere Lehrer können gleichzeitig in einer Klasse unterrichten;
- es werden sowohl harte als auch weiche Zeitvorgaben unterstützt (z.B. Lehrer L_1 hat Montag Nachmittag keinen Unterricht und die erste Stunde täglich lieber nicht);
- für Lehrer, Klassen und Räume können Wunschzeiten eingetragen werden, an denen deren Unterricht stattfinden soll;
- es können maximal 6 Tage mit jeweils 17 Zeiteinheiten verwaltet werden; die Länge der Zeiten ist frei konfigurierbar;
- Unterrichtspläne können für Lehrer, Klassen und Gruppen ausgegeben werden.

Fazit

SSE eignet sich besonders für kleine Schulen (z.B. Grundschulen), bei denen das Unterrichtsangebot überschaubar ist. Hervorzuheben ist die Möglichkeit, 14–tägig stattfindende Veranstaltungen zu verwalten. Von Nachteil erscheint jedoch, daß nur Unterricht betrachtet werden kann, der im Klassenverband unterrichtet wird.

Sollte kein vollständiger Gesamtplan für eine Schule existieren, ist es sehr schwer, die Nichtlösbarkeit des Problemes zu erkennen.

3.3.2 Oz College Timetabling (OCT)

Das Programm ist zur Veranschaulichung der Constraintprogrammierung in Oz entwickelt worden. Es liegt eine detaillierte Beschreibung ([HeWü94]) sowie der Quelltext vor. Diese beiden Dokumente bilden die Grundlage der folgenden Zusammenfassung.

Problembeschreibung

Die Grundlage des Programmes bildet der Stundenplan einer Saarbrücker Bildungseinrichtung. Sämtliche Daten stammen aus dem Frühjahrssemester 1995.

Die Studenten durchlaufen ihr Studium in einem Zyklus aus vier Jahren. Jedes der vier Semester besucht spezielle Pflichtveranstaltungen. Darüberhinaus existieren Veranstaltungen, die freiwillig von jedem Studenten besucht werden können.

Insgesamt gibt es 91 Unterrichtseinheiten, 34 Lehrer und 7 Räume unterschiedlicher Größe. Jeder Einheit ist ein Lehrer und eine Mindestraumgröße zugeordnet.

Der zu betrachtende Planungszeitraum gliedert sich in fünf Unterrichtstage (Montag bis Freitag). Die Veranstaltungen können zwischen 8.15 Uhr und 17.00 Uhr stattfinden. Sie beginnen jeweils zur vollen Viertelstunde. Es gibt kurze (45 Minuten) und lange Veranstaltungen (90 Minuten). Nach jeder Unterrichtseinheit muß eine Pause gewährleistet sein; bei kurzen Einheiten wenigstens 15 Minuten, sonst mindestens 30 Minuten.

Darüberhinaus gelten die folgenden Bedingungen:

- einige Unterrichtseinheiten sind auf bestimmte Zeiten eingeschränkt;
- einige Lehrer haben Ausschlußzeiten, während denen sie nicht unterrichten können;
- es gibt für die einzelnen Semester verschiedene Zeiten für die Mittagspause;
- einige Unterrichtseinheiten müssen nach anderen stattfinden;
- die Einheiten einiger Veranstaltungen müssen zeitgleich stattfinden, die der anderen müssen an verschiedenen Tagen gehalten werden;
- jeder Kurs muß in einem Raum ausreichender Größe stattfinden;
- jeder Lehrer kann zu einer bestimmten Zeit maximal eine Einheit unterrichten;

- zwei Lehrer müssen ihr behindertes Kind betreuen und können deshalb nicht zur selben Zeit unterrichten;
- die Pflichtveranstaltungen eines Semesters dürfen sich nicht überschneiden;
- einige der freiwillig zu besuchenden Veranstaltungen dürfen sich nicht mit den Pflichtveranstaltungen der ersten zwei Semester überschneiden, andere dürfen sich mit keiner der Pflichtveranstaltungen eines beliebigen Semesters überlagern;
- zwei Pflichteinheiten dürfen sich überschneiden, wenn die Studenten des Semesters in Gruppen aufgeteilt werden;
- die Lehrer wollen nicht an mehr als drei Tagen unterrichten.

Weiterhin sollten folgende Kriterien so gut wie möglich erfüllt werden:
Die Unterrichtseinheiten der ersten beiden Semester sollten montags, dienstags und mittwochs stattfinden. Ist das nicht möglich, sollten sie donnerstags bzw. am Freitag im Verlauf des Vormittags geplant werden. Das dritte Semester sollte hauptsächlich mittwochs unterrichtet werden und das vierte möglichst dienstags und donnerstags. Die Anzahl der Einheiten nach der Mittagspause sowie am Donnerstag und Freitag sind zu minimieren.

Unterschiede zwischen OCT und StuPlan

Da alle vorgegebenen Zeiten im Viertelstundentakt liegen und die Länge des Unterrichts und der Pausen ein Vielfaches von 15 Minuten beträgt, wird jeder Tag in 36 Zeiteinheiten mit einer Länge von 15 Minuten unterteilt. Insgesamt gibt es pro Woche 180 Zeiteinheiten. Kurze Unterrichtseinheiten sind dabei wie ein Dreierblock des Stundenplanungssystems StuPlan anzusehen, lange Einheiten wie ein Sechserblock.

Ein Hauptunterschied zwischen OCT uns StuPlan liegt somit darin, daß OCT mit einer Startzeit und einer vorgegebenen, jedoch variablen Zeitdauer pro Block operiert. In StuPlan wird demgegenüber ein n–Block in n Einheiten der festen Länge 1 unterteilt; es existieren deshalb n aufeinanderfolgende Startzeiten.

3.5 Beispiel (Verhinderung von Unterrichtsüberschneidungen)
Es sollen zwei Unterrichtseinheiten U_1, U_2 betrachtet werden, deren Länge eine Zeiteinheit beträgt ($U_1.\texttt{dur} = U_2.\texttt{dur} = 1$). Das Verhindern von Überschneidungen wird in OCT wie folgt ausgedrückt:

$$\begin{aligned} & U_1.\texttt{start} + U_1.\texttt{dur} \leq U_2.\texttt{start} \\ \vee\ & U_2.\texttt{start} + U_2.\texttt{dur} \leq U_1.\texttt{start} \end{aligned}$$

Das läßt sich wegen $U_1.\texttt{dur} = U_2.\texttt{dur} = 1$ einfacher beschreiben durch:

$$\begin{aligned} & U_1.\texttt{start} < U_2.\texttt{start} \\ \vee\ & U_2.\texttt{start} < U_1.\texttt{start} \end{aligned}$$

In StuPlan wird dies ausgedrückt durch[3]:

$$U_1.\texttt{time} \neq U_2.\texttt{time}$$

Die Disjunktion aus OCT wird in StuPlan durch $(n \cdot m)$ Ungleichungen ausgedrückt, wobei n und m die Größen der Blöcke sind. Da in Schulen Einheiten i.allg. nur Einzelstunden sind, reicht meist eine Ungleichung, um das Überschneiden von Einheiten zu verhindern. □

Ein zweiter Unterschied besteht in der Handhabung der Raumzuteilung. OCT geht von einer festgelegten Menge verfügbarer Räume für jede Unterrichtseinheit aus. Einheiten, die einen großen Raum beanspruchen, können dabei nur in einem der großen Räume stattfinden. Unterrichtseinheiten mit wenigen Studenten können hingegen in jedem Raum stattfinden. Die Menge der verfügbaren Räume einer Einheit bleibt jedoch zu allen Zeiten gleich.
Da in StuPlan auch für Räume Ausschlußzeiten gelten können, ist die Menge verfügbarer Räume einer Unterrichtseinheit abhängig von der Zeit, zu der diese stattfindet. Aus diesem Grund müssen erst alle Zeiten der Einheiten feststehen, um dann zu überprüfen, ob eine korrekte Zuteilung der Einheiten zu den Räumen möglich ist.

Weitere spezielle Unterschiede zwischen den beiden Programmen werden in den folgenden Abschnitten dargestellt.

Distribuierung

In Kapitel 2.4.3 wurde die Distribuierung eines nicht vollständig gelösten Problemes beschrieben, wie sie in Oz stattfindet. Dabei existieren Constraintvariablen, deren Wertebereiche mehrere mögliche Werte enthalten. Können diese Bereiche durch Propagierung nicht weiter eingeschränkt werden, ist der Berechnungsraum, in dem sich die Constraintvariablen befinden, stabil.

Bei der Distribuierung eines Problemes bestehen zwei Wahlmöglichkeiten:

1. die Wahl einer Variable Var aus der Menge der Variablen mit mehreren möglichen Werten und

2. die Wahl eines Wertes Val aus der Wertemenge der Variable Var.

Eine einfache Strategie, die bei vielen Anwendungen zu schnellen Ergebnissen führt, wird *first–fail*–Strategie genannt. Dabei wird die Variable mit der geringsten Anzahl von möglichen Werten ausgesucht. Dieser Variable wird ein Wert (meist der kleinste) ihres Wertebereiches zugewiesen. Zum Erreichen eines stabilen Berechnungsraumes wird erneut eine Propagierung durchgeführt. Diese Strategie hat jedoch auf Grund der Komplexität des Beispiels zu keiner Lösung innerhalb eines anvisierten Zeitrahmens geführt.

Als zweites wurde die sogenannte *first–fit*–Strategie ausprobiert. Sie versucht, eine Unterrichtseinheit zuerst an dem Tag zu planen, an dem bisher am wenigsten

[3] Slot `time` ist das Äquivalent zu `start` aus OCT

Einheiten geplant wurden. Auch diese Strategie scheiterte an der enormen Berechnungszeit.

Das Scheitern der beiden gebräuchlichen Distribuierungsstrategien liegt daran, daß das Setzen von neuen Unterrichtseinheiten in den Plan kompakt erfolgt. D.h., es wird immer erst versucht, die Einheiten z.B. vorrangig am Wochenanfang (Montag bis Mittwoch) zu setzen, wobei die Tage am Ende der Woche erst sehr spät Beachtung finden (*first–fail*). Es wird also versucht, einen kompakten Plan zu erzeugen, bei dem sehr viele Unterrichtseinheiten an den ersten Tagen des Planungszeitraumes gesetzt werden. Solche Pläne sind jedoch i.allg. nicht praktikabel.
Da die Unterrichtseinheiten relativ gleichmäßig über die Woche verteilt sind, wurden die Tage in zwei Blöcke aufgeteilt; es existiert jeweils einer für den Vormittag und den Nachmittag. Für jede Unterrichtseinheit kann eine individuelle Reihenfolge der 10 Blöcke festgelegt werden, die angibt, welcher Block den anderen gegenüber zu bevorzugen ist. Durch eine sorgfältige Auswahl der Blöcke konnte eine erste Lösung sehr schnell gefunden werden.

StuPlan verfolgt dagegen eine andere Distribuierungsstrategie. Der Unterrichtseinheit mit dem kleinsten Wertebereich wird eine Zeiteinheit zugewiesen, die sich besonders früh an einem Tag befindet. Dabei ist es nicht wichtig, welcher Tag das ist. Stellen mehrere Tage die gleiche früheste Zeiteinheit zur Verfügung, wird der erste dieser Tage gewählt.

Optimierung

Für das Stundenplanproblem existieren i.allg. mehrere Lösungen, von denen eine optimale gefunden werden soll. Nachdem eine erste Lösung des Problemes existiert, wird eine andere Lösung gesucht, die anhand eines Optimierungskriteriums als besser bewertet werden kann.

Diese Aufgabe erfüllt der *Branch–and–Bound*–Algorithmus (siehe Algorithmus 3.6). Ausgehend von der aktuellen Lösung wird das zu lösende Problem neu beschrieben. Zusätzlich zur allgemeinen Beschreibung des Problemes fließt eine Ordnungsrelation mit ein. Diese beschreibt eine bessere Lösung bezüglich der aktuellen. Wird mit der neuen Problembeschreibung keine bessere Lösung gefunden, ist die aktuelle die optimale Lösung. Ansonsten wird die Suche fortgesetzt, wobei jede neu gefundene Lösung zur aktuellen wird.

Als Optimierungskriterium dient in OCT die Minimierung der Unterrichtseinheiten, die nachmittags stattfinden. Die Suche nach einer optimalen Lösung ist wiederum durch die Komplexität der Aufgabenstellung und der daraus resultierenden Berechnungszeit nicht machbar. Aus diesem Grund kann die Suche jederzeit unterbrochen werden. Anhand der aktuellen Lösung kann entschieden werden, ob man mit dieser einverstanden ist oder nicht. Die Suche kann beliebig oft fortgesetzt werden.

Im Gegensatz zu OCT wird in StuPlan die Anzahl der Freistunden der Klassen und Lehrer durch ein Multiagentensystem minimiert. Das Kriterium, daß der Unterricht eher am Vormittag stattfinden soll, wird durch die Distribuierungsstrategie von StuPlan erreicht.

Fazit

OCT stellt einen guten Ansatz dar, das Stundenplanungsproblem mit Hilfe von Constraints zu lösen. Gerade der *Branch–and–Bound*–Algorithmus stellt ein mächtiges Werkzeug zur Optimierung dar. Fragwürdig erscheint hingegen die Distribuierungsstrategie. In der Praxis ist es sicherlich möglich, einer Unterrichtseinheit bevorzugte Zeiten zuzuweisen, jedoch ist das relativ willkürlich und subjektiv.

Algorithmus 3.6 Branch–and–Bound

```
fun {BranchAndBound Problem Order}
   fun {BAB Problem LastSol}
      case {SearchOne Problem}
      of [AktSol] then
         {BAB proc {$ BetterSol}
                 {Problem BetterSol}
                 {Order AktSol BetterSol}
              end [AktSol]}
      else LastSol
      end
   end
in
   {BAB Problem nil}
end
```

Kapitel 4

Ergebnisse und Ausblick

4.1 Zusammenfassung der Ergebnisse

Die Optimierung des Beispieles aus Anhang A erfolgt in 9 Runden. In jeder dieser Runden haben die 58 Agenten (21 Lehrer, 37 Gruppen) jeweils einmal die Möglichkeit, den Plan zu verändern. In der 9. Runde werden keine Veränderungen vorgenommen, so daß die Optimierung danach endet.

Die Parameter der Optimierung besitzen dabei die folgenden Werte:

- Die Temperatur beträgt am Anfang der Optimierung 6 und wird in jeder Runde um 1 verringert, bis sie ab der 7. Runde bei 0 liegt.

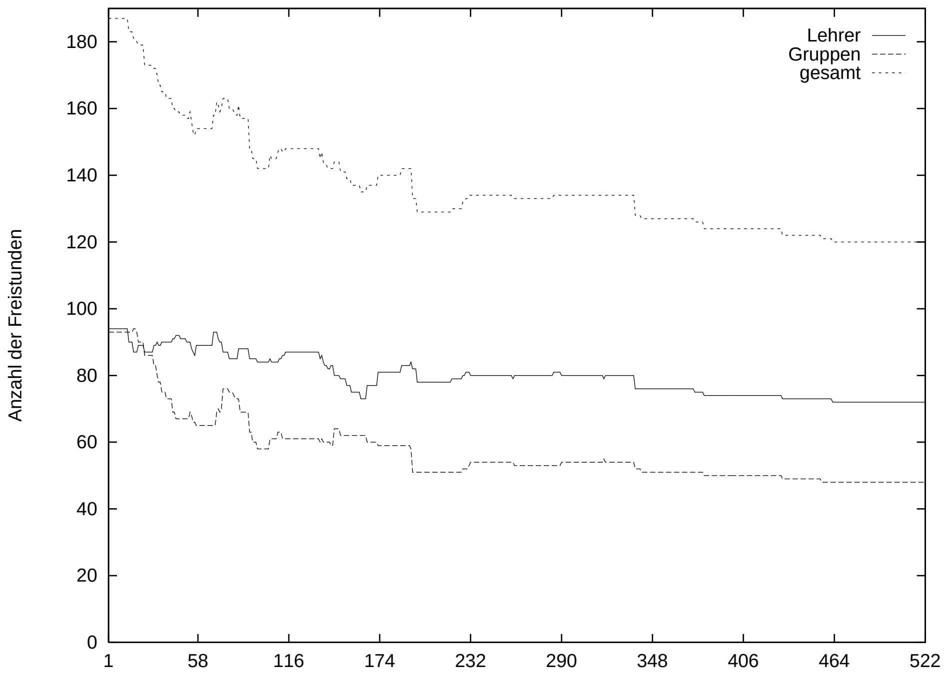

Abbildung 4.1: Anzahl der Freistunden im Verlauf der Optimierung

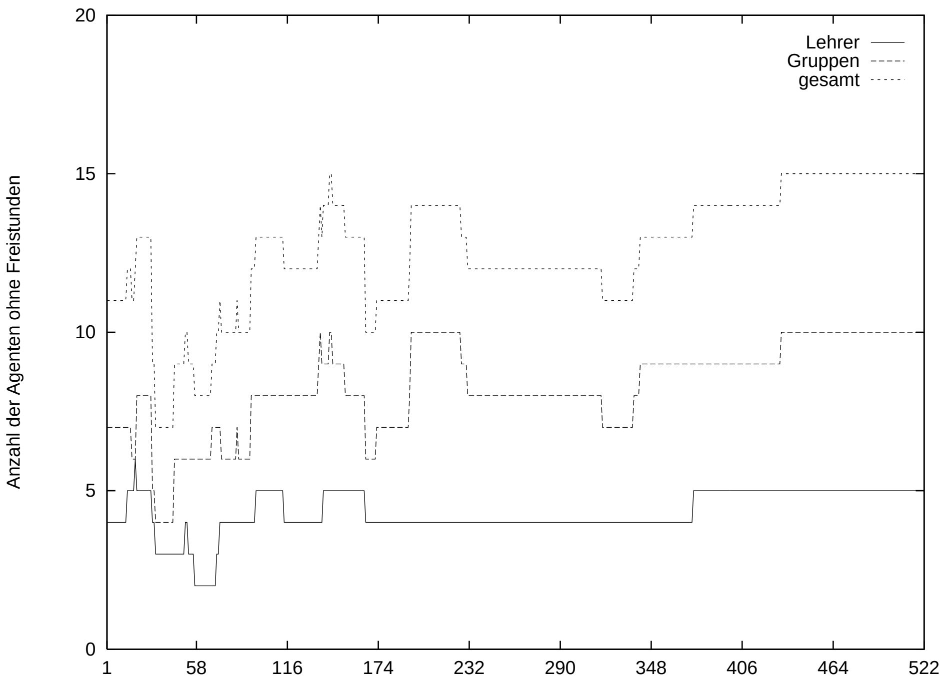

Abbildung 4.2: Anzahl der Agenten ohne Freistunden im Verlauf der Optimierung

- Die Anzahl hintereinanderliegender Versuche, eine Planveränderung vorzuschlagen, ist auf einen Versuch beschränkt.
- Die Zeitdauer bis zum Timeout beträgt 5 Minuten. Wird in dieser Zeit keine Planveränderung gefunden, beendet der Agent seine Bemühungen.

Während der Optimierung verringert sich die Anzahl der Freistunden aller Agenten von 187 auf 120. Abbildung 4.1 zeigt den Verlauf der Optimierung jeweils für alle Agenten, Lehrer und Gruppen.
Ein zweites Kriterium, die Anzahl lückenloser Agenten, wird in Abbildung 4.2 dargestellt.

Im folgenden werden die Pläne einiger Agenten jeweils vor und nach der Optimierung vorgestellt. Die dazugehörigen Stundenpläne sind ab Seite 72 abbgebildet.

Lehrer *Eb*

Der Ausgangsplan weist 9 Freistunden auf. Die Unterrichtseinheiten sind zwischen der ersten und der achten Zeiteinheit verteilt. Die beiden längsten Unterrichtstage dauern jeweils 7 Stunden.
Nach der Optimierung liegt ein kompakterer Plan vor. Die meisten Unterrichtseinheiten hängen zusammen, so daß nur noch zwei Freistunden existieren. Die letzte Unterrichtseinheit findet auch in der 8. Stunde statt, die früheste Einheit liegt jedoch erst in der 3. Stunde. Der längste Unterrichtstag dauert 6 Stunden.
Die Verteilung der Unterrichtseinheiten ist nach der Optimierung des Planes geringfügig schlechter. Es gibt zwei Tage, an denen jeweils nur eine Einheit stattfindet.

Lehrer *Loh*

Hier wirkt die Optimierung gerade umgekehrt. Der Endplan enthält mehr Freistunden und während der 8. Stunde findet jetzt auch Unterricht statt. Lediglich die Verteilung der Einheiten ist besser geworden.
Der optimierte Plan kann trotz der vielen Freistunden auch als verbesserter Plan angesehen werden. Auf Grund der vielen Unterrichtseinheiten, die der Lehrer unterrichtet, lockern die Freistunden den Tagesplan auf. Vor der Optimierung war der Plan so kompakt, daß donnerstags 7 Einheiten hintereinander zu halten waren.

Lehrer *Wei*

Der durch die Optimierung entstandene Plan kann als sehr gut angesehen werden. Durch nur 3 Unterrichtsverschiebungen verschwinden alle 6 Freistunden aus dem Plan. Er wird dadurch kompakt und relativ gleichverteilt.

Klasse 6b

Bei Klasse 6b ist zu beachten, daß die Mädchen und Jungen getrennt voneinander Sportunterricht haben. Die Zeiten dieses Unterrichtes sind fest vorgegeben. Optimal ist es, wenn nach diesen Einheiten kein weiterer Unterricht geplant wird.

Der Ausgangsplan zwingt die Mädchen dienstags zu einer Freistunde und die Jungen freitags zu zwei. Die Freistunde am Dienstag konnte durch die Optimierung entfernt werden, so daß die Mädchen keine Freistunde haben. Dadurch verschlechterte sich jedoch die Gleichverteilung der Stunden geringfügig.
Die Freistunden der Jungen bleiben bestehen, da eine Physikstunde nach dem Sportunterricht der Mädchen stattfindet. Diese Stunde wurde durch die Optimierung nicht versetzt. Abhilfe könnte dadurch geschaffen werden, daß der Unterricht der Klasse 6b täglich auf die ersten 7 Stunden beschränkt wird.

Klasse 7r

Bei Ausbildungsprofilen mit unterschiedlichem Lehrangebot in einer Klasse bieten sich folgende Möglichkeiten an, um einen guten Plan für die gesamte Klasse zu erreichen:

1. Der spezielle Unterricht der Profile findet gleichzeitig statt. Dadurch entsteht zu dieser Zeit für keine Gruppe eine Lücke. Vor allem bei unterschiedlichen Ressourcen (verschiedene Lehrer, Räume) bietet sich diese Möglichkeit an.

2. Der Unterricht, der nicht für die gesamte Klasse zu planen ist, findet in den Randstunden (als erste oder als letzte Einheiten am Tag) statt. Dadurch beginnt und endet der Unterricht für die unterschiedlichen Gruppen zu verschiedenen Zeiten am Tag. Diese Herangehensweise bietet sich vor allem dann an, wenn sich verschiedene Gruppen gleiche Ressourcen teilen. Das trifft im Beispiel auf den Sportunterricht zu, der immer in der Sporthalle stattfindet.

Bei Klasse 7r wird der Wirtschaftsunterricht dadurch zu einer Randstunde, daß die folgende Einheit versetzt wird. Am Mittwoch nachmittag führen jedoch die zwei

hintereinanderliegenden Einheiten verschiedener Gruppen zu Problemen. Männliche Schüler mit Wirtschaftsprofil haben hier eine Lücke.
Die Möglichkeit der mehrfachen Vergabe einer Zeiteinheit an verschiedene Gruppen wird am Freitag vormittags praktiziert. Trotzdem verbleibt für die Wirtschaftler eine Freistunde, die die Techniker als Wegeeinheit nutzen.

Klasse 8r

Von den bei Klasse 7r beschriebenen Möglichkeiten der Lückenvermeidung wird hier nur die zweite genutzt. Von den 9 Einheiten, die nicht im gesamten Klassenverband unterrichtet werden, ist anfangs lediglich eine eine Randstunde. Nach der Optimierung sind 6 dieser Einheiten so am Anfang oder am Ende der Unterrichtstage plaziert, daß sie keine Freistunden erzeugen.

Klasse 9m

Obwohl der Ausgangsplan relativ kompakt erscheint, enthält er insgesamt 6 Freistunden. Davon entfallen jeweils 3 auf die Schüler mit technischem Profil und mit wirtschaftlichem Profil. Jeder Schüler hat somit 3 Freistunden, für die beiden Gruppen werden aber insgesamt 6 Lücken gezählt.
Nach der Optimierung haben die Wirtschaftler einen lückenlosen Plan, die Techniker hingegen noch 2 Freistunden. Die Gesamtanzahl an Lücken hat sich dadurch um 4 verringert.

4.2 Bewertung der Ergebnisse

Die durch das Planungssystem StuPlan erzielten Ergebnisse stellen eine gute Optimierung dar.

Werden die Einzelpläne der Agenten betrachtet, so haben sich nur sehr wenige Agenten verschlechtert. Ein paar haben einen gleich guten Plan wie vor der Optimierung. Die meisten Agenten können jedoch eine Verbesserung ihrer Pläne aufweisen.

Problematisch erscheint dagegen noch immer der Zeitbedarf für die Optimierung. Während mit den Beispieldaten ein erster Plan innerhalb weniger Minuten berechnet wird, benötigt die Optimierung mehrere Stunden[1]. Zum einen liegt das an der Komplexität des Stundenplanungsproblemes, zum anderen an der dieser Komplexität noch nicht in ausreichendem Maße gewachsenen Rechentechnik.

4.3 Ausblick auf zukünftige Entwicklungen

Die Bemühungen um die Lösung und Optimierung des Stundenplanungsproblemes können mit dieser Arbeit noch nicht abgeschlossen sein. Dafür ist das Problem zu komplex.

Die weitere Fortsetzung der Aufgabe erscheint somit sinnvoll. Zum einen kann das

[1] ein 486er PC rechnete 58 Stunden

Planungssystem StuPlan weiter ausgebaut werden, zum anderen können die zugrundeliegenden Konzepte im Planungssystem weiter verbessert werden. Vor allem das Multiagentensystem bietet dafür Potential, wenn in Zukunft mit leistungsfähigeren Rechnern gearbeitet werden kann. Ein reflexiverer Systemansatz verspricht eine größere Individualität der Agenten und damit an die Agenten angepaßtere Einzelpläne.

Für eine größere Transparenz gegenüber praxisorientierten Aufgabenstellungen sollten folgende Punkte in das System eingearbeitet werden:

- Planung von Veranstaltungen, die im Zwei–Wochen–Rythmus stattfinden
- Beachtung von Wegzeiten zwischen hintereinander liegenden Unterrichtseinheiten, die in voneinander entfernten Gebäuden stattfinden
- dynamisches Eingreifen in den Optimierungsprozeß
- komfortable Benutzeroberfläche für die Ein– und Ausgabe der Daten sowie zur Steuerung der Optimierung

Darüberhinaus besteht die Möglichkeit, am Beispiel der Stundenplanung weitere Konzepte der KI zu testen. Naheliegend ist der Einsatz von Fallbasiertem Schließen. Liegen einer Schule ihre früheren Pläne vor, so können Teile der Lösung des aktuell zu lösenden Planes aus diesen übernommen werden. Die Berechnungszeit eines optimalen Stundenplanes liese sich damit sicher verringern.

In den folgenden Pläne gelten die Abkürzungen:

- **m/w** ... Gruppen mit männlichen bzw. weiblichen Schülern
- **T/W/H** ... Gruppen mit technischem, wirtschaftlichem bzw. hauswirtschaftlichem Profil
- **GK** ... Gesellschaftskunde
- **Info** ... Informatik
- **Weg** ... eingeplante Freistunde für den Rückweg zur Schule

---

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15			**Info** 9w		**Info** 7r (W)
8.20 – 9.05			**Info** 9w	**Info** 9m (W)	
9.25 – 10.10	**Info** 10w				
10.15 – 11.00	**Info** 10w			**GK** 10w	
11.10 – 11.55			**GK** 10t	**Werken** 5b (1/2)	
12.00 – 12.45	**GK** 9m	**GK** 10t		**Werken** 5b (1/2)	**GK** 9m
13.15 – 14.00	**GK** 8h				**GK** 10w
14.05 – 14.50				**GK** 8r	

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15					
8.20 – 9.05					
9.25 – 10.10	**Info** 10w				**Info** 7r (W)
10.15 – 11.00	**Info** 10w			**Info** 9m (W)	**GK** 9m
11.10 – 11.55			**GK** 10w	**Werken** 5b (1/2)	**GK** 10t
12.00 – 12.45	**GK** 9m			**Werken** 5b (1/2)	**GK** 10w
13.15 – 14.00		**GK** 8r		**Info** 9w	**GK** 8h
14.05 – 14.50	**GK** 10t			**Info** 9w	

Tabelle 4.1: Stundenpläne von Lehrer *Eb* vor und nach der Optimierung

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15			**Hausw.** 8 (H)	**Wirt.** 9m (W)	**Hausw.** 7wh (H)
8.20 – 9.05		**Wirt.** 9w	**Wirt.** 7r (W)	**Wirt.** 10w	**Wirt.** 7wt/wh (W)
9.25 – 10.10	**Hausw.** 8 (H)	**Wirt.** 10w	**Wirt.** 10w	**Wirt.** 9w	**Hausw.** 8 (H)
10.15 – 11.00	**Wirt.** 9w	**Wirt.** 8r (W)	**Hausw.** 7wh (H)	**Hausw.** 8 (H)	**Wirt.** 7r (W)
11.10 – 11.55		**Hausw.** 7wh (H)	**Wirt.** 7wt/wh (W)	**Wirt.** 8r (W)	**Wirt.** 9m (W)
12.00 – 12.45	**Hausw.** 7wh (H)	**Wirt.** 9m (W)	**Wirt.** 8r (W)	**Wirt.** 7wt/wh (W)	
13.15 – 14.00				**Wirt.** 7r (W)	
14.05 – 14.50					

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**Wirt.** 10w	**Wirt.** 9m (W)	**Wirt.** 8r (W)	**Hausw.** 8 (H)	**Hausw.** 7wh (H)
8.20 – 9.05	**Wirt.** 9m (W)			**Wirt.** 8r (W)	**Wirt.** 7wt/wh (W)
9.25 – 10.10		**Wirt.** 10w	**Wirt.** 9w	**Wirt.** 10w	**Wirt.** 9w
10.15 – 11.00	**Wirt.** 9w	**Wirt.** 8r (W)	**Hausw.** 8 (H)		**Wirt.** 7r (W)
11.10 – 11.55					
12.00 – 12.45	**Hausw.** 7wh (H)	**Wirt.** 7wt/wh (W)	**Hausw.** 7wh (H)		
13.15 – 14.00				**Wirt.** 7r (W)	**Wirt.** 9m (W)
14.05 – 14.50	**Wirt.** 7wt/wh (W)	**Hausw.** 8 (H)	**Wirt.** 7r (W)	**Hausw.** 7wh (H)	**Hausw.** 8 (H)

Tabelle 4.2: Stundenpläne von Lehrer *Loh* vor und nach der Optimierung

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15		**Mathe** 5a	**Mathe** 5a	**Mathe** 5a	
8.20 – 9.05	**Mathe** 5a			**Mathe** 10t	**Mathe** 5a
9.25 – 10.10	**Mathe** 10t				**Geo** 5a
10.15 – 11.00			**Mathe** 10t		**Mathe** 10t
11.10 – 11.55					
12.00 – 12.45					
13.15 – 14.00				**Geo** 5a	
14.05 – 14.50					

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15			**Mathe** 5a	**Mathe** 5a	
8.20 – 9.05	**Mathe** 5a		**Geo** 5a	**Mathe** 10t	**Mathe** 5a
9.25 – 10.10	**Mathe** 10t	**Mathe** 5a	**Mathe** 10t		**Geo** 5a
10.15 – 11.00					**Mathe** 10t
11.10 – 11.55					
12.00 – 12.45					
13.15 – 14.00					
14.05 – 14.50					

Tabelle 4.3: Stundenpläne von Lehrer *Wei* vor und nach der Optimierung

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**Mathe**	**Mathe**	**Engl**	**Engl**	**Mathe**
8.20 – 9.05	**Engl**	**Engl**	**Deu**	**Mathe**	**Engl**
9.25 – 10.10	**Bio**	**Deu**	**Mathe**	**Deu**	**Deu**
10.15 – 11.00	**Deu**	**Musik**	**Ges**	**Geo**	**Geo**
11.10 – 11.55	**Werken**	**Bio**	**Phy**	**Ethik**	**Ges**
12.00 – 12.45	**Werken**		**Kunst**	**m: Sport**	**w: Sport**
13.15 – 14.00	**m: Sport**	**w: Sport**		**m: Sport**	**w: Sport**
14.05 – 14.50					**Phy**

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**Kunst**	**Mathe**	**Mathe**	**Engl**	**Deu**
8.20 – 9.05	**Engl**	**Engl**	**Deu**	**Mathe**	**Mathe**
9.25 – 10.10	**Mathe**	**Bio**	**Engl**	**Deu**	**Engl**
10.15 – 11.00	**Deu**	**Musik**	**Bio**	**Phy**	**Ges**
11.10 – 11.55	**Werken**	**Ges**	**Geo**	**Geo**	**Ethik**
12.00 – 12.45	**Werken**	**Deu**		**m: Sport**	**w: Sport**
13.15 – 14.00	**m: Sport**	**w: Sport**		**m: Sport**	**w: Sport**
14.05 – 14.50					**Phy**

Tabelle 4.4: Stundenpläne der Klasse 6b vor und nach der Optimierung

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**w: Sport**	**m: Sport**	**Deu**	**m: Sport**	**T: Technik** **W: Info**
8.20 – 9.05	**w: Sport**	**Deu**	**W: Wirt.**	**m: Sport**	**T: Technik**
9.25 – 10.10	**Engl**	**Phy**	**Mathe**	**Mathe**	**T: Technik**
10.15 – 11.00	**Mathe**	**Mathe**	**Engl**	**Engl**	**T: Technik** **W: Wirt.**
11.10 – 11.55	**Deu**	**Engl**	**Geo**	**Deu**	**T: Weg**
12.00 – 12.45	**Phy**	**Bio**	**Bio**	**Ges**	**Ges**
13.15 – 14.00	**Musik**	**Geo**	**w: Sport**	**W: Wirt.**	**Musik**
14.05 – 14.50				**Ethik**	**Kunst**

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**w: Sport**	**m: Sport**	**Deu**	**m: Sport**	**T: Technik**
8.20 – 9.05	**w: Sport**	**Deu**	**Mathe**	**m: Sport**	**T: Technik**
9.25 – 10.10	**Engl**	**Musik**	**Bio**	**Mathe**	**T: Technik** **W: Info**
10.15 – 11.00	**Mathe**	**Mathe**	**Engl**	**Deu**	**T: Technik** **W: Wirt.**
11.10 – 11.55	**Geo**	**Engl**	**Phy**	**Ethik**	**T: Weg**
12.00 – 12.45	**Deu**	**Geo**	**Ges**	**Kunst**	**Engl**
13.15 – 14.00	**Phy**	**Ges**	**w: Sport**	**W: Wirt.**	**Musik**
14.05 – 14.50			**W: Wirt.**		**Bio**

Tabelle 4.5: Stundenpläne der Klasse 7r vor und nach der Optimierung

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**Deu**	**Deu**	**H: Hausw.**	**Mathe**	**Mathe**
8.20 – 9.05	**Mathe**	**Sport**	**Mathe**	**Deu**	**Deu**
9.25 – 10.10	**H: Hausw.**	**Sport**	**W: Info**	**Sport**	**H: Hausw.**
10.15 – 11.00	**Engl**	**W: Wirt.**	**W: Info**	**H: Hausw.**	**Phy**
11.10 – 11.55	**Ges**	**Engl**	**Engl**	**W: Wirt.**	**Engl**
12.00 – 12.45	**Che**	**Phy**	**W: Wirt.**	**Geo**	**Geo**
13.15 – 14.00	**Ethik**	**Musik**	**Kunst**	**Ges**	**Che**
14.05 – 14.50				**GK**	**Bio**

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**Deu**	**Deu**	**W: Wirt.**	**H: Hausw.**	**Deu**
8.20 – 9.05	**Mathe**	**Sport**	**Mathe**	**W: Wirt.**	**Phy**
9.25 – 10.10	**Kunst**	**Sport**	**Deu**	**Sport**	**Mathe**
10.15 – 11.00	**Engl**	**W: Wirt.**	**H: Hausw.**	**Mathe**	**Engl**
11.10 – 11.55	**Ges**	**Engl**	**Engl**	**Musik**	**Geo**
12.00 – 12.45	**Geo**	**Phy**	**W: Info**	**Ethik**	**Ges**
13.15 – 14.00	**Bio**	**GK**	**W: Info**	**Che**	**Che**
14.05 – 14.50		**H: Hausw.**			**H: Hausw.**

Tabelle 4.6: Stundenpläne der Klasse 8r vor und nach der Optimierung

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**Mathe**	**Mathe**	**Deu**	**T: Technik W: Wirt.**	**Sport**
8.20 – 9.05	**Deu**	**Deu**	**Phy**	**T: Technik W: Info**	**Sport**
9.25 – 10.10	**Ges**	**Che**	**Che**	**T: Technik**	**Engl**
10.15 – 11.00	**Ethik**	**Ges**	**Mathe**	**T: Weg**	**Mathe**
11.10 – 11.55	**Engl**	**Phy**	**Engl**	**Deu**	**W: Wirt.**
12.00 – 12.45	**GK**	**W: Wirt.**	**Sport**	**Engl**	**GK**
13.15 – 14.00		**Bio**	**Musik**	**Kunst**	
14.05 – 14.50					**Geo**

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**Mathe**	**W: Wirt.**	**Deu**	**T: Technik**	**Sport**
8.20 – 9.05	**W: Wirt.**	**Deu**	**Phy**	**T: Technik**	**Sport**
9.25 – 10.10	**Deu**	**Phy**	**Che**	**T: Technik**	**Mathe**
10.15 – 11.00	**Musik**	**Ges**	**Mathe**	**T: Weg W: Info**	**GK**
11.10 – 11.55	**Che**	**Mathe**	**Engl**	**Deu**	**Engl**
12.00 – 12.45	**GK**	**Engl**	**Sport**	**Engl**	**Bio**
13.15 – 14.00	**Ges**	**Ethik**		**Kunst**	**W: Wirt.**
14.05 – 14.50					**Geo**

Tabelle 4.7: Stundenpläne der Klasse 9m vor und nach der Optimierung

Anhang A

Ausgangsdaten des Beispieles

Die erzielten Resultate der Optimierung werden im Kapitel 4 anhand eines Beispieles verdeutlicht. Die zugrundeliegenden Daten einer Chemnitzer Mittelschule stammen aus dem Schuljahr 1995/96 und werden zum besseren Verständnis vorgestellt.

A.1 Allgemeine Daten

Der Stundenplan der Schule gliedert sich in fünf Tage (Montag bis Freitag) mit jeweils zehn Zeiteinheiten (7.30 bis 16.30 Uhr). Die frühen Zeiten sind bevorzugt zu belegen.

Einige Unterrichtseinheiten sind vorgegeben und können zeitlich wie räumlich nicht verändert werden, da sie von externen Bedingungen abhängig sind. Das betrifft den Sportunterricht sowie die Technikausbildung in einem Betrieb. Die Zeiten der vorgegebenen Einheiten sind in den Plänen A.2 und A.3 zusammengefaßt.
Die Klassen 5 und 6 haben in der Schule Werken, ab Klasse 7 findet der Technikunterricht als Block in einem Betrieb statt. Für die Klassen und den Lehrer ist jeweils eine Stunde für den Rückweg zur Schule einzuplanen.
Der Sportunterricht findet zu je einer Doppel– und einer Einzelstunde statt.

In Tabelle A.4 sind alle Veranstaltungen der Schule dargestellt. In jeder Zeile steht der Unterricht einer Klasse (unter dem Klassennamen befindet sich der Name des Klassenlehrers), jede Spalte enthält ein bestimmtes Unterrichtsfach. Die Kreuzfelder der Tabelle enthalten die Anzahl der Einheiten der Veranstaltung und den dazugehörigen Lehrer. Besonderheiten wie z.B. Doppelstunden sind gekennzeichnet.
Steht vor der Anzahl der Einheiten ein weiterer Zusatz, gibt dieser an, welche Gruppen einer Klasse diese Veranstaltung belegen.

Klassen	männl. weibl.	2 Hälften	Hauptschule	Realschule	Technik	Wirtschaft	Hauswirt.	Gruppen
5a		•						2
5b		•						2
6a	•							2
6b	•							2
7r	•			•	•	•		4
7wt	•		•	•	•	•		8
7wh	•		•	•		•	•	8
8r						•	•	2
8h					•		•	2
9m					•	•		2
9w						•		1
10w						•		1
10t					•			1

Tabelle A.1: Profile, Schularten, Ausrichtungen sowie Gruppenanzahl der Klassen

A.2 Klassen

Insgesamt gab es im betrachteten Schuljahr 13 Klassen, die sich u.a. auf Grund unterschiedlicher Unterrichtsprofile in 37 Gruppen aufteilen lassen (vgl. dazu Abschnitt 3.1.2). Tabelle A.1 zeigt Profile, Schularten und weitere Ausrichtungen, die in den Klassen vorkommen und unterschieden werden.

Klassenstufe 5

Die Klassen 5a und 5b haben maximal bis 12.45 Uhr Unterricht (1. bis 6. Zeiteinheit pro Tag).
Der Sportunterricht findet für die beiden Klassen jeweils im Klassenverband statt, so daß keine Unterteilung in Mädchen und Jungen vorgenommen wird. Für Klasse 5a findet Sport montags in der 5. und 6. Stunde sowie donnerstags in der 5. statt. Klasse 5b absolviert ihre Doppelstunde dienstags in der 5. und 6. sowie die Einzelstunde freitags in der 5. Stunde.
Beide Klassen werden in zwei Gruppen aufgeteilt, die getrennt voneinander den Werkunterricht durchführen. Der Unterricht findet für die Gruppen einer Klasse bei unterschiedlichen Lehrern statt. Bei ausreichender Raumkapazität (2 Werkräume) kann der Werkunterricht der beiden Gruppen einer Klasse zur gleichen Zeit unterrichtet werden.

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15	**Sport** Ri 7 (w)	**Sport** Hr3 7 (m)	**Sport** Ri/Hr3 10	**Sport** Hr3 7 (m)	**Sport** Ri/Hr3 9
8.20 – 9.05	**Sport** Ri 7 (w)	**Sport** Ri/Hr3 8	**Sport** Ri/Hr3 10	**Sport** Hr3 7 (m)	**Sport** Ri/Hr3 9
9.25 – 10.10		**Sport** Ri/Hr3 8		**Sport** Ri/Hr3 8	**Sport** Ri/Hr3 10
10.15 – 11.00					
11.10 – 11.55	**Sport** Ri 5a	**Sport** Ri 5b		**Sport** Ri 5a	**Sport** Ri 5b
12.00 – 12.45	**Sport** Ri 5a	**Sport** Ri 5b	**Sport** Ri/Hr3 9	**Sport** Stö 6 (m)	**Sport** Ri 6 (w)
13.15 – 14.00	**Sport** Sch 6 (m)	**Sport** Sch 6 (w)	**Sport** Ri 7 (w)	**Sport** Stö 6 (m)	**Sport** Ri 6 (w)

Tabelle A.2: Raumbelegungsplan der Sporthalle (m: Jungen-, w: Mädchengruppe)

Klassenstufe 6

Die Klassen 6a und 6b können bis 14.00 Uhr Unterricht haben. Das entspricht 7 Einheiten täglich.
Wie die Klassen 5 werden sie in je zwei Gruppen aufgeteilt. In dieser Klassenstufe wird jedoch jede Klasse für den Sportunterricht in Mädchen und Jungen unterteilt. Die Mädchen der beiden Klassen haben zusammen dienstags in der 7. sowie freitags in der 6. und 7. Unterrichtsstunde Sport, die Jungen in der 7. am Montag sowie in der 6. und 7. am Donnerstag.
Der Werkunterricht findet jeweils im Klassenverband statt.

Klassenstufe 7

Die Klassenstufe 7 besteht aus drei Klassen:

- 7r mit Realschülern des technischen und wirtschaftlichen Profiles;
- 7wt, besteht zusätzlich aus Hauptschülern der beiden Profile;
- 7wh mit Haupt– und Realschülern der Profile Wirtschaft und Hauswirtschaft.

Der Sportunterricht findet für alle Mädchen der drei Klassen montags in den ersten beiden und mittwochs in der 7. Stunde statt. Die Jungen werden dienstags in der ersten sowie donnerstags in den ersten beiden Stunden in Sport unterrichtet.
Die Schüler der Klassen 7r und 7wt mit technischem Profil haben freitags von der ersten bis zur vierten Stunde zusammen Technikausbildung in einem Betrieb. Anschließend ist eine Zeiteinheit für den Rückweg zur Schule zu planen.
Alle Schüler mit wirtschaftlichem Profil haben anstelle von Technik Informatik und Wirtschaft. Die Klassen 7wt und 7wh werden in beiden Unterrichtsfächern gemeinsam unterrichtet.
Die Hauptschüler der Klassen 7wt und 7wh haben gemeinsam Deutsch, Englisch, Mathematik und Physik. Gleiches gilt für die Realschüler der beiden Klassen. Dieser Unterricht der Haupt– und Realschüler muß nicht zwangsläufig zur gleichen Zeit stattfinden.

Klassenstufe 8

Die Schüler der Klasse 8r werden in die Profile Wirtschaft und Hauswirtschaft unterteilt, die Schüler der Klasse 8h in die Ausbildungsprofile Technik und Hauswirtschaft.
Am Unterrichtsfach Hauswirtschaft nehmen beide Klassen gemeinsam teil. Die Gruppe mit wirtschaftlichem Profil hat dagegen Informatik und Wirtschaft, die Gruppe mit technischem Profil wird mittwochs von der ersten bis zur vierten Stunde im Unterrichtsfach Technik ausgebildet (danach ist eine Zeiteinheit für den Weg einzuplanen).
Sportunterricht findet für alle Schüler der Klassenstufe 8 dienstags in der 2. und 3. sowie donnerstags zur 3. Stunde statt. Dafür stehen jeweils zwei Lehrer zur Verfügung.

Klassenstufe 9

Klasse 9m besteht aus Schülern mit technischem und wirtschaftlichem Profil. In Klasse 9w existiert nur ein wirtschaftliches Profil.
Die Techniker sind donnerstags bis zur dritten Stunde im Betrieb (danach ist eine Stunde für den Rückweg einzuplanen).
Der Sportunterricht ist wie in Klassenstufe 8 geregelt, jedoch zu anderen Zeiten: mittwochs in der 6. und freitags in den ersten beiden Stunden.

Klassenstufe 10

Klasse 10w besteht nur aus Schülern mit wirtschaftlichem, Klasse 10t nur aus Schülern mit technischem Profil.
Die Technikausbildung findet dienstags in den ersten beiden Unterrichtsstunden statt. Die anschließende Stunde ist für die Techniker freizuhalten (Rückweg).
Sportunterricht findet wie für die Klassenstufen 8 und 9 statt: mittwochs in den beiden ersten und freitags in der dritten Unterrichtsstunde.

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
7.30 – 8.15		**Technik** Rup 10t	**Technik** Rup 8h (T)	**Technik** Rup 9m (T)	**Technik** Rup 7r/7wt (T)
8.20 – 9.05		**Technik** Rup 10t	**Technik** Rup 8h (T)	**Technik** Rup 9m (T)	**Technik** Rup 7r/7wt (T)
9.25 – 10.10			**Technik** Rup 8h (T)	**Technik** Rup 9m (T)	**Technik** Rup 7r/7wt (T)
10.15 – 11.00			**Technik** Rup 8h (T)		**Technik** Rup 7r/7wt (T)

Tabelle A.3: Stundenplan des Betriebes (T: Gruppen mit Technikprofil)

A.3 Lehrer

Von den 21 Lehrern der Schule fällt Lehrer *Bau*[1] im betrachteten Schuljahr aus. D.h., die ihm zur Verfügung stehende Menge von Zeiteinheiten ist leer.

Für folgende Lehrer bestehen weitere Zeiteinschränkungen:

- die Sportlehrer *Ri*, *Hr3*, *Sch* und *Stö* haben jeweils vor und nach einer Anzahl aufeinanderfolgender Sportstunden eine Freistunde;
- Lehrer *Rup* hat nach dem Technikunterricht jeweils eine Freistunde für den Rückweg zur Schule.

A.4 Räume

Insgesamt stehen 19 Räumlichkeiten zur Verfügung, von denen nur das Informatikkabinett (Raum 1) Zeiteinschränkungen unterliegt. Es ist montags die ersten beiden sowie die 6. bis 8. Stunde, dienstags die letzten 6 und freitags die 5. bis 8. Stunde für Klassen einer anderen Schule zu reservieren.

In den einzelnen Räumen kann folgender Unterricht durchgeführt werden:

- Raum 1: Informatik;
- Raum 8: Physik, Astronomie;
- Raum 11: allg. Fächer[2], Wirtschaft;
- Raum 16: allg. Fächer;

[1] die Namen der Lehrer stehen nur als Kürzel zur Verfügung
[2] Ethik, Deutsch, Englisch, Geographie, Geschichte, Mathematik, Gesellschaftskunde

- Raum 17: allg. Fächer;
- Raum 18: allg. Fächer, Biologie;
- Raum 21: allg. Fächer, Musik;
- Raum 22: allg. Fächer, Kunst;
- Raum 23: allg. Fächer;
- Raum 24: allg. Fächer;
- Raum 25: allg. Fächer;
- Raum 26: allg. Fächer;
- Raum 27: allg. Fächer;
- Raum 28: allg. Fächer, Chemie;
- Sporthalle: Sport;
- Küche: Hauswirtschaft;
- Betrieb: Technik;
- Werkraum 1: Werken;
- Werkraum 2: Werken.

<table>
<tr><th></th><th>Ethik</th><th>Deu</th><th>Engl</th><th>Geo</th><th>Ges</th><th>Ma</th><th>Phy</th><th>Astro</th><th>Che</th><th>Bio</th><th>GK[3]</th><th>Sp (w)</th><th>Sp (m)</th><th>Mu</th><th>Ku</th><th>T/We</th><th>Info</th><th>Wi</th><th>HW</th></tr>
<tr><td>5a
Wei</td><td>1
Fri</td><td>5
Fri</td><td>5
Schu</td><td>2
Wei</td><td>1
Ho</td><td>5
Wei</td><td></td><td></td><td></td><td>2
Tu</td><td></td><td colspan="2">3
Ri[1]</td><td>2
Rie</td><td>2
Die</td><td>2 Fri[3]
2 Rup[3]</td><td></td><td></td><td></td></tr>
<tr><td>5b
Schu</td><td>1
Fri</td><td>5
Fri</td><td>5
Schu</td><td>2
Pau</td><td>1
Ho</td><td>5
Stö</td><td></td><td></td><td></td><td>2
Tu</td><td></td><td colspan="2">3
Ri[1]</td><td>2
Rie</td><td>2
Die</td><td>2 Eb[3]
2 Rup[3]</td><td></td><td></td><td></td></tr>
<tr><td>6a
Ott</td><td>1
Fri</td><td>5
Fri</td><td>5
Ott</td><td>2
Hr3</td><td>2
Ho</td><td>5
Sch</td><td>2
La</td><td></td><td></td><td>2
Tu</td><td></td><td>2
Ri[1]</td><td>2
Stö[1]</td><td>1
Rie</td><td>1
Die</td><td>2
Rup[1]</td><td></td><td></td><td></td></tr>
<tr><td>6b
Fri</td><td>1
Fri</td><td>5
Fri</td><td>5
Ott</td><td>2
Pau</td><td>2
Ho</td><td>5
Sch</td><td>2
La</td><td></td><td></td><td>2
Tu</td><td></td><td>1
Sch</td><td>1
Sch</td><td>1
Rie</td><td>1
Die</td><td>2
Rup[1]</td><td></td><td></td><td></td></tr>
<tr><td>7r
Stö</td><td>1
Fri</td><td>4
Rom</td><td>4
Ott</td><td>2
Pau</td><td>2
Ho</td><td>4
Stö</td><td>2
La</td><td></td><td></td><td>2
Tu</td><td></td><td rowspan="3">3
Ri[1]</td><td rowspan="3">3
Hr3[1]</td><td>2
Rie</td><td>1
Die</td><td rowspan="2">T: 4
Rup[2]</td><td>W: 1
Eb</td><td>W: 2
Loh</td><td></td></tr>
<tr><td>7wt
Rie</td><td>1
Bra</td><td>HS: 4
Rie</td><td>HS: 4
Schu</td><td>2
Pau</td><td>2
Ho</td><td>HS: 4
Ro</td><td>HS: 2
La</td><td></td><td></td><td>2
Tu</td><td></td><td>2
Rie</td><td>1
Die</td><td rowspan="2">W: 1
La</td><td rowspan="2">W: 3
Loh</td><td></td></tr>
<tr><td>7wh
Rom</td><td>1
Bra</td><td>RS: 4
Rom</td><td>RS: 4
Ott</td><td>2
Pau</td><td>2
Rom</td><td>RS: 4
Stö</td><td>RS: 2
La</td><td></td><td></td><td>2
Tu</td><td></td><td>1
Rie</td><td>1
Die</td><td></td><td></td></tr>
<tr><td>8r
Bau</td><td>1
Bra</td><td>4
Ho</td><td>4
Schu</td><td>2
Pau</td><td>2
Ho</td><td>4
Ro</td><td>2
La</td><td></td><td>2
Bra</td><td>1
Tu</td><td>1
Eb</td><td colspan="2" rowspan="2">3
Ri Hr3[1]</td><td>1
Rie</td><td>1
Die</td><td></td><td>W: 2
La</td><td>W: 3
Loh</td><td rowspan="2">H: 4
Loh</td></tr>
<tr><td>8h
Paul</td><td>1
Bra</td><td>4
Pau</td><td>4
Schu</td><td>2
Pau</td><td>2
Ho</td><td>4
Ro</td><td>2
La</td><td></td><td>2
Bra</td><td>1
Tu</td><td>1
Eb</td><td>1
Rie</td><td>1
Die</td><td>T: 4
Rup[2]</td><td></td><td></td></tr>
<tr><td>9m
Bra</td><td>1
Bra</td><td>4
Ho</td><td>4
Ott</td><td>1
Pau</td><td>2
Ho</td><td>4
Stö</td><td>2
La</td><td></td><td>2
Bra</td><td>1
Tu</td><td>2
Eb</td><td colspan="2" rowspan="2">3
Ri Hr3[1]</td><td>1
Rie</td><td>1
Die</td><td>T: 3
Rup[2]</td><td>W: 1
Eb</td><td>W: 3
Loh</td><td></td></tr>
<tr><td>9w
Loh</td><td>1
Bra</td><td>4
Rom</td><td>4
Schu</td><td>1
Pau</td><td>2
Hr3</td><td>4
Stö</td><td>2
La</td><td></td><td>2
Bra</td><td>1
Tu</td><td>2
Rom</td><td>1
Rie</td><td>1
Die</td><td></td><td>2
Eb</td><td>3
Loh</td><td></td></tr>
<tr><td>10w
Tur</td><td>1
Bra</td><td>4
Rom</td><td>3
Ott</td><td>1
Pau</td><td>2
Hr3</td><td>4
Ro</td><td>1
La</td><td>1
La</td><td>2
Bra</td><td>2
Tu</td><td>2
Eb</td><td colspan="2" rowspan="2">3
Ri Hr3[1]</td><td>1
Rie</td><td>1
Die</td><td></td><td>2
Eb</td><td>3
Loh</td><td></td></tr>
<tr><td>10t
Die</td><td>1
Bra</td><td>4
Die</td><td>3
Ott</td><td>1
Pau</td><td>2
Hr3</td><td>4
Wei</td><td>1
La</td><td>1
La</td><td>2
Bra</td><td>2
Tu</td><td>2
Eb</td><td>1
Rie</td><td>1
Die</td><td>2
Rup[2]</td><td>2
La</td><td></td><td></td></tr>
</table>

Tabelle A.4: Die Veranstaltungen des Testbeispiels (HS: Hauptschule, RS: Realschule, h/H: hauswirtschaftliches Profil, m: gemischtes Profil (wirt./techn.), t/T: technisches Profil, w/W: wirtschaftliches Profil; [1]: Doppelstunde, [2]: Block, [3]: Doppelstunde, halbe Klasse)

Anhang B

Thesen

1. Constraints dienen der effizienten Problembeschreibung und Suchraumeinschränkung.

2. Mit Finite–Domain–Constraints über dem Wertebereich der natürlichen Zahlen lassen sich die meisten praxisrelevanten Probleme beschreiben.

3. Constraints sind nicht geeignet für die Optimierung von Problemen mit sehr großem Suchraum und vielen Lösungen.

4. Die Programmierung von Constraints, deren Propagierung und die Problemdistribuierung sind in Oz effizient gelöst.

5. Durch den kontinuierlichen Anstieg von verteiltem Wissen werden Multiagentensysteme zunehmend an Bedeutung gewinnen.

6. Agenten, deren Arbeitsweise in hohem Maße an die des Menschen angelehnt ist, sind stark reflexiv. Abhängig von der Aufgabe müssen sie jedoch beim gegenwärtigen Stand der Technik so reaktiv wie möglich sein, um komplexe, kombinatorische Probleme zu beherrschen.

7. Der Kommunikationsaufwand in Multiagentensystemen ist zugunsten des Berechnungsaufwandes auf ein Minimum zu reduzieren.

8. Kann eine Aufgabe durch Multiagentensysteme nicht effizient bewältigt werden, sind andere problemlösende Methoden bzw. eine Kombination aus diesen und Multiagentensystemen vorzuziehen.

9. Um die verschiedenen Verhandlungspartner in einer Schule (Lehrer und Schüler) geeignet darzustellen, eignen sich Agenten in besonderem Maße.

Anhang C

Selbständigkeitserklärung

Ich erkläre, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Chemnitz, den 16. Dezember 1996

Literatur

[Anke96] Kay Anke. Test der Programmiersprache Oz bei der Lösung von Problemen der Künstlichen Intelligenz. Technische Universität Chemnitz–Zwickau, Fakultät für Informatik, Projektarbeit, 1996.

[Brot94] Sven Brotkorb. Entwurf und Implementierung einer Simulationsumgebung für ein Multiagentensystem zur Fertigungssteuerung. Technische Universität Chemnitz–Zwickau, Fakultät für Informatik, Diplomarbeit, 1994.

[Dilg94] Werner Dilger. Multiagentensysteme. Technische Universität Chemnitz–Zwickau, Fakultät für Informatik, Vorlesungsscript, 1994.

[Dilg95] Werner Dilger, Rainer Staudte, Kay Anke. Testing Oz Facilities in Distributed Problem Solving. WOz'95, International Workshop on Oz Programming, Martigny, Switzerland, 1995, pp. 45–49.

[Fisch93] Klaus Fischer. Verteiltes und kooperatives Planen in einer flexiblen Fertigungsumgebung. Infix, 1993.

[HeMü94] Martin Henz und Martin Müller. Programming in Oz. DFKI Oz Dokumentation, Deutsches Forschungsinstitut für Künstliche Intelligenz (DFKI), Stuhlsatzenhausweg 3, D–66123 Saarbrücken, 1994.

[HeWü94] Martin Henz und Jörg Würtz. Using Oz for College Time Tabling. Extended Abstract, DFKI Oz Dokumentation, Deutsches Forschungsinstitut für Künstliche Intelligenz (DFKI), Stuhlsatzenhausweg 3, D–66123 Saarbrücken, 1994.

[KI3/96] KI: Künstliche Intelligenz — Forschung, Entwicklung, Erfahrungen. Organ des Fachbereichs 1 „Künstliche Intelligenz“ der Gesellschaft für Informatik e.V. (GI), 1996, Heft 3, Seite 55ff.

[KMMP91] Hilger Kruse, Roland Mangold, Bernhard Mechler und Oliver Penger. Programmierung Neuronaler Netze. Addison–Wesley, 1991, Seite 169f.

[MPSW94] Tobias Müller, Konstantin Popow, Christian Schulte und Jörg Würtz. Constraint programming in Oz. DFKI Oz documentation series, Deutsches Forschungsinstitut für Künstliche Intelligenz (DFKI), Stuhlsatzenhausweg 3, D–66123 Saarbrücken, 1994.

[Schw95] Jens Schweitzer. Mehrstufige Constraint–Systeme. Technische Universität Chemnitz–Zwickau, Fakultät für Informatik, Diplomarbeit, 1995.

[SHG90] Eberhard Schöneburg, Nikolaus Hansen und Andreas Gawelczyk. Neuronale Netzwerke: Einführung, Überblick und Anwendungsmöglichkeiten. Markt–und–Technik–Verlag, 1990, Seite 118.

[Smol94] Gerd Smolka. An Oz primer. DFKI Oz Dokumentation, Deutsches Forschungsinstitut für Künstliche Intelligenz (DFKI), Stuhlsatzenhausweg 3, D–66123 Saarbrücken, 1994.