Über Methoden des constraintbasierten Lösens von Problemen der Stundenplanung*

Hans-Joachim Goltz
GMD – Forschungszentrum Informationstechnik GmbH
GMD-FIRST, Kekuléstr. 7, 12489 Berlin
e-mail: goltz@first.gmd.de

1 Einleitung

Ein Stundenplan ist eine Zuordnung von Ereignissen zu Zeitpunkten bzw. Zeitintervallen, so daß die geforderten Bedingungen (Constraints) erfüllt sind. Bei dieser Zuordnung sind problemspezifische Bedingungen (Constraints) zu beachten. Neben den Bedingungen, die immer erfüllt sein müssen und auch harte Constraints genannt werden, existieren auch Bedingungen, die möglichst erfüllt sein sollen und als weiche Constraints bezeichnet werden. Seit langem ist bekannt, daß das Stundenplanungsproblem zu der Klasse der NP-vollständigen Problemen gehört (siehe z.B. [4]). Für die automatische Stundenplanung, die ein aktuelles Forschungsgebiet ist, existieren verschiedene grundlegende Softwaretechniken und Ansätze (siehe z.B. [10]). Die constraintlogische Programmierung (CLP) konnte in vielen Anwendungen zur Lösung komplexer diskreter Probleme sehr erfolgreich eingesetzt werde. Verschiedene CLP-Ansätze werden beispielsweise in [1, 7, 8, 9, 11] diskutiert.
Ein Schwerpunkt unserer Forschungsarbeiten ist die Entwicklung von Methoden, Verfahren und Konzepten für eine interaktive, automatische Stundenplanung, die in Universitäten und Schulen angewendet werden können. Die Systeme der Stundenplanung sollen so flexibel sein, daß Besonderheiten der Anwender berücksichtigt und Constraints leicht geändert werden können, falls keine grundsätzliche konzeptionelle Änderung der Stundenplanung erforderlich ist. Die constraintlogische Programmierung bildet die Grundlage zur Verwirklichung unserer Forschungsziele. Für die Realisierung einer effizienten automatischen Lösungssuche ist neben der Verwendung geeigneter Suchstrategien die Problemmodellierung von entscheidender Bedeutung. Deswegen sind für uns die Entwicklung von Konzepten und Methoden der Modellierung von Problemen der Stundenplanung und Untersuchungen zum Einfluß verschiedener Problemmodellierungen auf die Lösungssuche wichtige Forschungsschwerpunkte. Auch die Entwicklung und der Vergleich verschiedener Lösungsstrategien ist ein Bestandteil unserer Forschungsarbeiten.
Im Rahmen unserer Forschungen entwickelten wir ein System für die interaktive, automatische Stundenplanung an der Medizinischen Fakultät Charité der Humboldt Universität zu Berlin. Als Implementationssprache wurde die constraintlogische Programmiersprache CHIP (Version 5.2) der Firma COSYTEC aus Frankreich gewählt. Auch die grafischen Schnittstellen wurden in CHIP implementiert. Neben den globalen Constraints erwies sich die objektorientierte Komponente von CHIP besonders vorteilhaft für die Implementation.
Die Stundenplanung der Charité wird seit Sommersemester 1998 mit unserem System erzeugt. Seitdem wurde das System kontinuierlich weiterentwickelt. Die Vorteile einer kombinierten interaktiven und automatischen Stundenplanerzeugung konnten eindeutig nachgewiesen wer-

*in *Proc. 14. Workshop Logische Programmierung*, F.Bry, U.Geske, D.Seipel (eds.), GMD Report 90 (ISBN 3-88457-971-1), Januar 2000, Seiten 77–88

den. Die erzeugten Pläne werden als HTML-Dateien ausgegeben und sind im Internet unter `http://www.first.gmd.de/plan/charite` zu finden.
Das Stundenplanungssystem besteht aus verschiedenen Komponenten. In [6] wird das Gesamtsystem vorgestellt. Auf ein Teilproblem, der Planung von Blockpraktika, wird dort nicht eingegangen. In dieser Arbeit diskutieren wir Methoden zum Lösen dieses Teilproblems. Im nächsten Abschnitt wird eine kurze Problembeschreibung gegeben. Im 3. Abschnitt wird die Modellierung der geforderten Bedingungen beschrieben. Anschließend werden mögliche redundante Constraints angegeben. Lösungsstrategien diskutieren wir im 5. Abschnitt. Für verschiedene Methoden und 6 Beispiele werden dann Laufzeitergebnisse der Lösungssuche dargestellt und ausgewertet.

2 Problembeschreibung

In der medizinischen Ausbildung müssen die Studenten relativ viele Pflichtveranstaltungen absolvieren. Neben den Vorlesungen eines Semesters finden Seminare, Untersuchungskurse und Praktika statt, die in Gruppen von bis zu 20 Studenten durchgeführt werden. Dazu werden die Studenten eines Semesters in Seminargruppen unterteilt. In bestimmten Studienabschnitten müssen die Studenten Praktika verschiedener Fächer absolvieren, die in der Regel jeweils an aufeinanderfolgenden Werktagen stattfinden, wobei die Anzahl der Tage bzw. Wochen vorgegeben ist. Diese Praktika werden als Blockpraktika bezeichnet.
Der gesamte Stundenplan besteht aus verschiedenen Komponenten. Für die Planung kann jede Komponente relativ unabhängig betrachtet werden, wobei aber eine bestimmte Reihenfolge einzuhalten ist. Zuerst sind die Vorlesungen für alle Semester zu planen. Anschließend können dann für jedes Semester die Lehrveranstaltungen geplant werden, die in Seminargruppen durchgeführt werden, wobei die relevanten Vorlesungszeiten als "Sperrzeiten" zu berücksichtigen sind. Die Planung der Blockpraktika kann völlig unabhängig erfolgen, da diese in vorgegebenen Wochen tageweise stattfinden.
Im folgenden betrachten wir nur die Planung von Blockpraktika. Diese Lehrveranstaltungen finden in einem vorgegebenen Zeitraum statt und werden in Gruppen von bis zu 20 Studenten durchgeführt. Die Dauer der Praktika ist jeweils vorgegeben. Bei der Planung von Blockprakika sind nur die Tage zu bestimmen, an denen sie stattfinden sollen. Die zugeordnete Tageszeit ist für die Planung nicht von Bedeutung. Ein gegebenes Praktikum ist von alle Gruppen oder einer angegebenen Menge von Gruppen zu absolvieren. Als *Praktikumseinheit* oder *Einheit eines Praktikums* bezeichnen wir im folgenden das Praktikum, das für eine bestimmte Gruppe stattfindet. Folglich kann jedem Praktikum eine Menge von *Praktikumseinheiten* zugeordnet werden.
Zwei Arten von Blockpraktika können unterschieden werden: *Wochen-Praktika* und *Tage-Praktika*. Den Wochen-Praktika sind eine bestimmte Anzahl von Wochen zuzuordnen. Wenn mehr als eine Woche zuzuordnen ist, müssen diese Wochen aufeinanderfolgen. Diese Praktika beginnen immer am ersten Werktag der Woche. Bei der Planung dieser Praktikumsart werden Feiertage ignoriert. Folglich kann die Anzahl der Tage, an denen diese Praktika tatsächlich stattfinden, unterschiedlich sein. Dagegen ist bei den Tage-Praktika diese Anzahl fest. Den Tage-Praktika ist eine bestimmte Anzahl von aufeinanderfolgenden Werktagen zuzuordnen, wobei freie Tage (Wochenende, Feiertage) nicht mitgezählt werden. Praktika dieser Art können an jedem Werktag beginnen. Es wird vorausgesetzt, daß es in jedem Zeitabschnitt (mit einer Länge, die der Dauer entspricht), in dem ein Wochen-Praktikum durchgeführt wer-

den kann, weniger als fünf Feiertage gibt (d.h., weniger als Praktikumstage einer Woche). Die wichtigsten Bedingungen für Blockpraktika sind außerdem:

1. die Praktikumseinheiten, die einer Gruppe zugeordnet sind, dürfen sich nicht überlappen;
2. die Anzahl der Einheiten eines Praktikums, die zur gleichen Zeit stattfinden dürfen, ist beschränkt; diese Schranke muß nicht für die gesamte Zeit gleich sein;
3. es können Praktika existieren, die erst nach der Absolvierung von anderen Praktika stattfinden können;
4. es können Praktika existieren, bei denen jeweils ein direkt nachfolgendes Praktikum vorgegeben ist;
5. einige Praktika dürfen an bestimmten Tagen nicht beginnen;
6. Für jeweils zwei Einheiten eines Praktikums kann die folgende Bedingung gefordert werden: Entweder sie beginnen am gleichen Tag oder sie dürfen sich nicht überlappen;
7. die Anzahl der Tage, an denen die Praktikumseinheiten eines Praktikums beginnen dürfen, kann als minimal oder maximal vorgegeben sein, wobei auch eine genaue Anzahl dieser Tage vorgegeben werden kann.

3 Modellierung der Bedingungen

Der Constraintlöser über endlichen Domänen von CHIP bildet die Grundlage unserer Problemmodellierung. Die gewählte Modellierung eines Problems der Stundenplanung besitzt einen großen Einfluß auf die Propagationseigenschaften des Constraintlösers und damit auf die Effizienz der Lösungssuche. Die Problemmodellierung ist auch abhängig von den Möglichkeiten und Propagationseigenschaften des gewählten Constraintlösers. Die in CHIP enthaltenen globalen Constraints haben sich für unsere Modellierung als sehr wertvoll erwiesen.
Für jede Gruppe, die ein gegebenes Blockpraktikum absolvieren muß, wird eine Praktikumseinheit definiert. Allen Praktikumseinheiten sind Zeiten so zuzuordnen, daß alle Bedingungen erfüllt sind. Da die Dauer eines Praktikums vorgegeben ist, genügt es, jeweils die *Startzeiten* der Praktikumseinheiten zu bestimmen. Die Startzeit wird als Domänenvariable betrachtet. Die möglichen Ausgangswerte einer Domäne ergeben sich aus der fortlaufenden Numerierierung aller Tage, an denen Praktika stattfinden können. Die Domäne der Startzeitvariable eines Wochen-Praktikas besteht anfangs nur aus den natürlichen Zahlen, die der Menge der ersten Werktage der gegebenen Wochen entsprechen.
Weil die Dauer der Wochen-Praktika in Abhängigkeit von der zugeordneten Startzeit durch mögliche Feiertage variieren kann, wird die Dauer von Praktikumseinheiten solcher Praktika als Domänenvariable betrachtet. Für die Modellierung der Beziehung zwischen Dauer und der Startzeitvariable kann das vordefinierte symbolische Constraint `element/3` verwendet werden. Zum Beispiel gilt `element(N,List,Value)` genau dann, wenn das `N`-te Element der nichtleeren Liste `List` natürlicher Zahlen gleich `Value` ist, wobei `N` und `Value` Domänenvariablen sein können.

Die in der Problembeschreibung genannten Bedingungen können direkt durch vordefinierte Constraints formuliert und vor der Lösungssuche erzeugt werden. Für die Formulierung der Bedingungen 1 und 2 kann das globale Constraint `cumulative` direkt verwendet werden.

Dieses Constraint sichert, daß von einer Ressource zu jedem Zeitpunkt nicht mehr als die angegebene Anzahl benötigt wird. Falls diese Schranke für den gesamten Zeitraum nicht gleichmäßig ist, können "Dummy"-Einheiten definiert werden, die zu den entsprechenden Zeiten Kapazitäten blockieren. Die Bedingungen 3 und 4 können einfach durch Gleichungen und Ungleichungen formuliert werden. Durch Streichen von Elementen aus den Domänen von Startzeitvariablen kann gesichert werden, daß die Bedingung 5 erfüllt ist.

Die Modellierung der Bedingungen 6 und 7, die etwas komplizierter ist, wird im folgenden etwas genauer betrachtet. Sei P ein Praktikum, bei dem die Praktikumseinheiten die Bedingung 6 erfüllen müssen. Jeweils 2 Einheiten dieses Praktikums müssen folglich genau parallel stattfinden oder sie dürfen sich nicht überlappen. $X_1, \ldots, X_n$ seien die Tage, an denen mindestens eine Praktikumseinheit beginnt. Somit existiert für jede Praktikumseinheit genau ein X_i, das mit der Startzeit dieser Einheit übereinstimmt. Ohne Beschränkung der Allgemeinheit kann vorausgesetzt werden, daß $X_i < X_j$ für $i < j$ gilt. Wenn D die Dauer der Praktikumseinheiten ist, gilt die Bedingung 6 genau dann, wenn für jedes $i, j \in \{1, \ldots, n\}$ mit $i < j$ die Ungleichung $X_i + D \leq X_j$ erfüllt ist. Wenn P ein Wochen-Praktikum ist, muß für D der minimalste Wert einer Dauer gewählt werden. Die Ungleichung gilt dann auch für diesen Fall, da eine Beschränkung der maximalen Anzahl von Feiertagen innerhalb eines Duchführungszeitraumes vorausgesetzt wurde. Diese Bedingungen und Zusammenhänge werden nun durch Constraints ausgedrückt.
Sei *Starts* die möglichen Startzeiten und n die Anzahl der möglichen verschiedenen Startzeiten der zugeordneten Praktikumseinheiten. Wenn $AnzT$ die Gesamtzahl der Tage ist, an denen überhaupt ein Praktikum durchgeführt werden kann, ist die Zahl n durch den Quotienten $AnzT/D$ beschränkt. Die möglichen verschiedenen Startzeiten $X_1, \ldots, X_n$ werden als Domänenvariablen mit der Domäne *Starts* definiert und für jedes $i, j \in \{1, \ldots, n\}$ mit $j = i + 1$ wird das Constraint $X_i + D \leq X_j$ erzeugt. Für jede Praktikumseinheit P_i des Praktikums P sei $Start_i$ die Startzeitvariable mit der Anfangsdomäne *Starts*. Um die Beziehungen zwischen den Startzeitvariablen und den Variablen $X_1, \ldots, X_n$ ausdrücken zu können, wird für jede Praktikumseinheit eine weitere Domänenvariable definiert, die mit *StartNr* bezeichnet und Startnummervariable genannt wird. Die Anfangsdomäne jeder dieser Variablen wird mit $1 \ldots n$ festgelegt. Für jede Praktikumseinheit P_i wird nun das folgende Constraint[2] formuliert:

$$\texttt{element}(StartNr_i, [X_1, \ldots, X_n], [0, \ldots, 0], Start_i, [\texttt{all,all,all,all}])$$

Dieses Constraint sichert, daß die Beziehung $Start_i = X_{StartNr_i}$ gilt. Das 3. Argument ist im allgemeinen eine Liste von natürlichen Zahlen der Länge n, durch die eine konstante Abweichung von dieser Gleichung ausgedrückt wird. In unserem Fall ist diese Abweichung jeweils gleich 0. Für die Kontrolle der Propagation wird das letzte Argument verwendet. Durch die gewählte Angabe, wird das Constraint immer dann aktiviert, wenn sich die Domäne einer der beteiligten Variablen ändert.

Für die Modellierung der 7. Bedingung verwenden wir die eingeführten Startnummervariablen. Somit werden auch die Constraints vorausgesetzt, die die Beziehungen zwischen den Startzeitvariablen und den Startnummervariablen charakterisieren. Falls ein Praktikum die Bedingung 6 nicht erfüllen muß, ist die Anzahl n der möglichen verschiedenen Startzeiten nicht durch den Quotienten $AnzT/D$ beschränkt und die Ungleichung $X_i + D \leq X_j$ ist durch die Ungleichung $X_i < X_j$ zu ersetzen. Wenn in der Bedingung 7 eine maximale Anzahl

[2] Das symbolischen Constraint `element/5` ist in der Version 5.2 von CHIP erstmalig enthalten.

verschiedener Startzeiten der Einheiten eines Praktikums gefordert ist, so kann dies durch die Wahl der Zahl n und somit durch die Wahl der Domänen für die Startnummervariable gesichert werden.
Wenn eine minimale Anzahl Min verschiedener Startzeiten für ein Praktikum P gefordert ist, bedeutet dies, daß die Menge $\{ StartNr_1, \ldots, StartNr_m \}$ mindestens Min verschiedene Elemente enthält, wobei m die Anzahl der Praktikumseinheiten von P sei. Zuerst bestimmen wir für jedes $k \in \{1, \ldots, n\}$ die Anzahl N_k, die aussagt, wie oft der Wert k in der Liste $[StartNr_1, \ldots, StartNr_m]$ vorkommt. Sei $MaxPar$ die Anzahl der Praktikumseinheiten von P, die höchstens parallel stattfinden dürfen, und für jedes $k \in \{1, \ldots, n\}$ sei N_k eine Domänenvariable mit der Domäne $0 \ldots MaxPar$. Für jedes $k \in \{1, \ldots, n\}$ wird folgendes `among`-Constraint erzeugt[3]:

$$\texttt{among}(N_k, [StartNr_1, \ldots, StartNr_m], [0, \ldots, 0], [k], \texttt{all})$$

Dieses Constraint gilt, wenn von den Variablen $StartNr_1, \ldots, StartNr_m$ genau N_k-vielen der Wert k zugeordnet wird. Das 3. Argument ist im allgemeinen eine Liste von natürlichen Zahlen der Länge m, durch die eine Abweichung vom Wert k für die entsprechende Variable ausgedrückt werden kann. In unserem Fall wird keine Abweichung erlaubt. Durch die Angabe "`all`" im letzten Argument wird dieses Constraint immer dann aktiviert, wenn sich die Domäne einer der relevanten Variablen ändert. Für die Domänenvariablen $N_1, \ldots, N_n$ muß außerdem die folgende Gleichung gelten: $N_1 + \ldots + N_n = m$. Durch ein weiteres `among`-Constraint kann nun ausgedrückt werden, wieviele der Domänenvariablen $N_1, \ldots, N_n$ höchstens den Wert 0 annehmen dürfen, wobei Y eine Domänenvariable mit der Domäne $0 \ldots (n - Min)$ ist: $\texttt{among}(Y, [N_1, \ldots, N_n], [0, \ldots, 0], [0], \texttt{all})$.

4 Redundante Constraints

Redundante Constraints sind zusätzliche Constraints, die logisch aus den anderen Constraints folgen. Solche Constraints werden hinzugefügt, um die Propagationseigenschaften zu verbessern. Folglich könnte dadurch der Suchraum reduziert werden. Durch Verwendung redundanter Constraints ist im allgemeinen aber eine Erhöhung der Rechenzeit für einen Suchschritt zu erwarten, wobei eine Verringerung des Suchraumes nicht garantiert werden kann. Für ein gegebenes Problem ist somit zu untersuchen, für welche redundante Constraints im Durchschnitt ein besseres Gesamtergebnis zu erwarten ist. Einige Beispiele redundanter Constraints für das gegebene Problem werden im folgenden beschrieben. Anschließend werden einige Testbeispiele mit verschiedenen Varianten des Hinzufügens redundanter Constraints angegeben und die Ergebnisse werden ausgewertet.
Die Anzahl der Einheiten eines Praktikums, die zur gleichen Zeit stattfinden dürfen, ist beschränkt (2. Bedingung). Diese Kapazitätsbeschränkung kann mittels des globalen Constraints `cumulative` auch bezüglich der Variablen *StartNr* eines Praktikums formuliert werden, falls diese Variablen relevant sind. Dieses Constraint folgt logisch aus den Beziehungen der Variablen *StartNr* und *Start* und aus der constraintlogischen Modellierung der 2. Bedingung bezüglich der Variablen *Start*. Im folgenden setzen wir voraus, daß für alle Praktika, für die *StartNr* eine Domänenvariable ist, dieses redundante Constraint erzeugt wird.

[3] In CHIP kann diese Menge von `among`-Constraints zu einem speziellen `among`-Constraint zusammengefaßt werden.

Die Beschränkung der Kapazität von Ressourcen kann auch mittels dem globalen Constraint `diffn` ausgedrückt werden. Durch dieses globale Constraint können abstrakte Bedingungen der folgenden Art direkt modelliert werden: *Eine Liste n-dimensionaler Rechtecke ist überlappungsfrei in einem gegebenen n-dimensionalen Rechteck zu plazieren.* Solche Constraints könnten auch die `cumulative`-Constraints ersetzen. In diesem Bericht werden wir nur die Möglichkeit diskutieren, dieses Constraint bei den Ressourceneinschränkungen für die erste und zweite Bedingung zusätzlich in allen Fällen zu verwenden, wobei jeweils 2-dimensionale Rechtecke betrachtet werden. Bei der Modellierung der ersten Bedingung ist die Zeit eine Dimension, wobei die Dauer die Länge des Rechtecks bestimmt. In der anderen Dimension unterscheiden sich die Rechtecke nicht (z.B. Länge gleich 1 und Plazierung bei 0). Für die zweite Bedingung können `diffn`-Constraints für die Variablen *Start* und für die Variablen *StartNr* erzeugt werden. Diese Variablen bestimmen auch die Werte in der einen Dimension. Die Werte der anderen Dimension werden durch die mögliche Anzahl paraller Einheiten bestimmt. Dazu wird jeder Einheit eine Parallelitätsnummer zugeordnet und die Länge des entsprechenden Rechtecks in dieser Dimension beträgt 1.
Durch `among`-Constraints kann indirekt ausgedrückt werden, wieviele verschiedene Werte mindestens eine Liste von Domänenvariablen annehmen muß. Dies ergibt sich aus der Modellierung der 7. Bedingung. Auch wenn für ein Praktikum keine minimale Anzahl verschiedener Startzeiten gefordert wird, können `among`-Constraints so erzeugt werden, wie im 3. Abschnitt beschrieben wurde, wobei auf das `among`-Constraint bezüglich $N_1, \ldots, N_n$ verzichtet wird. Das Constraint $N_1 + \ldots + N_n = m$ wird aber erzeugt. Analog könnten solche Constraints auch bezüglich den Startzeitvariablen erzeugt werden.
Für zwei beliebige Praktikumseinheiten P_i und P_j eines Praktikums können die Beziehungen zwischen den Startzeitvariablen und den Startnummervariablen zusätzlich durch folgende bedingte Constraints ausgedrückt werden:

`if` $Start_i < Start_j$ `then` $StartNr_i < StartNr_j$ `else` $StartNr_i \geq StartNr_j$
`if` $StartNr_i < StartNr_j$ `then` $Start_i < Start_j$ `else` $Start_i \geq Start_j$

Wenn alle Möglichkeiten dieser Art der Beziehungen betrachtet werden sollen, sind im allgemeinen eine relativ große Anzahl solcher bedingter Constraints zu erzeugen.

5 Lösungssuche

Constraintlöser über endlichen Domänen sind nicht vollständig. Für die Erzeugung einer Lösung ist i. allg. Suche erforderlich, die oft durch *"Labelling"* realisiert wird. Dabei werden schrittweise den Constraintvariablen Werte aus ihren Domänen zugeordnet. Als Modifizierung dieser Suchmethode haben wir in [5] vorgeschlagen, daß die Wertzuweisung durch eine Einschränkung der Domäne der ausgewählten Variable ersetzt wird. Diese Methode der backtrackbaren Domänenreduzierung ist eine Verallgemeinerung der Labelling-Methode. Die Domänenreduzierung wird auch bei einigen Strategien, die wir im folgenden betrachten, verwendet. In der Lösungssuche sind zwei Arten von Nichtdeterminismus enthalten: Auswahl einer Domänenvariable und Auswahl eines Wertes aus der relevanten Domäne bzw. Auswahl der Domäneneinschränkung für die ausgewählte Variable. Natürlich ist es i. allg. nicht möglich, alle Auswahlmöglichkeiten der Suche zu probieren. Für die Lösungssuche sind folglich Heuristiken erforderlich, die die jeweilige Auswahl unterstützen. In diesem Beitrag werden wir einige Varianten von unterschiedlichen Suchstrategien diskutieren.
Bei der Planung der Blockpraktika ist eine Menge von Praktikumseinheiten gegeben. Den

Domänenvariablen *Start* und *StartNr* jeder Praktikumseinheit müssen Werte aus ihren Domänen so zugeordnet werden, daß alle Constraints erfüllt sind. In einer Suchstratgie muß auch festgelegt werden, in welcher Reihenfolge diese beiden Arten von Variablen bei der Suche einbezogen werden. Wir betrachten nun die folgenden Suchstrategien, wobei eine sortierte Liste von Praktikumseinheiten als gegeben vorausgesetzt wird:

- S 1: für jede Einheit: Wertzuweisung von *Start* und gleich anschließend Wertzuweisung von *StartNr*;
- S 2: erst für alle Einheiten Wertzuweisung von *StartNr* und anschließend für alle Einheiten Wertzuweisung von *Start*;
- S 3: erst für alle Einheiten Wertzuweisung von *Start* und anschließend für alle Einheiten Wertzuweisung von *StartNr*;
- S 4: erst für alle Einheiten Domänenreduzierung von *Start* und anschließend für alle Einheiten Wertzuweisung von *StartNr* und von *Start*;
- S 5: für jede Einheit zuerst Domänenreduzierung von *Start* und dann gleich Wertzuweisung von *StartNr*, anschließend für alle Einheiten Wertzuweisung von *Start*;
- S 6: für jede Einheit zuerst Wertzuweisung von *StartNr* und dann gleich Domänenreduzierung von *Start*, anschließend für alle Einheiten Wertzuweisung von *Start*
- S 7: erst für alle Einheiten Domänenreduzierung von *Start* und dann für alle Einheiten Wertzuweisung von *StartNr*; und anschließend für alle Einheiten Wertzuweisung von *Start*.

Für die Sortierung aller Praktikumseinheiten bieten sich zunächst zwei Grundvarianten an: *Sortierung nach Gruppen* und *Sortierung nach Praktika*. Bei unseren Testversuchen erwies sich die 2. Variante als wesentlich besser. Deswegen werden wir hier folgendes Ordnungsprinzip voraussetzen. In der sortierten Liste der Praktikumseinheiten sind die Einheiten eines Praktikums zusammenhängend angeordnet, wobei sich die Reihenfolge durch die Priorität des Praktikums und durch die Gruppenzuordnung ergibt. Die Priorität der Praktika ist entscheidend für die Reihenfolge der Suche. Diese Reihenfolge besitzt natürlich einen großen Einfluß auf die Lösungssuche und kann auch über Erfolg oder Mißerfolg der Suche entscheiden. In diesem Bericht werden wir dies aber nicht genauer untersuchen. In den Beispielen, die im nächsten Abschnitt betrachtet werden, wird die Reihenfolge der Praktikumseinheiten für jedes Beispiel jeweils als gegeben und fest betrachtet.

6 Vergleich verschiedener Methoden

In diesem Abschnitt werden verschiedene Methoden an einigen Testergebnissen verglichen. Grundlage der Beispiele sind zwei praktische Probleme der Blockpraktikaplanung, die sich bei der Stundenplanung für das Sommersemester 1999 und für das Wintersemester 1999/2000 ergaben. Die erzeugten Pläne dieser Probleme sind auch unter der in der Einleitung gegebenen Internetaddresse zu finden. In den Abbildungen 1 und 2 sind diese Lösungen auch dargestellt. Wegen Feiertagen und Schulferien ist die ungleichmäßige Verteilung gewünscht. Bei beiden Problemen sind die Blockpraktika für das 5. klinische Semester zu planen. Diese Blockpraktika sind in 14 bzw. 16 Wochen zu absolvieren, wobei die gesamte Praktikumszeit die letzten 6 bzw. 8 Wochen des aktuellen Semesters und die ersten 8 Wochen des nächsten Semesters umfaßt.

Bei diesen beiden Problemen kann vorausgesetzt werden, daß alle Praktika Wochen-Praktika sind. Die beiden Probleme unterscheiden sich bezüglich der Anzahl der Gruppen und der Anzahl der erlaubten parallen Einheiten eines Praktikums. Für jedes dieser beiden Probleme betrachten wir im folgenden 3 Beispiele, die jeweils aus dem Problem abgeleitet wurden.

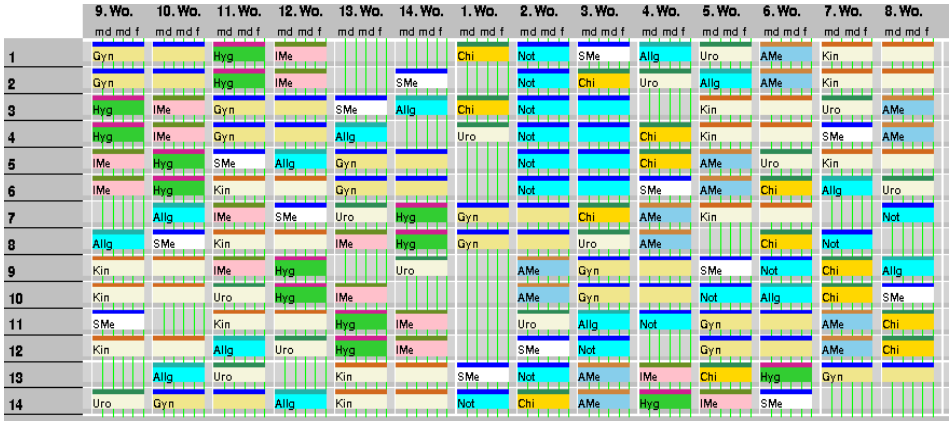

Abbildung 1: Eine Lösung der Beispiele Bsp 11, Bsp 12, Bsp 13

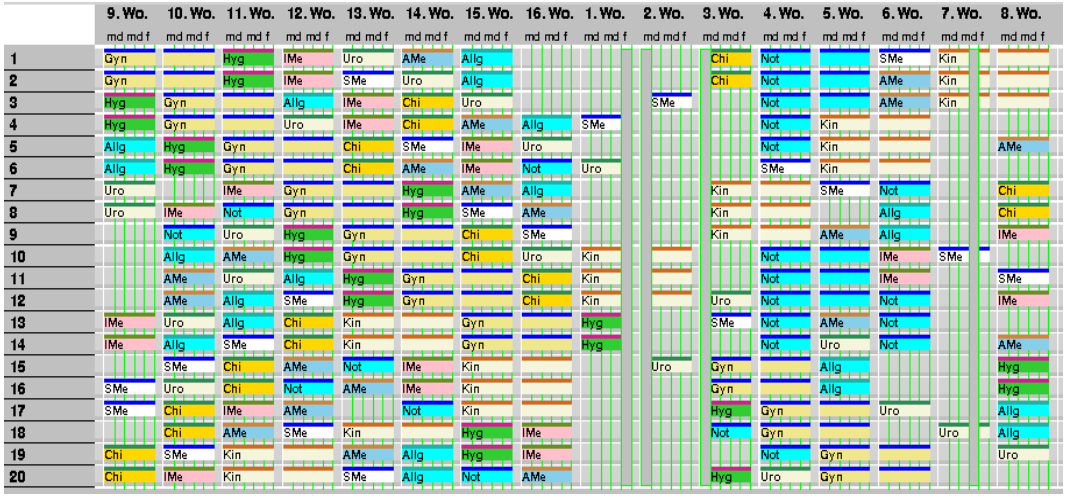

Abbildung 2: Eine Lösung der Beispiele Bsp 21, Bsp 22, Bsp 23

Wenn ein Praktikum für alle Gruppen zu absolvieren ist, könnte bei der Modellierung dieses Praktikum auch in 2 Praktika aufgeteilt und diesen beiden Praktika disjunkte Mengen von Gruppen zugeordnet werden. Die Beispiele Bsp 11 und Bsp 12 unterscheiden sich durch eine solche unterschiedliche Modellierung. Außerdem sind die zugeordneten Prioritäten der Praktika verschieden. Nur eine unterschiedliche Priorioriität eines Praktikums ist die Differenz zwischen den Beispielen Bsp 12 und Bsp 13 . Die anderen 3 Beispiele gehören zum

2. Problem, bei dem 20 Gruppen die Praktika absolvieren müssen. Bei diesem Problem wird für zwei Praktika gefordert, daß in jeder Woche jeweils wenigstens eine Praktikumseinheit durchgeführt wird. Diese Bedingung kann der 7. Bedingung zugeordnet und entsprechend modelliert werden. Eine andere Möglichkeit ist, ein solches Praktikum in zwei Praktika zu teilen, wobei einem Praktikum 16 Gruppen und dem anderen die restlichen 4 Gruppen zugeordnet werden. Für die beiden neu entstandenen Praktika wird dann jeweils gefordert, daß keine Praktikumseinheiten parallel durchgeführt werden dürfen. Bei Bsp 21 wurden diese beiden Praktika in dieser Weise geteilt und bei Bsp 22 wurde ein Praktikum so geteilt. Diese Art der Teilung wurde bei Bsp 23 nicht verwendet. Sonst unterscheiden sich diese 3 Beispiele nicht, auch die Prioritäten der Praktika sind identisch.

Methode		Bsp 11		Bsp 12		Bsp 13		Bsp 21		Bsp 22		Bsp 23	
		#	sec	#	sec	#	sec	#	sec	#	sec	#	sec
S 1	a	–	14,5	–	15,7	–	23,9	–	11,5	692	14,5	667	12,5
	b	11	1,1	–	30,4	145	3,5	134	3,9	692	14,7	667	12,4
	c	11	1,1	–	30,9	83	3,0	24	2,1	4	1,8	184	5,9
	d	0	0,8	–	37,1	31	1,8	0	1,6	1	1,7	1	1,6
S 2	a	–	28,4	–	12,0	–	10,5	–	11,4	–	10,6	69	2,6
	b	–	28,3	–	14,5	–	12,2	33	2,5	8	1,6	69	2,6
	c	–	29,9	–	15,6	–	13,2	33	2,6	8	1,7	69	2,8
	d	0	0,8	–	34,0	14	0,9	0	1,6	27	2,9	0	1,7
S 3	a	2	0,6	186	3,3	–	27,8	–	11,5	0	1,4	–	26,2
	b	1	0,6	11	0,8	53	1,8	0	1,3	0	1,4	–	26,4
	c	1	0,6	11	0,8	44	2,0	0	1,4	0	1,5	–	54,1
	d	1	0,7	10	0,9	23	1,4	0	1,4	0	1,5	–	52,2
S 4	a	6	0,7	185	3,1	–	22,6	–	12,6	–	29,1	–	30,4
	b	5	0,7	11	0,9	136	3,0	0	1,4	–	29,1	–	30,3
	c	5	0,7	11	0,9	136	3,2	0	1,5	–	32,3	–	45,8
	d	3	0,7	10	1,0	28	1,3	0	1,5	34	2,3	–	45,8
S 5	a	–	15,2	–	15,1	–	20,2	–	12,7	–	16,2	–	13,5
	b	11	1,1	–	23,9	21	1,2	123	4,4	–	16,3	–	13,7
	c	11	1,1	–	24,4	20	1,3	44	1,9	–	30,3	–	46,6
	d	0	0,8	–	37,9	15	1,2	4	1,8	3	1,8	1	1,7
S 6	a	–	14,0	–	12,7	–	20,0	–	12,8	–	11,4	–	12,5
	b	18	1,1	–	16,0	21	1,0	132	4,0	–	11,4	–	12,2
	c	18	1,2	–	16,4	20	1,1	62	1,8	–	23,3	–	26,6
	d	0	0,8	–	35,3	14	1,0	0	1,7	1	1,8	0	1,8
S 7	a	–	24,4	–	13,1	–	11,4	–	11,8	–	12,1	69	2,9
	b	–	25,8	–	16,2	–	13,4	–	9,4	8	1,7	69	2,9
	c	–	27,5	–	17,1	–	14,4	33	2,9	8	1,9	69	3,0
	d	0	0,9	–	38,0	14	1,1	0	1,7	27	3,2	0	1,8

Tabelle 1: Ergebnisse des ersten Vergleiches

Für den ersten Vergleichstest setzen wir voraus, daß als redundante Constraints die `diffn`-Constraints und die bedingten Constraints (wie im 4. Abschnitt beschrieben) hinzugefügt werden. Bezüglich der Hinzunahme von zusätzlichen `among`-Constraints betrachten wir verschiedene Varianten:

a: *keine solchen redundanten Constraints werden erzeugt*;

b: *nur für kritische Praktika* (Praktika, bei denen in fast allen Wochen Einheiten stattfinden müssen);

c: *für alle Praktika bezüglich der Startzeitvariablen*;

d: *für alle Praktika bezüglich der Startnummervariablen.*

Für alle Strategien, die im 5. Abschnitt beschrieben wurden, alle 6 Beispiele und die 4 Varianten der Hinzunahme zusätzlicher `among`-Constraints werden in der Tabelle 1 die Ergebnisse der Lösungssuche dargestellt. Die Suche wurde abgebrochen, wenn eine Lösung nicht innerhalb von 1000 Backtracking-Schritten für Entscheidungen der Lösungssuche gefunden wurde. In den Spalten mit der Bezeichnung # ist die Anzahl der benötigten Backtracking-Schritte angegeben, falls die Suche erfolgreich war. Das Zeichen – wurde eingetragen, wenn die Suche abgebrochen wurde. Die benötigte Zeit für die Lösungssuche (auf einer SUN-*Ultra-Sparc 1*) sind in Sekunden angegeben.

Methode		Bsp 11		Bsp 12		Bsp 13		Bsp 21		Bsp 22		Bsp 23	
		#	sec	#	sec	#	sec	#	sec	#	sec	#	sec
S 1	d	0	0,8	–	37,1	31	1,8	0	1,6	1	1,7	1	1,6
	e	1	0,6	–	28,9	39	1,6	0	1,3	2	1,4	2	1,4
	f	482	9,6	–	23,4	814	10,3	0	0,9	1	1,0	1	1,0
	g	212	2,6	–	13,8	328	2,6	0	0,6	2	0,6	2	0,6
S 2	d	0	0,8	–	34,0	14	0,9	0	1,6	27	2,9	0	1,7
	e	0	0,7	–	28,9	14	0,8	0	1,4	27	2,6	0	1,4
	f	259	4,4	–	9,6	242	3,0	0	0,8	27	1,5	0	0,8
	g	259	3,3	–	8,0	246	2,2	0	0,7	27	1,1	0	0,6
S 3	d	1	0,7	10	0,9	23	1,4	0	1,4	0	1,5	–	52,2
	e	25	0,8	34	1,1	36	1,4	–	20,7	–	17,0	–	20,7
	f	1	0,3	–	13,6	84	1,3	0	0,7	0	0,7	–	25,8
	g	25	0,4	–	7,4	84	0,9	–	9,6	–	8,7	–	10,1
S 4	d	3	0,7	10	1,0	28	1,3	0	1,5	34	2,3	–	45,8
	e	–	10,6	34	1,2	28	1,1	–	12,4	–	14,5	–	27,4
	f	3	0,4	–	6,0	92	1,3	0	0,8	34	1,1	–	21,8
	g	–	7,3	–	4,5	96	0,8	–	8,1	–	8,9	–	11,6
S 5	d	0	0,8	–	37,9	15	1,2	4	1,8	3	1,8	1	1,7
	e	1	0,7	–	29,3	16	1,1	1	1,4	2	1,5	1	1,4
	f	452	9,7	–	24,2	559	8,3	4	1,0	3	1,0	1	1,0
	g	212	2,7	–	14,4	304	2,3	1	0,6	2	0,6	1	0,7
S 6	d	0	0,8	–	35,3	14	1,0	0	1,7	1	1,8	0	1,8
	e	0	0,7	–	28,9	14	0,9	0	1,4	1	1,4	0	1,4
	f	–	11,7	–	15,9	272	3,5	0	0,9	1	1,1	0	1,0
	g	–	7,0	–	9,6	274	1,9	0	0,6	1	0,7	0	0,6
S 7	d	0	0,9	–	38,0	14	1,1	0	1,7	27	3,2	0	1,8
	e	0	0,7	–	29,2	14	0,8	0	1,4	27	2,5	0	1,4
	f	259	5,4	–	12,3	236	3,6	1	1,0	27	1,8	0	0,9
	g	259	3,3	–	8,1	246	2,1	0	0,6	27	1,1	0	0,6

Tabelle 2: Ergebnisse des zweiten Vergleiches

Aus der Tabelle 1 ist sofort ersichtlich, daß die Variante "d " der Hinzunahme zusätzlicher `among`-Constraints die beste ist. Bei dem zweiten Vergleich wird deshalb diese Variante vorausgesetzt und es werden Varianten der Hinzunahme der anderen redundanten Constraints betrachtet, wobei die Variante "d" in beiden Tabellen identisch ist:

d: *mit* diffn-Constraints und *mit* den bedingten Constraints;

e: *mit* diffn-Constraints und *ohne* die bedingten Constraints;

f: *ohne* diffn-Constraints und *mit* den bedingten Constraints;

g: *ohne* diffn-Constraints und *ohne* die bedingten Constraints.

Aus diesen beiden Tabellen können unter anderem die folgenden Schlußfolgerungen gezogen werden:

- Unter den Suchstrategien S 1 – S 7 gibt es keine, welche für alle Beispiele durchgängig die beste wäre.
- Einige Suchstrategien zeigen unter einander ein relativ ähnliches Verhalten. Mit Abstrichen können diese Suchstratgien bezüglich ihrer Ergebnisse in zwei Gruppen eingeteilt werden: { S 1, S 2, S 5, S 6, S 7 } und { S 3, S 4 }.
- Obwohl sich durch Hinzunahme redundanter Constraints die Zeit der Lösungssuche verlängern kann, ist dies im allgemeinen vorteilhaft.
- Zusätzliche `among`-Constraints sollten für alle Praktika bezüglich der Startnummervariablen erzeugt werden.
- Die Hinzunahme redundanter Constraints verringert nicht in jedem Fall den Suchraum.
- Obwohl relativ viele redundante bedingte Constraints zu erzeugen sind (1708 für die ersten 3 Beispiele und 3268 für die anderen Beispiele), ist die für die Lösungssuche zusätzlich benötigte Zeit verhältnismäßig gering.
- Bei Vergleich der Suchergebnisse der Beispiele Bsp 12 und Bsp 13 wird deutlich, welche Auswirkungen eine kleine Änderung der Reihenfolge der Domänenvariablen bewirken kann.

7 Zusammenfassung und Ausblick

Die Ergebnisse der Lösungssuche zeigen, daß unsere grundsätzliche Vorgehensweise bei der Lösungssuche richtig ist (siehe auch [6]): die Anzahl der erlaubten Backtracking-Schritte für Entscheidungen der Lösungssuche wird stark eingeschränkt und es werden verschiedene Heuristiken ausprobiert. Somit wird ein Backtracking über verschiedene Heuristiken durchgeführt. In unserer Standardeinstellung ist die Anzahl der erlaubten Backtracking-Schritten für Entscheidungen der Lösungssuche auf 100 begrenzt. Aus den dargestellten Ergebnissen ergeben sich für die Heuristiken, die ausprobiert werden sollten, folgende Schlußforgerungen:

- Alle redundanten Constraints sollten hinzugefügt werden (`among`-Constraints bzgl. Startnummervariablen); evtl. kann im ersten Versuch auf das Erzeugen der zusätzlichen `diffn`-Constraints und bedingten Constraints verzichtet werden.

- Aus den beiden Gruppen "ähnlicher" Suchstrategien sollte jeweils eine probiert werden (beispielweise S 3 und S 6).
- Verschiedene Heuristiken der Variablenauswahl sind zu verwenden.

In weiteren Untersuchungen ist festzustellen, ob sich unsere Schlußfolgerungen auch auf anderen Probleme übertragen lassen. Außerdem sind weitere Untersuchungen zu Heuristiken der günstigsten Variablenauswahl durchzuführen. Insbesondere ist das Erkennen von Engpässen wichtig. Verschiedene Methoden und Heuristiken sind auch für die anderen Teilprobleme der Stundenplanung zu untersuchen. Neben der vollständigen automatischen Planerzeugung ist für die praktische erfolgreiche Anwendung des Stundenplansystems die interaktive Beeinflussung der Lösungssuche von entscheidender Bedeutung. Beispielsweise ist es möglich einzelne Lehrveranstaltungen einzuplanen oder umzuplanen. Bei der interaktiven Beeinflussung der Lösungssuche erfolgt kein "Backtracking" über Entscheidungen des menschlichen Planers.

Literatur

[1] P. Boizumault, Y. Delon, and L. Peridy. Constraint logic programming for examination timetabling. *J. Logic Programming*, 26(2):217–233, 1996.

[2] E. Burke and M. Carter, editors. *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1998.

[3] E. Burke and P. Ross, editors. *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1996.

[4] T. B. Cooper and J. H. Kingston. The complexity of timetable construction problems. In *[3]*, pages 183–295, 1996.

[5] H.-J. Goltz. Reducing domains for search in CLP(FD) and its application to job-shop scheduling. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming – CP'95*, volume 976 of *Lecture Notes in Computer Science*, pages 549–562, Berlin, Heidelberg, 1995. Springer-Verlag.

[6] H.-J. Goltz and D. Matzke. University timetabling using constraint logic programming. In G. Gupta, editor, *Practical Aspects of Declarative Languages*, volume 1551 of *Lecture Notes in Computer Science*, pages 320–334, Berlin, Heidelberg, New York, 1999. Springer-Verlag.

[7] C. Guéret, N. Jussien, P. Boizumault, and C. Prins. Building university timetables using constraint logic programming. In *[3]*, pages 130–145, 1996.

[8] M. Henz and J. Würtz. Using Oz for college timetabling. In *[3]*, pages 162–177, 1996.

[9] G. Lajos. Complete university modular timetabling using constraint logic programming. In *[3]*, pages 146–161, 1996.

[10] A. Schaerf. A survey of automated timetabling. Technical Report CS-R9567, Centrum voor Wiskunde en Informatica, 1995.

[11] G.M. White and J. Zhang. Generating complete university timetables by combining tabu search with constraint logic. In *[2]*, pages 187–198, 1998.