Constraintbasierte Stundenplanung für Universitäten

Slim Abdennadher und Michael Marte
Institut für Informatik, Ludwig-Maximilians-Universität
Oettingenstraße 67, 80538 München
{Slim.Abdennadher,Michael.Marte}@informatik.uni-muenchen.de
`http://www.pms.informatik.uni-muenchen.de/ifiplan/`

Zusammenfassung

Stundenplanung für das Institut für Informatik der Universität München erfordert die Verarbeitung harter und weicher Constraints. Harte Constraints sind unbedingt zu erfüllende Bedingungen, weiche Constraints dürfen zwar verletzt werden, sollten aber so gut wie möglich erfüllt werden. Dieses Papier zeigt die Modellierung unseres Stundenplanungsproblems als Partial Constraint Satisfaction Problem und stellt einen kurzen und deklarativen Löser für endliche Bereiche vor, der auch weiche Constraints propagiert und damit für diese die Möglichkeit eröffnet, eine aktive Rolle im Lösungsprozess einzunehmen.

1 Einleitung

Constraints sind vordefinierte Relationen, die es erlauben, in deklarativer Weise Probleme durch Angabe von Nebenbedingungen zu formulieren, die im Vergleich zu herkömmlichen Programmiersprachen die Suche nach geeigneten Lösungen beschleunigen [vH89].

Das wichtigste Gebiet der kommerziellen Anwendung von Constraints ist das Lösen von kombinatorischen Problemen. Sie sind Bestandteil vieler industrieller Anwendungen, wie zum Beispiel Scheduling, Konfiguration oder Stundenplanung. Aus einer großen Menge von möglichen Wertekombinationen gilt es diejenigen zu bestimmen, die gewisse Constraints erfüllen (oder besonders gut erfüllen). Klassische Ansätze zur Lösung solcher Probleme leiden oft an mangelnder Flexibilität und Erweiterbarkeit. Der Erfolg der Constraintprogrammierung bei der Lösung solcher Optimierungsprobleme beruht auf der deklarativen Modellierung, der Constraintpropagierung und der guten Kombinierbarkeit mit Such- und Optimierungsverfahren.

Meistens werden Stundenpläne von Hand erstellt, wobei verschiedene „harte“ Bedingungen unbedingt erfüllt werden müssen, und „weiche“ Bedingungen nach Möglichkeit. Die meisten existierenden Stundenplaner sind mit Systemen implementiert, die von Hause aus keine weichen Constraints behandeln können. Einige Systeme behandeln dann nur harte Constraints [AB94]. Andere weisen jedem weichen Constraint des Problems fiktive, mit der Verletzung des Constraints verbundene Kosten zu, suchen eine Lösung, die alle harten Constraints erfüllt, und optimieren diese mit Branch-And-Bound [FHS95, HW95]. Ein weiterer Ansatz ist die Anwendung von Constraintverfahren, die sowohl harte als auch weiche Constraints behandeln [Mey97]. Harte Constraints werden unmittelbar zur Reduktion der betroffenen Wertebereiche verwendet. Weiche Constraints hingegen werden zur Ermittlung einer Bewertung der Elemente der Wertebereiche eingesetzt. Diese Bewertungen steuern die Suche. Dadurch erzielt man, daß die erste Lösung brauchbar ist, d.h. nicht nur alle harten Constraints erfüllt, sondern auch viele weiche Constraints. Dieser Ansatz wurde für die Dienstplanung in Krankenhäusern mit C++ implementiert [Mey97].

Unser Ziel war der Einsatz dieser Technik für die Stundenplanung in unserem Institut. Dafür haben wir einen Constraintlöser mit der deklarativen Constraintsprache *Constraint Handling Rules* (CHR) [Frü95] implementiert. Mit CHR konnte die Behandlung harter und weicher Constraints in einem Löser kurz und deklarativ spezifiziert werden. Basierend auf diesem Löser ahmt unser System, der IfI-Planer, die manuelle Planung nach. Er erstellt für unser Institut, jeweils ausgehend vom Stundenplan des Vorjahres, einen neuen Plan für das kommende Semester. Dabei stellt die Kontinuität in der Stundenplanung eines der Optimalitätskriterien dar.

Dieses Papier ist wie folgt gegliedert: Abschnitt 2 beschreibt unser Planungsproblem und Abschnitt 3 dessen Modellierung als Partial Constraint Satisfaction Problem. In Abschnitt 4 wird dann beschrieben, wie dieses Planungsproblem mit CHR gelöst wurde. Abschnitt 5 ist Zusammenfassung und Ausblick.

2 Das Problem

Unser Ziel war die Generierung eines Stundenplans für eine gegebene Menge von Angeboten unter Berücksichtigung der Dozentenwünsche und des Vorjahresplans. Unser System sollte Teil des Planungsprozesses werden, der wie folgt abläuft.

Nach Sammlung von Dozentenwünschen und Informationen über neue Angebote wird ein erster Vorschlag basierend auf dem Vorjahresplan entwickelt. Durch nicht mehr stattfindende Angebote freigewordene Plätze werden dabei zuerst wiederverwendet, und zwar bevorzugt für neue Angebote desselben Lehrstuhls. Dozentenwünsche sind dabei aber wichtiger als der Vorjahresplan. Nach Verteilung des Vorschlags an alle Beteiligten werden Bewertungen und neue Wünsche gesammelt. Mit dem aktuellen Vorschlag als Ausgangspunkt wird dann ein nächster Vorschlag entwickelt, der diese Reaktionen berücksichtigt, wobei aber wieder sowenig wie möglich geändert wird, usw. Dieses konservative Vorgehen reduziert nicht nur den Aufwand für die Erstellung eines neuen Stundenplans, sondern sichert auch dessen Aktzeptanz durch Beibehaltung der gewohnten wöchentlichen Abläufe.

Die Anforderungen an den Stundenplan sind wie folgt.

- Veranstaltungen eines Dozenten dürfen sich nicht überschneiden. Zwischen Veranstaltungen eines Dozenten sollte mindestens eine Stunde Pause sein.
- Veranstaltungen eines Lehrstuhls dürfen sich nicht mit anderen Veranstaltungen desselben Lehrstuhls überschneiden.
- Einige Dozenten bevorzugen bestimmte Zeiten oder Tage für ihre Veranstaltungen. Montag nachmittag ist reserviert für die Vorstandssitzung der Professoren.
- Ein Angebot besteht typischerweise aus zwei Vorlesungen und einer Übung pro Woche. Zwischen den beiden Vorlesungen sollte mindestens ein Tag liegen. Eine Übung sollte nicht mit einer der Vorlesungen am gleichen Tag liegen. Veranstaltungen sollten zwischen neun und achtzehn Uhr stattfinden, Vorlesungen aber nicht am Freitag nachmittag und Übungen nicht am späten Freitag nachmittag.
- Veranstaltungen eines Semesters des Grundstudiums dürfen sich nicht überschneiden. Veranstaltungen des Hauptstudiums sollten sich nicht überschneiden.

Erste Untersuchungen ergaben, daß bestehende Stundenpläne die gegebenen Forderungen nicht erfüllen: Veranstaltungen eines Lehrstuhls oder Veranstaltungen des Hauptstudiums überschneiden sich. Vorlesung und Übung eines Angebots finden am gleichen Tag statt. Außerdem ergibt schon die Betrachtung der Anzahl der angebotenen Veranstaltungen des Hauptstudiums, daß es zuwenig Platz gibt, um alle diese Veranstaltungen überschneidungsfrei zu plazieren. Im Grundstudium tritt dieses Problem nicht auf, da dort verschiedene Semester klar unterschieden werden, was das Überschneiden der Veranstaltungen verschiedener Semester erlaubt. Wir waren also mit zwei Problemen konfrontiert:

- Inkrementelle Planung auf Basis des Vorjahresplans erforderte die Verarbeitung von Stundenplänen, die die angegebenen Bedingungen nicht erfüllen.
- Keine klare Struktur im Hauptstudium kostet Freiheiten bei der Planung und hat inkonsistente Stundenplanspezifikationen zur Folge.

Ein Auflösen des zweiten Problems durch das gezielte Zulassen von Überschneidungen gestaltete sich mühevoll und zeitaufwendig. Außerdem reagierten so entstandene konsistente Spezifikationen sehr empfindlich schon auf kleine Änderungen, die oftmals wieder Inkonsistenz zur Folge hatten. Die Klassifikation von Angeboten des Hauptstudiums nach Inhalt und typischer Größe des Publikums eröffnete die Möglichkeit, Überschneidungen zwischen Veranstaltungen verschiedener Klassen zuzulassen. Diese Maßnahme konnte Freiheiten gewinnen, aber die Konflikte nicht völlig auflösen. Die Notwendigkeit einer Art gewichteter Constraints wurde deutlich.

3 Modellierung als PCSP

Ein *Partial Constraint Satisfaction Problem* (PCSP) [FW92] besteht aus einer Menge von Variablen und einer Menge von Relationen zwischen den Variablen (Constraints), wobei jedes Constraint mit einem Gewicht versehen ist. Gewichte erlauben die Unterscheidung harter und weicher Constraints. Harte Constraints müssen erfüllt werden. Die Verletzung weicher Constraints ist zwar zulässig, sollte aber doch weitgehend vermieden werden. Harte Constraints sind durch ein unendliches Gewicht gekennzeichnet. Die endlichen Gewichte weicher Constraints ermöglichen die Spezifikation von Präferenzen innerhalb der Menge der weichen Constraints. Jeder Variable ist außerdem ein endlicher Wertebereich zugeordnet. Die Lösung eines PCSP bildet jede Variable unter Erfüllung aller harten Constraints auf einen Wert aus ihrem Wertebereich ab, sodaß die Summe der Gewichte der verletzten weichen Constraints minimal ist.

Zur Modellierung unseres Stundenplanungsproblems als PCSP führen wir für jede zu plazierende Veranstaltung eine Variable ein. Der anfängliche Wertebereich jeder Variablen ist die ganze Woche, wobei für jeden Tag 24 Zeitpunkte vorgesehen sind. Die anwendungsspezifischen Anforderungen und die Dozentenwünsche können durch folgende Constraints ausgedrückt werden:

- Ein *Überschneidungsconstraint* verlangt, daß zwei Veranstaltungen sich nicht überschneiden dürfen.

- Ein *Zeitconstraint* bringt entweder einen Dozentenwunsch zum Ausdruck oder fordert, daß eine Veranstaltung zu bestimmten Zeitpunkten stattfinden muß bzw. zu diesen Zeitpunkten nicht stattfinden darf.

- Ein *Verteilungsconstraint* stellt sicher, daß mindestens ein Tag (eine Stunde) zwischen zwei Veranstaltungen liegt oder daß zwei Veranstaltungen an verschiedenen Tagen stattfinden.

- Ein *Anschlußconstraint* stellt sicher, daß eine Veranstaltung direkt nach einer anderen Veranstaltung stattfindet.

Bei den Gewichten für die weichen Constraints unterscheiden wir die drei Abstufungen „schwach bevorzugt“, „bevorzugt“ und „stark bevorzugt“, die in die ganzzahligen Gewichte 1, 3 bzw. 9 übersetzt werden.

4 Lösung mit CHR

Constraint Handling Rules (CHR) [Frü95] ist eine deklarative, auf einem Regelformalismus basierende Sprache zur Implementierung von Constraintlösern, mit der eine beliebige Basissprache, z.B. Prolog oder Lisp, um benutzerdefinierte Constraints erweitert werden kann. Man unterscheidet zwei Arten von Regeln: Simplifikationsregeln beschreiben, wie Constraints sich zu neuen Constraints vereinfachen (z.B. `X > Y, Y > X <=> false`) und Propagationsregeln beschreiben, wie Constraints neue Constraints propagieren (z.B. `X > Y, Y > Z ==> X > Z`). Aus Platzgründen kann hier keine formale Definition der Syntax und Semantik von CHR gegeben werden. Wir verweisen auf [Frü95] und [Abd97].

Um den Suchaufwand zur Lösung kombinatorischer Probleme zu reduzieren, werden in der Künstlichen Intelligenz *Konsistenzverfahren* [Mac92, Kum92] eingesetzt. Die Idee dieser Verfahren besteht darin, Werte aus den Wertebereichen der Problemvariablen zu entfernen, die nicht Bestandteil einer Lösung sein können. Die beiden Constraints $x \in \{2,3,\ldots,6\}$ und $x \in \{4,5,\ldots,9\}$ können z.B. durch das neue Constraint $x \in \{4,5,6\}$ ersetzt werden.

Dieses Verfahren kann leicht mit CHR implementiert werden [FA97], aber dieses Schema ist nicht ausreichend für unsere Bedürfnisse: Da weiche Constraints verletzt werden dürfen, ist die Entfernung von Werten aus Wertebereichen, deren Auswahl die Verletzung weicher Constraints bedeuten würde, nicht möglich. Außerdem benötigen wir für die Wertauswahl während der Suche eine Basis für die Entscheidung, ob ein Wert hinsichtlich der Erfüllung weicher Constraints eine gute Wahl darstellt oder nicht. Wir versehen deswegen jeden Wert mit einer Einschätzung und repräsentieren einen Wertebereich als eine Liste von Paaren (Wert, Einschätzung). Sei z.B. `[(3, 0), (4, 1), (5, -1)]` der Wertebereich von `X`. Dann darf `X` einen der Werte `3`, `4` und `5` annehmen, wobei `4` mit Einschätzung `1` gegenüber `3` und `5` mit den Einschätzungen `0` bzw. `-1` bevorzugt wird.

Die Propagation eines weichen Constraints bedeutet dann die Veränderung der Einschätzungen für die mit dem Constraint unvereinbaren Werte. Betrachten wir z.B. ein Zeitconstraint, das mit Gewicht `2` den Wert `3` für `X` ablehnt. Dann müssen wir die Einschätzung für `3` um `2` nach unten korrigieren, womit wir den neuen Wertebereich `[(3, -2), (4, 1), (5, -1)]` für `X` erhalten. Die Propagation eines harten Constraints bedeutet aber weiterhin die Entfernung von Werten aus den Wertebereichen der betroffenen Variablen.

Der Constraintlöser basiert auf drei Arten von Constraints:

- `domain(C, S, D)` bedeutet, daß die Veranstaltung `C` nur zu einem Zeitpunkt `S` stattfinden darf, der im Wertebereich `D` enthalten ist, wobei `D` eine Liste von Paaren (Zeitpunkt, Einschätzung) ist.
- `in(C, L, W)`: Wenn `W = infinite`, d.h. wenn das Constraint hart ist, dann darf die Veranstaltung `C` nicht zu einem Zeitpunkt stattfinden, der nicht in `L` enthalten ist. Wenn `W` eine Zahl ist, d.h. wenn das Constraint weich ist, dann sollen die Einschätzungen für die Zeitpunkte, die in `L` vorkommen, um `W` erhöht werden.
- `notin(C, L, W)`: Wenn das Constraint hart ist, dann darf die Veranstaltung `C` nicht zu einem Zeitpunkt stattfinden, der in `L` enthalten ist. Wenn das Constraint weich ist, dann sollen die Einschätzungen für die Zeitpunkte in `L` um `W` verkleinert werden.

Wir stellen nun den Kern unseres Constraintlösers vor. Dieser besteht aus zwei Regeln zur Verarbeitung von `in`- und `notin`-Constraints. Weitere acht Regeln dienen der Übersetzung der anwendungsspezifischen Constraints.

Die Verarbeitung eines `in`-Constraints bedeutet entweder das Entfernen von Werten aus dem Wertebereich oder die Erhöhung der Einschätzung für die angegebenen Zeitpunkte. Dies kann in CHR mit der folgenden Simplifikationsregel ausgedrückt werden.

```
domain(C, S, D), in(C, L, W) <=>
  (
     W = infinite
  -> domain_intersection(D, L, D1),
  ;  increase_assessment(W, L, D, D1)
  ),
  domain(C, S, D1).
```

Wenn ein neues `in`-Constraint vorliegt, dann sucht die Simplifikationsregel nach dem zugehörigen `domain`-Constraint und ersetzt beide durch ein neues `domain`-Constraint mit einem veränderten Wertebereich. Für ein hartes `in`-Constraint, d.h. für `W = infinite`, ist das der Schnitt des Wertebereichs `D` mit der Liste von Zeitpunkten `L`. Für ein weiches `in`-Constraint bleiben alle Werte erhalten, lediglich die Einschätzungen für die Werte, die in `L` vorkommen, werden um `W` erhöht. `notin`-Constraints können durch eine weitere Simplifikationsregel in analoger Art und Weise behandelt werden.

Wenn ein Wertebereich leer wird, dann ist die Spezifikation inkonsistent. Eine Veranstaltung kann nicht ohne Verletzung harter Constraints plaziert werden. Diesen Fall erledigt folgende Simplifikationsregel:

```
domain(_, _, []) <=> fail.
```

Bis jetzt haben wir uns nur mit den grundlegenden, unären Constraints unseres PCSP-Lösers für endliche Bereiche beschäftigt. Im folgenden stellen wir exemplarisch die Realisierung von anwendungsspezifischen n-ären Constraints mittels `in`- und `notin`-Constraints dar.

`no_clash(W, Cs)` bedeutet in Abhängigkeit vom Gewicht `W`, daß sich die Veranstaltungen der Liste `Cs` nicht überschneiden dürfen oder sollten. Es wird in `notin`-Constraints übersetzt, wobei die Übersetzung datengetrieben ist: Wenn eine der Veranstaltungen in `Cs` plaziert wurde, dann wird der gewählte Platz durch die folgende Regel für die anderen Veranstaltungen verboten oder abgelehnt.

```
no_clash(W, Cs) <=>
  Cs \= [_],
  select_ground_var(Cs, X, CsRest)
  |
  post_notin_constraints(W, X, CsRest),
  no_clash(W, CsRest).
```

Der Wächter stellt zunächst sicher, daß die Liste `Cs` der Veranstaltungen mindestens zwei Elemente enthält. Dann wählt er eine instanziierte Variable `X` aus `Cs` aus und merkt sich die anderen Veranstaltungen in `CsRest`. Gibt es keine instanziierte Variable in `Cs`, dann scheitert `select_ground_var`. Ist der Wächter erfüllt, dann wird `no_clash(W, Cs)` ersetzt durch

- von `post_notin_constraints` erzeugte `notin`-Constraints, eines für jede Veranstaltung in `CsRest`, die den Platz `X` verbieten oder ablehnen und
- einem `no_clash`-Constraint, das fordert, daß sich die Veranstaltungen in `CsRest` nicht überschneiden dürfen oder sollten.

`post_notin_constraints` scheitert, falls `CsRest` den Wert von `X` enthält.

Eine einelementige Veranstaltungsliste bedeutet, daß es nichts mehr zu tun gibt. Diesen Fall erledigt die folgende Regel.

```
no_clash(_, [_]) <=> true.
```

Der hier vorgestellte PCSP-Löser für endliche Bereiche implementiert Kantenkonsistenz für harte binäre Überschneidungs-, Verteilungs- und Anschlußconstraints. Für n-äre Constraints wird inkrementell ein Constraintnetzwerk aus binären Constraints aufgespannt. Für harte Überschneidungsconstraints ist dieser Ansatz äquivalent zu der in CHIP [vH89] für das `all_distinct`-Constraint implementierten Methode. Wir wissen um die schlechte Propagationsleistung eines Netzwerks binärer Constraints für ein n-äres Constraint [Reg94], aber dieser Ansatz ist einfach zu implementieren und erwies sich als ausreichend zur Lösung unseres Problems.

Die Propagation weicher Constraints berechnet Einschätzungen für die Werte, die noch nicht durch Propagation harter Constraints entfernt wurden. Diese Einschätzungen können zur Information der Suchprozedur über vielversprechende Werte verwendet werden, wodurch weiche Constraints zu einem aktiven Bestandteil des Lösungsprozesses werden. Wenig Propagation für weiche n-äre Überschneidungsconstraints kann zur Folge haben, daß die erste Lösung viele weiche Constraints verletzt. In unserem Fall trat dieses Problem aber nicht auf.

Die Plangenerierung läuft wie folgt ab: Zuerst wird jeder Veranstaltung ein `domain`-Constraint zugeordnet. Die anfängliche Einschätzung ist `0` für jeden Zeitpunkt, d.h. kein Zeitpunkt wird gegenüber einem anderen bevorzugt. Dann werden die Bedingungen, die der Stundenplan erfüllen soll, in `in`- und `notin`-Constraints übersetzt. Die Suchprozedur verwendet zur Variablenauswahl das *first-fail*-Prinzip, d.h. eine der Variablen mit dem kleinsten Wertebereich wird gewählt. Zur Wertauswahl wird die *best-fit*-Strategie benutzt, d.h. einer der Werte mit der besten Einschätzung wird gewählt. Optimistisch betrachtet wird das einer der Zeitpunkte sein, für den die Gewichtssumme der verletzten weichen Constraints minimal ist, aber die Einschätzung kann wegen der Unvollständigkeit des Lösers für n-äre Überschneidungsconstraints zu gut sein. Da weiche Constraints vom Constraintlöser und vom Suchverfahren berücksichtigt werden, war die erste Lösung meistens sehr zufriedenstellend. Der IfI-Planer sucht deswegen nicht nach optimalen Lösungen. Durch Aufsummieren der Einschätzungen der für eine Lösung verwendeten Werte erhält man aber ein Qualitätsmerkmal, daß in Branch-And-Bound zur Verbesserung der ersten Lösung eingesetzt werden kann.

Der IfI-Planer benötigt etwa fünf Minuten zur Erstellung eines neuen Plans, der 89 Veranstaltungen innerhalb von 42 Stunden plaziert. Legt man den Vorjahresplan zugrunde, dann reduziert sich der Aufwand auf etwa zweieinhalb Minuten.

5 Zusammenfassung und Ausblick

Das Papier zeigt, daß Constraint Handling Rules zur Implementierung eines PCSP-Lösers für endliche Bereiche gut geeignet ist. Der Löser ist kurz und deklarativ, propagiert harte und weiche Constraints und ist leistungsfähig genug, um als Kern eines Stundenplaners für unser Institut zu dienen.

Unser Planer läuft auf ECL^{i}PS^e [ACD$^+$94] ergänzt um die CHR-Bibliothek [BFL$^+$94]. Dozenten können neue Wünsche und Angebote selbst über das WWW in die Spezifikation eintragen. Die WWW-Schnittstelle basiert auf HTML, die Seiten werden ebenfalls von einem Prolog-Programm generiert. Die Entwicklung der WWW-Schnittstelle dauerte zwei Wochen, die Entwicklung des Planers nahm weitere drei Wochen in Anspruch.

Unser Prototyp wird mittlerweile im Institut für Informatik an der Universität München eingesetzt. Der Stundenplan für das Sommersemester 98 ist mit dem IfI-Planer erstellt worden. Um die Qualität der erzeugten Stundenpläne weiter zu verbessern, möchten wir das Constraintlösen mit Optimierungsverfahren (z.B. Branch-And-Bound) kombinieren.

Der in dieser Anwendung vorgestellte Ansatz wird jetzt zur Erstellung eines Stundenplans für ein Gymnasium untersucht. Diese Anwendung unterscheidet sich vom IfI-Planer dadurch, daß auf den Lehrplan des vorherigen Jahres nicht zurückgegriffen werden kann. Außerdem ist das Problem komplexer wegen der größeren Anzahl zu plazierender Stunden und den vielfältigen Qualitätskriterien, die der Plan erfüllen sollte.

Literatur

[AB94] F. Azevedo und P. Barahona. Timetabling in Constraint Logic Programming. In *Proceedings of 2nd World Congress on Expert Systems*, 1994.

[Abd97] S. Abdennadher. Operational Semantics and Confluence of Constraint Propagation Rules. In *Third International Conference on Principles and Practice of Constraint Programming, CP'97*, LNCS 1330. Springer, 1997.

[ACD+94] A. Aggoun, D. Chan, P. Dufrense, E. Falvey, H. Grant, A. Herold, G. MacCartney, M. Maier, D. Miller, B. Perez, E. van Rossum, J. Schimpf, P. Tsahageas und D. de Villeneuve. *ECL^{i}PS^e 3.4 User Manual.* ECRC Munich, 1994.

[BFL+94] P. Brisset, T. Frühwirth, P. Lim, M. Meier, T. Le Provost, J. Schimpf und M. Wallace. *ECL^{i}PS^e 3.4 Extensions User Manual.* ECRC Munich, 1994.

[FA97] T. Frühwirth und S. Abdennadher. *Constraint-Programmierung: Grundlagen und Anwendungen.* Springer, 1997.

[FHS95] H. Frangouli, V. Harmandas und P. Stamatopoulos. UTSE: Construction of Optimum Timetables for University Courses — A CLP Based Approach. In *Proceedings of the Third International Conference on the Practical Applications of Prolog (PAP'95).* Alinmead Software Ltd, 1995.

[Frü95] T. Frühwirth. Constraint Handling Rules. In A. Podelski, Hrsg., *Constraint Programming: Basics and Trends*, LNCS 910. Springer, 1995.

[FW92] E. C. Freuder und R. J. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58(1-3):21–70, 1992.

[HW95] M. Henz und J. Wurtz. Using Oz for college time tabling. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, 1995.

[Kum92] V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, 13(1), 1992.

[Mac92] A. Mackworth. Constraint Satisfaction. In Stuart C. Shapiro, Hrsg., *Encyclopedia of Artificial Intelligence*, Band 1. Wiley, 1992.

[Mey97] H. Meyer auf'm Hofe. ConPlan/SIEDAplan: Personnel Assignment as a Problem of Hierarchical Constraint Satisfaction. In *Proceedings of the 3rd International Conference on the Practical Application of Constraint Technology*, Blackpool, 1997. The Practical Application Ltd.

[Reg94] J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proc. 12th Conf. American Assoc. Artificial Intelligence*, 1994.

[vH89] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Massachusetts, 1989.