MAX-PLANCK-INSTITUT FÜR INFORMATIK

Künstliche Intelligenz und Operations Research

Workshop, Berlin, 13. - 14. September 1993

Extended Abstracts

A. Bockmayr, F. J. Radermacher (Hrsg.)

MPI–I–93–234 September 1993

INFORMATIK

Im Stadtwald

66123 Saarbrücken

Germany

Künstliche Intelligenz und Operations Research

Workshop, Berlin, 13. - 14. September 1993
Extended Abstracts

A. Bockmayr, F. J. Radermacher (Hrsg.)

MPI–I–93–234 September 1993

Adressen

Dr. A. Bockmayr, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken
bockmayr@mpi-sb.mpg.de

Prof. Dr. Dr. F. J. Radermacher, Forschungsinstitut für anwendungsorientierte Wissensverarbeitung, Helmholtzstr. 16, Postfach 2060, D-89010 Ulm, radermac@faw.uni-ulm.de

Danksagung

Der Workshop wird unterstützt durch die Gesellschaft für Informatik (GI), das Max-Planck-Institut für Informatik in Saarbrücken und das Forschungsinstitut für anwendungsorientierte Wissensverarbeitung an der Universität Ulm.

Zusammenfassung

Zwischen Künstlicher Intelligenz und Operations Research bestehen zahlreiche Querverbindungen, die bisher nicht die Beachtung gefunden haben, die sie eigentlich verdienen. Beide Disziplinen haben eine Fülle von Fragestellungen und Anwendungsgebieten gemeinsam. Dazu zählen zum Beispiel

- logische Inferenz
- Constraintlösungsverfahren
- heuristische Suche
- Entscheidungsunterstützung

und vieles andere mehr. Ergebnisse und Methoden aus dem einen Bereich können oft sehr fruchtbar in dem anderen eingesetzt werden.

Der Workshop auf der 17. Fachtagung für Künstliche Intelligenz KI'93 in Berlin soll dazu beitragen, von der KI aus eine Brücke zum Operations Research zu schlagen. Sein Ziel ist es, den Austausch zwischen beiden Gebieten zu fördern und Einsatzmöglichkeiten von Begriffen und Methoden der Künstlichen Intelligenz im Operations Research und umgekehrt aufzuzeigen.

Inhaltsverzeichnis

1 Einführung

Zwischen Künstlicher Intelligenz und Operations Research bestehen zahlreiche Querverbindungen, die bisher nicht die Beachtung gefunden haben, die sie eigentlich verdienen. Beide Disziplinen haben eine Fülle von Fragestellungen und Anwendungsgebieten gemeinsam. Dazu zählen zum Beispiel

- logische Inferenz
- Constraintlösungsverfahren
- heuristische Suche
- Entscheidungsunterstützung

und vieles andere mehr. Ergebnisse und Methoden aus dem einen Bereich können oft sehr fruchtbar in dem anderen eingesetzt werden. Leider gibt es nur wenige Wissenschaftler und Praktiker, die sowohl in der Künstlichen Intelligenz als auch im Operations Research zu Hause sind. Es besteht daher ein dringender Bedarf, den Austausch zwischen diesen beiden Gebieten zu intensivieren.

Auf den Tagungen der Deutschen Gesellschaft für Operations Research (DGOR) und der Gesellschaft für Mathematik, Ökonomie und Operations Research (GMÖOR) finden regelmäßig Veranstaltungen zu Themen der Künstlichen Intelligenz statt. Von der Seite des Fachbereichs "Künstliche Intelligenz" der Gesellschaft für Informatik (GI) steht ein vergleichbares Engagement bisher aus. Der Workshop auf der 17. Fachtagung für Künstliche Intelligenz KI'93 in Berlin soll dazu beitragen, diese Lücke zu schließen und von der KI aus eine Brücke zum Operations Research zu schlagen. Sein Ziel ist es, den Austausch zwischen beiden Gebieten zu fördern und Einsatzmöglichkeiten von Begriffen und Methoden der Künstlichen Intelligenz im Operations Research und umgekehrt aufzuzeigen. Der Workshop wendet sich ausdrücklich auch an Anwender aus der industriellen Praxis, für die diese Themen eine besondere Bedeutung besitzen.

Im einzelnen sind folgende Beiträge vorgesehen: *J. N. Hooker* (Carnegie-Mellon-University) gibt in seinem Tutorial "Optimization and Logical Inference" einen breiten Überblick über das Zusammenspiel von Logik und Optimierung. In unmittelbarem Zusammenhang damit stehen die Beiträge von *P. Barth* (MPI Saarbrücken), der ein symbolisches Constraintlösungsverfahren für lineare 0-1 Ungleichungen vorstellt, sowie von *K. Ries* und *R. Hähnle* (Universität Karlsruhe), die gemischt-ganzzahlige Programmierung für das prädikatenlogische Beweisen einsetzen. Die Lösung von Schedulingproblemen mit Hilfe der logischen Programmierung mit Constraints steht im Mittelpunkt der Beiträge von *N. Beldiceanu* und *H. Simonis* (COSYTEC Orsay) und ebenso von *S. Breitinger* und *H. C. R. Lock* (IBM Heidelberg). *A. Herold, M. Wallace* und *V. Küchenhoff* (ECRC München) setzen Constraints in Kombination mit Approximationsalgorithmen zur Optimierung ein. *M. Zahn* (Universität Karlsruhe) untersucht die Lösung von Constraint-Erfüllungs-Problemen mit Hilfe von Join-Trees. *D. Cvetković* (MPI Saarbrücken) wendet genetische Algorithmen auf das Erfüllbarkeitsproblem an, während *J. Bartnick* (Universität Dortmund) die Verwendung neuronaler Netze zur Aggregation von Präferenz-Relationen vorschlägt. Einen Ansatz zur Modellauswahl und -integration in Entscheidungsunterstützungssystemen stellt *M. Lachmann* (Universität Karlsruhe) vor. *J. Dorn, M. Girch* und *W. Slany* (TU Wien) erörtern die Reparatur von Plänen durch fallbasiertes Schließen. *A. Reinefeld* (Uni-GH Paderborn) berichtet über iterative Tiefensuche auf massiv parallelen MIMD-Systemen.

2 Programm

Montag, 13. September 1993

14.00 - 15.30 Uhr TUTORIAL - J. N. Hooker (Carnegie Mellon University): OPTIMIZATION AND LOGICAL INFERENCE I

16.00 - 16.30 Uhr N. Beldiceanu, H. Simonis (COSYTEC Orsay): Maintenance Scheduling with Cumulative Constraints

16.30 - 17.00 Uhr S. Breitinger, H. C. R. Lock (IBM Heidelberg): Heuristics for scheduling with CLP

17.00 - 17.30 Uhr A. Herold, M. Wallace, V. Küchenhoff (ECRC München): Optimisation by Constrained Approximation

17.30 - 18.00 Uhr M. Zahn (Universität Karlsruhe): Lösung von Constraint-Erfüllungs-Problemen mit Hilfe von Join-Trees

18.00 - 18.30 Uhr D. Cvetković (MPI Saarbrücken): Genetic Algorithms and Satisfiability Problems

Dienstag, 14. September 1993

10.30 - 11.00 Uhr J. Bartnick (Universität Dortmund): Aggregation von Präferenz-Relationen mit Neuronalen Netzen

11.00 - 11.30 Uhr M. Lachmann (Universität Karlsruhe): Ein Ansatz zur Modellauswahl und -integration in Entscheidungsunterstützungssystemen

11.30 - 12.00 Uhr J. Dorn, M. Girch und W. Slany (TU Wien): Reparatur von Plänen durch fallbasiertes Schließen

12.00 - 12.30 Uhr A. Reinefeld (Uni-GH Paderborn): Iterative Tiefensuche auf massiv parallelen MIMD-Systemen

14.00 - 15.30 Uhr TUTORIAL - J. N. Hooker (Carnegie Mellon University): OPTIMIZATION AND LOGICAL INFERENCE II

17.00 - 17.30 Uhr P. Barth (MPI Saarbrücken): A complete symbolic 0-1 constraint solver

17.30 - 18.00 Uhr K. Ries, R. Hähnle (Universität Karlsruhe): Prädikatenlogisches Beweisen mit gemischt ganzzahliger Optimierung - ein tableaubasierter Ansatz

18.00 Uhr PODIUMSDISKUSSION

Künstliche Intelligenz und Operations Research

A. Bockmayr* F. J. Radermacher†

1 "Two Heads Are Better than One."

Mit diesen Worten hat sich Herbert A. Simon, einer der Väter der Künstlichen Intelligenz, 1987 vor der amerikanischen Gesellschaft für Operations Research für eine engere Zusammenarbeit zwischen Künstlicher Intelligenz (KI) und Operations Research (OR) eingesetzt [15].

In ihrer Anfangszeit waren die beiden Wissenschaften sehr eng miteinander verbunden. Einige der Pioniere der KI, wie zum Beispiel Allen Newell oder Ed Feigenbaum, erwarben ihren Doktortitel mit einer Arbeit über Künstliche Intelligenz im Bereich *Industrial Administration*.

Ab etwa 1960 gingen KI und OR dann getrennte Wege. Mehrere Generationen von Wissenschaftlern wurden seither in jeweils einem der beiden Bereiche ausgebildet und nahmen den anderen oft nicht mehr zur Kenntnis. Erst ab 1980 ist wieder eine stärkere Annäherung zu beobachten, die damit zusammenhängt, daß in der KI entwickelte Expertensysteme in traditionelle Bereiche des OR vorstießen und so die beiden Forschungsgemeinden wieder aufeinander aufmerksam wurden.

Die Gründe für das Auseinanderdriften von KI und OR sind vielfältig. Die KI hat sich ab 1960 vor allem der neu entstehenden Informatik zugewandt und dabei ihre frühere Bindung an Operations Research und Managementwissenschaften weitgehend aufgegeben. Daneben hat auch der unterschiedliche Charakter der beiden Wissenschaften eine Trennung begünstigt.

Im Operations Research geht es insbesondere um die Anwendung von Optimierungstechniken oder entscheidungstheoretischen Ansätzen auf die Lösung komplexer Probleme, die über den reellen oder ganzen Zahlen ausgedrückt werden können. Im Vordergrund stehen dabei komplexe, mathematische Modelle und Fragen der Optimierung. In der Künstlichen Intelligenz hat man es mit weniger gut strukturierten Problemen zu tun, die vielfach nichtquantifizierbare Komponenten enthalten, unscharf spezifiziert sind, und zu deren Lösung große Wissensbasen und Regelkalküle erforderlich sind.

Während das Operations Research somit vorwiegend mathematisch orientiert ist und Modelle und Algorithmen mit einer reichhaltigen mathematischen Struktur untersucht, ist die Künstliche Intelligenz stärker empirisch und ingenieurmäßig ausgerichtet und nutzt weitgehend regelhafte bzw. relationale Strukturen und Zusammenhänge.

Trotz dieser Unterschiede haben Künstliche Intelligenz und Operations Research aber eine Fülle von Fragestellungen und Themen gemeinsam. Beide entwerfen Modelle ihres Problembereichs, beide verwenden zur Problemlösung heuristische Suchverfahren, wenn exakte Methoden

*Max-Planck-Institut für Informatik, Im Stadtwald,D-66123 Saarbrücken

†Forschungsinstitut für anwendungsorientierte Wissensverarbeitung, Helmholtzstr. 16, Postfach 2060, D-89010 Ulm

nicht verfügbar oder praktikabel sind, beide stützen sich auf Mathematik, beide implementieren ihre Methoden auf dem Rechner, beide brauchen den interdisziplinären Austausch mit anderen Wissenschaften.

Im folgenden sollen exemplarisch einige Themenbereiche genannt werden, die sich für eine Zusammenarbeit von KI und OR besonders anbieten.

2 Optimierung und logische Inferenz

Einer der interessantesten Berührungspunkte von OR und KI liegt in der Anwendung mathematischer Optimierungsmethoden auf logische Inferenzprobleme [9, 4].

Besonders einfach läßt sich dies im Fall der Aussagenlogik illustrieren. Aussagenlogische Inferenzprobleme können in sehr einfacher Weise in lineare 0-1 Optimierungsaufgaben übersetzt werden. Um etwa zu entscheiden, ob x_2 aus der Klauselmenge

$$\begin{array}{llllll} x_1 & \vee & x_2 & \vee & x_3, & \\ \neg x_1 & \vee & x_2 & \vee & & \neg x_4, \\ \neg x_1 & \vee & & & x_3, & \\ \neg x_1 & \vee & & & \neg x_3 \vee & x_4, \\ x_1 & \vee & & & \neg x_3 & \end{array}$$

folgt, kann man das lineare 0-1 Optimierungsproblem

Minimiere x_2 unter den Nebenbedingungen

$$\begin{array}{llllll} +x_1 & +x_2 & +x_3 & & \geq & 1, \\ -x_1 & +x_2 & & -x_4 & \geq & -1, \\ -x_1 & & +x_3 & & \geq & 0, \\ -x_1 & & -x_3 & +x_4 & \geq & -1, \\ +x_1 & & -x_3 & & \geq & 0. \end{array}$$

lösen. Die logische Folgerung gilt genau dann, wenn das Minimum von x_2 gleich 1 ist.

Dies ist nur ein einfaches Beispiel eines sehr viel tiefer reichenden Zusammenhangs zwischen Deduktion und mathematischer Optimierung: so stehen etwa die logischen Konzepte *Resolution, erweiterte Resolution, Eingabe- und Einheitsresolution, Davis-Putnam-Prozedur* oder das *Ziehen von Schlußfolgerungen in Bezug auf ein bestimmtes Thema* jeweils in enger Beziehung zu den mathematischen Begriffen *Schnittebene, Chvàtal-Methode, elementarer Abschluß, Branch-and-Bound* und *Projektion eines Polytops* [9].

Ein Resolutionsschritt zwischen zwei Klauseln $x_1 \vee x_2 \vee x_4$ und $\neg x_1 \vee \neg x_3 \vee x_4$ läßt sich beispielsweise auch durch einen Chvàtal-Gomory-Schnitt vom Rang 1 erhalten, das heißt durch eine nicht-negative Linearkombination der zugehörigen Klauselungleichungen und anschließende Rundung. Addiert man die Ungleichungen

$$\begin{array}{llllll} +x_1 & +x_2 & & +x_4 & \geq & 1 \\ -x_1 & & -x_3 & +x_4 & \geq & -1 \\ & +x_2 & & & \geq & 0 \\ & & -x_3 & & \geq & -1 \end{array}$$

mit dem Gewicht 1/2, so erhält man

$$x_2 - x_3 + x_4 \geq -1/2.$$

Eine anschließende Rundung der rechten Seite liefert die Schnittebene

$$x_2 - x_3 + x_4 \geq 0,$$

die genau der Resolvente

$$x_2 \vee \neg x_3 \vee x_4$$

entspricht.

Derartige Zusammenhänge gestatten es, Methoden der mathematischen Optimierung im Rahmen von Deduktionsverfahren einzusetzen und umgekehrt, was für beide Seiten sehr fruchtbar sein kann.

Die Anwendung mathematischer Optimierungsmethoden ist aber keineswegs auf die Aussagenlogik beschränkt. Für eine Reihe weiterer Logiken, die auch in der KI eine wichtige Rolle spielen, ist ein ähnliches Vorgehen möglich.

3 Logisches Programmieren und Constraintlösungsverfahren

Ein anderer Bereich, in dem sich das Zusammenspiel von Künstlicher Intelligenz und Operations Research als sehr fruchtbar erwiesen hat, ist das *logische Programmieren mit Constraints* (engl. constraint logic programming) [13, 1].

Die Grundidee dabei ist, eine logische Programmiersprache wie PROLOG um ein Lösungsverfahren für eine bestimmte Art von Constraints zu erweitern. Technisch geschieht dies dadurch, daß die Unifikation in Prolog durch ein geeignetes Constraintlösungsverfahren ersetzt wird. In Frage kommen dafür lineare Gleichungen und Ungleichungen über den reellen Zahlen CLP($\mathcal{R}$), Boolesche Algebra CLP($\mathcal{B}$), endliche Bereiche CLP($\mathcal{FD}$), 0-1 Constraints CLP($\mathcal{PB}$) und noch viele andere. Bei der Realisierung dieser Constraintlösungsverfahren spielen sowohl Methoden aus der Künstlichen Intelligenz als auch aus dem Operations Research eine zentrale Rolle. Das Lösen linearer Constraints über den reellen Zahlen geschieht beispielsweise meist mit einer Variante des Simplexalgorithmus, während Constraints über endlichen Bereichen in der Regel mit Konsistenztechniken aus der KI behandelt werden.

Die Vorteile einer solchen Kombination von logischer Programmierung und Constraintlösung sind vielfältig. Aus der Sicht der logischen Programmierung führt der Einsatz von Constraintlösungstechniken zu einer erheblichen Effizienzsteigerung. Aus der Sicht des Operations Research ist neben einem mathematischen Optimierungswerkzeug eine deklarative Programmiersprache verfügbar, in der viele Problemeigenschaften ausgedrückt werden können, die innerhalb des Formalismus der Optimierung keinen Platz finden. Ein weiterer Vorzug constraintbasierter logischer Programmiersprachen liegt darin, daß sie nicht nur numerische, sondern auch symbolisch repräsentierte Antworten auf Anfragen geben können.

Ein einfaches logisches Programm in der Sprache CLP($\mathcal{R}$) zur Berechnung von Hypotheken [13] lautet wie folgt (K bezeichnet das Kapital, T die Zeit in Monaten, I den Zinsatz, Z die monatliche Zahlung und R den Restbetrag):

```
mortgage(K, T, I, R, Z) :-
            T <= 1,
            R + Z = K * (1 + I).

mortgage(K, T, I, R, Z) :-
            T > 1,
            mortgage(K * (1 + I) - Z, T - 1, I, R, Z).
```

Stellt man die Anfrage

```
?- T = 5, I = 0.1, R = 0, mortgage(K, T, I, R, Z).
```

erhält man die symbolische Antwort

```
Z = 0.263797 * K,
```

die angibt, wie bei gegebener Laufzeit und Zinssatz die monatliche Zahlung vom Kapital abhängt.

Bei einer Reihe klassischer Optimierungsaufgaben ist das logische Programmieren mit Constraints bisher sehr erfolgreich eingesetzt worden [16, 5].

4 Heuristische Suche

Eine weitere Verbindung zwischen Künstlicher Intelligenz und Operations Research ergibt sich aus ihrem gemeinsamen Interessse am Lösen komplexer Probleme durch heuristische Suche [8].

Beide Disziplinen stehen vor der Aufgabe, mit der kombinatorischen Explosion fertig zu werden, die immer dann entsteht, wenn viele Einzelentscheidungen nacheinander ausgeführt werden müssen und auf diese Weise ein exponentiell wachsender Suchraum zu bewältigen ist.

In der KI tritt dieses Problem nahezu überall auf, etwa im Bereich wissensbasierter Systeme, Deduktion oder Robotik. Im Operations Research ergeben sich ganz ähnliche Schwierigkeiten bei Optimierungsaufgaben, wie sie bei Produktion, Beschaffung, Lagerhaltung, Investition oder Finanzierung auftreten.

Neben klassischen Verfahren wie Branch-and-Bound, das in der ganzzahligen Optimierung eine wichtige Rolle spielt und eng verwandt ist mit dem Prinzip des Best-First-Search aus der KI, haben sich in den letzten Jahren verschiedene neue Ansätze zur heuristischen Suche herausgebildet, die für KI und OR gleichermaßen von Bedeutung sind.

Zu nennen sind hier insbesondere

- Genetische Algorithmen
- Neuronale Netze
- Simulated Annealing
- Tabusuche
- Zielanalyse (engl. target analysis)

Während genetische Algorithmen und neuronale Netze auf biologischen Ansätzen beruhen und sich das Simulated Annealing aus der Physik ableitet, ergeben sich Tabusuche und Zielanalyse aus allgemeinen Grundsätzen intelligenten Problemlösens.

Diese Ansätze zur heuristischen Suche, die sich keineswegs ausschließen, sondern gegenseitig ergänzen, werden gegenwärtig an vielen Stellen untersucht. Parallele Entwicklungen in einer Nachbardisziplin werden dabei aber oft nur wenig oder gar nicht zur Kenntnis genommen. Für die einzelnen Gebiete können sie jedoch eine erhebliche Bereicherung darstellen.

5 Entscheidungsunterstützende Systeme

Ein letztes gemeinsames Interessensgebiet von KI und OR, das hier angesprochen werden soll, ist der Bereich entscheidungsunterstützender Systeme [11, 12].

In einer Zeit zunehmend komplexer Entscheidungen besteht ein erheblicher Bedarf an Systemen, die einen Entscheidungsträger dahingehend unterstützen, daß sie

- die Auswirkungen verschiedener zur Wahl stehender Handlungen bestimmen und
- Handlungsmöglichkeiten ermitteln, die gewisse vorgegebene Zielfunktionen optimieren.

Bei der Entwicklung solcher Systeme kann die Künstliche Intelligenz zusammen mit klassischen Disziplinen wie dem Operations Research wesentliche Beiträge leisten.

In vielen praktischen Anwendungen reichen die Techniken aus OR und KI für sich allein betrachtet nicht aus. Die Problemtypen sind meist so stark miteinander verknüpft, daß nur eine enge Kopplung der beiden Ansätze Aussicht auf Erfolg verspricht. Das gilt sowohl auf der begrifflichen als auch auf der Implementierungsebene.

Obwohl es wichtig ist, die Stärken und Schwächen des jeweiligen Ansatzes zu kennen, sollte nicht die gegenseitige Abgrenzung im Vordergrund stehen, sondern vielmehr der Versuch, die besten Errungenschaften der jeweiligen Disziplin für die konkrete Anwendung fruchtbar zu machen. Voraussetzung dafür ist eine breit angelegte interdisziplinäre Zusammenarbeit, die sicher nicht leicht ist. In der Zusammenführung statistischer Verfahren zur Datenauswertung, mathematischer Modellierungs- und Optimierungstechniken des Operations Research und wissensbasierter Methoden der Künstlichen Intelligenz liegt jedoch ein vielversprechendes Potential für die Entwicklung neuer Formen der Entscheidungsunterstützung. Zentral sind hierbei insbesondere Methodenbanksysteme, die über Regelverarbeitung in integrierter Weise die jeweils besten exakten Algorithmen zum Einsatz bringen. Typische Beispiele hierfür sind Fragen der Ablaufplanung (Scheduling) und der Tourenplanung (Travelling-Salesman-Problem).

Literatur

[1] F. Benhamou and A. Colmerauer, editors. *Constraint Logic Programming - Selected Research.* MIT Press, 1993.

[2] A. Bockmayr. Embedding OR techniques in constraint logic programming. In *Operations Research '92. 17th Symposium on Operations Research, Hamburg*, 1992.

[3] D. E. Brown and C. C. White III, editors. *Operations research and artificial intelligence: the integration of problem-solving strategies.* Kluwer, 1990.

[4] V. Chandru and J. N. Hooker. *Optimization Methods for Logical Inference.* Wiley, In preparation.

[5] M. Dincbas, H. Simonis, and P. van Hentenryck. Solving large combinatorial problems in logic programming. *Journal of Logic Programming*, 8:75–93, 1990.

[6] G. I. Doukidis and R. J. Paul. A survey of the application of artificial intelligence techniques within the OR society. *Journal of the Operational Research Society*, 41:363 – 375, 1990.

[7] F. Glover and H. J. Greenberg, editors. *Linkages with artificial intelligence*, volume 21 of *Annals of Operations Research.* Baltzer, 1989.

[8] F. Glover and H.J. Greenberg. New approaches for heuristic search: a bilateral linkage with artificial intelligence. *Europ. J. Oper. Res.*, 39:119 – 130, 1989.

[9] J. N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4:45 – 69, 1988.

[10] R. E. Jeroslow, editor. *Approaches to intelligent decision support*, volume 12 of *Annals of Operations Research.* Baltzer, 1988.

[11] R. L. Keeney, R. H. Möhring, H. Otway, F. J. Radermacher, and M. M. Richter, editors. *Design aspects of advanced decision support systems*, volume 4/4 of *Decision Support Systems.* North Holland, 1988.

[12] R. L. Keeney, R. H. Möhring, H. Otway, F. J. Radermacher, and M. M. Richter, editors. *Multi-attribute decision making via O.R.-based expert systems*, volume 18 of *Annals of Operations Research.* Baltzer, 1989.

[13] C. Lassez. Constraint logic programming. *Byte*, August 1987.

[14] R. I. Phelps. Artificial intelligence - an overview of similarities with O.R. *Journal of the Operational Research Society*, 37:13 – 20, 1986.

[15] H. A. Simon. Two heads are better than one: the collaboration between AI and OR. *Interfaces*, 17:8 – 15, 1987.

[16] P. van Hentenryck. A logic language for combinatorial optimization. *Annals of Operations Research*, 21:247–274, 1989.

Optimization and Logical Inference

J. N. Hooker
Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213 USA
john.hooker@andrew.cmu.edu

Abstract

This will be a broad survey of how optimization methods may be applied to inference problems in several types of logics. I will begin by describing the close connection between integer programming and propositional logic that allows either to be used to solve problems in the other. In particular I will show how the concepts and theory of linear relaxations, cutting planes, and facet-defining cuts in integer programming have close parallels in logic. I will describe some of the best new methods for the propositional satisfiability problem, many of which are based on or suggested by optimization techniques. Moving to first order predicate logic, I will describe "primal" and "dual" methods based on partial instantiation and recent incremental satisfiability algorithms. I will show how to use integer programming to find minimal and stable models in nonmonotonic logic. Finally, I will show briefly how linear programming provides a framework for understanding and solving several logics of uncertainty, including Boole's probability logic, Dempster-Shafer theory, second-order probability logic, and the logic of belief functions.

A complete symbolic 0-1 constraint solver

Extended Abstract

Peter Barth barth@mpi-sb.mpg.de

Max-Planck-Institut für Informatik
Im Stadtwald
66123 Saarbrücken
Germany

Introduction

In the last years much work has been spent for instantiating $\mathcal{X}$ in the CLP($\mathcal{X}$)-scheme of [10] by various computational domains. The most popular ones are $\mathcal{X} = \mathcal{R}$ (equations and inequalities over the real numbers) as in Prolog-III [6] and CLP($\mathcal{R}$) [11] and $\mathcal{X} = \mathcal{FD}$ (equations and inequalities over finite domain variables) as in CHIP [7]. Because the latter entails solving of NP-complete problems the existing constraint solvers are not complete in the sense that they do not assure satisfiability (only partial constraint solving) and no solved forms are provided. We restrict our interest to a special finite domain, namely $\mathcal{B} = \{0, 1\}$ and obtain so called *pseudo-Boolean* constraints [2]. Pseudo-Boolean constraints are equations and inequalities between pseudo-Boolean functions. A *pseudo-Boolean* function is an integer-valued function $f : \{0, 1\}^n \to \mathcal{Z}$ and thus generalizes the notion of Boolean functions. The integration of pseudo-Boolean constraints into logic programming was first introduced in [4]. We discuss a complete symbolic constraint solver for $\mathcal{X} = \mathcal{PB}$ which ensures satisfiability and provides a solved form. Such a solver has a *deductive mechanism* which works on *basic constraints* and whose output is the *solved form*. Additionally we need a transformation function $\mathcal{T}$ which transforms arbitrary pseudo-Boolean constraints into *basic constraints*. The following approaches have already been investigated [5, 4, 9, 8]

basic constraints	**deduction mechanism**	**solved form**
Boolean equations	Boolean unification	most general Boolean unifier
pseudo-Boolean equations	pseudo-Boolean unification	most general pseudo-Boolean unifier
clauses	resolution	prime implicates
extended clauses	generalized resolution	prime inequalities

where *pseudo-Boolean unification* [4] is the only one which directly works with pseudo-Boolean functions. In this paper, we concentrate on *generalized resolution* [8] and show that this is a suitable approach in the context of constraint logic programming.

Other methods coming from polyhedral theory seem to more suitable in the case of *mixed 0-1 constraints* and provide solutions for *optimization problems* which is not the case for this symbolic method [3].

The Solver

A (classical) clause is a disjunction of literals $L_1 \vee \ldots \vee L_n$ where the L_i are either finite domain variables V with associated domain $\mathcal{B} = \{0, 1\}$ or their negation $\overline{V}$. Mathematically seen such a clausal constraint is equivalent to the linear inequality $L_1 + \ldots + L_n \geq 1$ where the negated variables $\overline{V}$ are interpreted as $1 - V$. By allowing other values than 1 on the right-hand side we obtain an extended clause

$$L_1 + \ldots + L_n \geq m \tag{1}$$

which is therefore a generalization of a classical clause. With extended clauses "m out of n" problems can be formulated very naturally. Furthermore they allow a more compact representation than classical clauses.

Note that an equivalent formulation of (1) using only classical clauses and no additional variables is

$$\bigwedge_{I\subseteq\{1,\ldots,n\}:|I|=n-m+1} (\vee_{i\in I} L_i)$$

which are $\binom{n}{n-m+1}$ clauses.

We now give some basic definitions similar to [8], needed for working with extended clauses. Let in the following C, D be extended clauses and S be a set of extended clauses.

- $Ext(C) = \{X \in \{0,1\}^n | X \text{ is a solution of } C\}$; $Ext(S) = \bigcap_{C\in S} Ext(C)$
- C *dominates*[1] D if $Ext(C) \subseteq Ext(D)$; C *strictly dominates* D if $Ext(C) \subset Ext(D)$
- S *dominates* D if $Ext(S) \subseteq Ext(D)$; S *strictly dominates* D if $Ext(S) \subset Ext(D)$
- C is *prime inequality* w.r.t. S iff S dominates C and S dominates no D which strictly dominates C
- $\pi(S) = \{C | C \text{ is prime w.r.t. } S\}$

We summarize some basic results of the set $\pi(S)$. We know that $Ext(S) = Ext(\pi(S))$, that is S and $\pi(S)$ have the same set of solutions [8]. The basic property of $\pi(S)$ is stated in the following

Theorem 1 *If S dominates C then there is a single $D \in \pi(S)$ such that D dominates C.*

So, given the set $\pi(S)$, we can easily decide logical entailment and therefore satisfiability.

Corollary 1 *S is unsatisfiable iff $0 \geq 1 \in \pi(S)$*

These properties justify to further investigate $\pi(S)$ as a candidate for a *solved form* in the context of constraint logic programming with pseudo-Boolean constraints.

In [8] a deductive system called *generalized resolution* was introduced which computes $\pi(S)$ from a set of extended clauses S. The system consists of 3 transition rules. The rules **Resolution** and **Diagonal Sum** generate valid clauses (a subset of all possibly Gomory rank-1 cuts [12]), whereas **Simplification** deletes clauses which are strictly dominated by other clauses. It is shown in [8] that the repeated application of the transition rules **Resolution, Diagonal Sum** and **Simplification** in an arbitrary order to an initial set of extended clauses S until none of the rules is applicable stops with $\pi(S)$.

Note that the order of applying these rules is not essential for the theoretical completeness of the algorithm, but for the run-time behaviour of the algorithm. Obviously, it is always a good idea to simplify first. The main problem is the (possibly exponential) number of resolvents and diagonal sums which can be generated. So good heuristics need to be found.

In the context of constraint logic programming we need an incremental version of the solver. That is given a set $\pi(S)$ and a clause C we want to obtain $\pi(\pi(S) \cup \{C\})$. In the case of prime implicates this problem has been investigated in [9] and hopefully some of the ideas can be applied to generalized resolution.

As already noted we need a transformation algorithm $\mathcal{T}$ transforming arbitrary 0-1 inequalities into a set of extended clauses. We consider here only arbitrary *linear* 0-1 inequalities [1]. We want to transform a linear 0-1 inequality $I \equiv c_1 * L_1 + \ldots + c_n * L_n \geq d$, where the L_i are literals and w.l.o.g. the c_i are positive integers to a set of extended clauses S such that $Ext(S) = \{X \in \{0,1\}^n | X \text{ is a solution of } I\}$. We say that $\sum_{1\leq i\leq n} c_i * L_i \geq d$ strictly reduces to $\sum_{i\in I\subset\{1,\ldots,n\}} c_i * L_i \geq d - \sum_{i\in\{1,\ldots,n\}\setminus I} c_i$ The finite set of all strict reductions of I is denoted with $SRed(I)$. We say that $SCP(I) = \sum_{1\leq i\leq n} L_i \geq \beta$ where $\sum_{1\leq i\leq \beta-1} c_i < d \leq \sum_{1\leq i\leq \beta} c_i$ is the *strongest cutting plane* of I. Note that a cutting plane is an extended clause. In [1] we have shown that the set of extended clauses $S_I = SCP(SRed(I))$ is equivalent to I. There are several improvements possible which calculate only a subset of S_I while preserving completeness which are discussed in detail in [1].

We can use the transformation algorithm for implementing one of the basic deduction rules of the deductive system *generalized resolution*, **Resolution**. It states that R is a resolvent of two ext. clauses C_1 and C_2 iff C_i dominates a class. clause C_i' (for $i \in \{1,2\}$) and C_1' and C_2' have the classical resolvent R. As shown in [1] we have : If there is a resolvent R between C_1 and C_2 then there is an $R' \in SCP(SRed(C_1 \oplus C_2))$ such that R' dominates R, where $\oplus$ denotes linear combination.

[1] Note that domination between extended clauses is easily decidable whereas domination between arbitrary 0-1 inequalities is a NP-hard problem.

Conclusion

We have suggested to use *extended clauses* as *basic constraints* or *normal forms* of the solver, which seems to be a good compromise between computational complexity and expressiveness compared to classical clauses and arbitrary 0-1 inequalities. The *solved form*, the set of prime inequalities, provides us with an easy satisfiability and logical entailment check. We think furthermore that the solved from is quite understandable for the user because of the natural "m out of n" interpretation.

Additional problems which need to be considered are the transformation of arbitrary 0-1 constraints into extended clauses. We have sketched how to do this for linear 0-1 inequalities. This approach can also be used for non-linear constraints after applying a linearization process.

Acknowledgement

I like to thank Alexander Bockmayr for pointing out the possible relevance of [8] in the context of constraint logic programming and for his valuable comments and fruitful discussions.

References

[1] P. Barth. Linear 0-1 inequalities and extended clauses. In *Logic Programming and Automated Reasoning : international conference LPAR '93; St. Petersburg, Russia; proceedings*, July 1993.

[2] P. Barth and A. Bockmayr. Solving 0-1 problems in CLP(PB). In *Proc. 9th Conf. Artificial Intelligence for Applications (CAIA), Orlando.* IEEE, 1993.

[3] A. Bockmayr. 0-1 constraints and 0-1 optimization. In *ACCLAIM Kick-Off Workshop, Stockholm*, November 1992.

[4] A. Bockmayr. Logic programming with pseudo-Boolean constraints. In A. Colmerauer and F. Benhamou, editors, *Constraint Logic Programming - Selected Research.* MIT Press, 1992. (to appear).

[5] W. Büttner and H. Simonis. Embedding Boolean expressions in logic programming. *Journal of Symbolic Computation*, 4(2):191–205, 1987.

[6] A. Colmerauer. Introduction to PROLOG III. In *4th Annual ESPRIT Conference, Bruxelles.* North Holland, 1987.

[7] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. The constraint logic programming language CHIP. In *Fifth Generation Computer Systems, Tokyo, 1988.* Springer-Verlag, 1988.

[8] J. N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 1992.

[9] Peter Jackson. Computing prime implicates incrementally. In Deepak Kapur, editor, *11th International Conference on Automated Deduction*, pages 253–267, Saratoga Springs, NY, June 1992. Springer LNAI 607.

[10] J. Jaffar and J. L. Lassez. Constraint logic programming. Technical Report 86/73, Monash University, Victoria, Australia, June 1986.

[11] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) language and system. Technical Report RC 16292 (#72336) 11/15/90, IBM Research Division, November 1990.

[12] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Series in Discrete Mathematics and Optimization. Wiley–Interscience, 1988.

Aggregation von Präferenz-Relationen mit Neuronalen Netzen

Jürgen Bartnick
Universität Dortmund
Postfach 500 500
4600 Dortmund 50

Abstract

Ein Gremium soll vorgegebene Alternativen in eine Rangfolge bringen. Die Entscheidungsträger (Mitglieder des Gremiums) stimmen über die Alternativen ab und halten das Ergebnis in einer Abstimmungsmatrix fest. Aus der Abstimmungsmatrix, die die Anzahl der Stimmen für jedes Paar von Alternativen enthält, läßt sich eine Matrix aus Nullen und Einsen konstruieren (einfache Mehrheitsentscheidung). Hierdurch wird eine Relation beschrieben. Diese Relation muß allerdings nicht transitiv sein (liefert also nicht unbedingt eine kollektive Rangordnung). Man kann diese Relation aber als ein verrauschtes Bild einer Präferenzrelation ansehen. Mit einem Neuronalen Netz (etwa Hopfield-Netz) kann man versuchen, die wahre Präferenzrelation aus dem verrauschten Bild zu erkennen.

Rein formal kann man eine kombinatorische Optimierungsaufgabe stellen, die eine optimale Präferenzrelation beschreibt. Die Zielfunktion dieser Optimierungsaufgabe kann man als Potentialfunktion für ein Hopfield-Netz interpretieren.

Das Problem ist dann, Gewichte für das Hopfield-Netz zu finden, die zu dieser Potentialfunktion passen. Die Gewichte sollen mit Hilfe der Stimmenzahlen der Abstimmungsmatrix konstruiert werden. Da monotone Transformationen das Optimum nicht verändern, lassen sich mehr oder weniger komplizierte Potentialfunktionen angeben. Es kann mehrere (sogar sehr viele) optimale Präferenzrelationen geben. Um die Anzahl zu reduzieren, werden Indifferenzen ausgeschlossen (Stimmengleichheit soll also nicht vorkommen).

Literatur

[1] J. Bartnick. An A* Algorithm for the Aggregation of Preference Relations. *Methods of Operations Research*, 59:311–322, 1989.

[2] J. Bartnick. On the number of optimal solutions of the linear ordering problem. Extended Abstract. *Methods of Operations Research*, 63:425–426, 1990.

Maintenance Scheduling with Cumulative Constraints

N. Beldiceanu
H. Simonis

COSYTEC SA
4, rue Jean Rostand
Parc Club Orsay Université
F-91893 Orsay Cedex
France

Tel: +33 (1) 60.19.37.38
Fax: +33 (1) 60.19.36.20

Abstract

In this talk we describe the use of cumulative constraints inside the constraint logic programming language CHIP for a problem of aircraft maintenance scheduling. The maintenance periods of each aircraft of a fleet should be scheduled around given due dates, taking restrictions on available hangar space and maintenance crews of different qualification into account. Resource consumption for each service type is specified by a profile of use of the resource over time. We present the modelisation of the problem and discuss design alternatives. Different optimisation criteria can be specified, like minimising the maximum number of personnel required, maximising the number of aircrafts available, minimising the deviation from the proposed due-dates etc. Besides global optimisation, the program can be used interactively to reschedule locally, further improving resource utilisation for a limited time period. Results on real life data show that we obtain high quality solutions in a few seconds.

Heuristics for Scheduling with CLP

Silvia Breitinger and Hendrik C.R. Lock*

Abstract

Constraint Logic Programming with finite domains is proposed as a means to solve and optimize job-shop scheduling problems. The inherent complexity of such problems (NP-complete) entails very large search spaces. This conflicts with the need for quick computations in an industrial context. Although constraint techniques already reduce search spaces considerably, for large data sets the remaining search space is still intractable. Therefore we investigate flexible heuristics that guide the search towards optimal solutions.

The following is proposed: Constraints should be represented in a way that defers choices between alternative local schedules as long as possible, so that two classes of heuristics can be applied:

- application-specific heuristics that schedule those tasks first which are assigned high priority,
- selective enumeration of values which restricts the number of alternatives considered for every variable.

As a result, schedules for large data sets can be found within a minute such that the solution approximates the mathematical optimum by 8 percent.

*IWBS, WZH, IBM Informationssysteme, Vangerowstr. 18, POB 103068, D-69020 Heidelberg, Germany, email: *lock|breiting*@dhdibm1.bitnet

Genetic Algorithms and Satisfiability Problems

Dragan Cvetković*
MPI Informatik
Im Stadtwald
D–66123 Saarbücken
dragan@mpi-sb.mpg.de

1 Theory ...

Genetic Algorithms (GAs) are known as good optimization techniques and good starting points for building approximation algorithms for some NP–hard problems. We tried to apply GAs to the satisfiability problem for propositional logic in clause form (resolution–ready form) where the number of variables and clauses is quite large, and where complete procedures (Davis–Putnam, resolution, etc.) are not practically useful. Since canonical GAs ([1]) are rather slow (for this problem especially, as the fitness function is expensive and time–consuming to calculate), we combined them with some heuristics (greedy method) that makes them converge faster. The main problem is that a 99%–solution (which in most cases is relatively easy to find) is not useful: we need a 100%–solution, which may differ completely from the 99%–solution!

Our procedure for solving satisfiability problems consists of two parts: the *genetic algorithm* part and the *greedy* part. The genetic algorithm part works in the following way: first, we generate (usually randomly) some set of *chromosomes* (bit strings), which represent an assignment of variables that occur in our set of clauses. The number of chromosomes corresponds to *population size*. After generating this initial population, we repeat the following procedure until either a solution (i.e. an assignment that satisfies the given set of clauses) is found or some pre–set maximum number of iterations is reached:

1. *Select* the next generation from the current one;
2. *Recombine* chromosomes in the generation using genetic and other operators;
3. Calculate the *fitness* of the generation.

There are various techniques for performing those tasks, but we decided to use the following:

- The selection procedure is either *roulette wheel*[1] or *tournament* with the *tournament size* as a parameter;
- The operators used are *crossover* (*uniform*, *one–point* or *n–points* crossover), *mutation* and *local hill–climbing* (this greedy operator is similar to one described below: flip all bits in chromosome one after another and replace chromosome with new one, but only if the new one has *better* fitness);

*On leave from Mathematical Faculty, University of Belgrade.
[1]Formal term is *stochastic sampling with replacement.*

- The fitness function is the number of clauses which the given assignment makes true[2].

At the end of the genetic part of the algorithm, we use technique of *hill-climbing* that is used in Selman's GSAT algorithm ([5]), i.e. we perform the following optimization procedure on the best chromosome which we have obtained in the genetic part:

- Flip all bits in the chromosome obtaining k new[3] chromosomes;
- Compute the fitness of all the new chromosomes;
- Among the best chromosomes choose one *randomly*;
- Repeat the whole procedure on this chromosome.

This is repeated until either a solution is found or a pre-set maximum number of iterations is reached.

The main difference between this hill-climbing operator and the local one in the genetic part is that this one is performed even if we get chromosomes with the *same* or even *worse* fitness (so called 'sideways' and 'downward' moves). According to [4], sideways and downward moves help escaping local optima.

One further remark is that for faster convergence it is not enough to have just the number of clauses satisfied as the fitness function, because for a large number of clauses, the granularity is not fine enough. Instead, we used *scaling* together with *exponentiation*, i.e. the fitness function for a given chromosome $C \in P$ is defined as:

$$E(C) = \begin{cases} \max\{0, \frac{n(C)-(\overline{P}-a\cdot P_\sigma)}{|P|}\}^e), & \text{if } e > 0; \\ n(C), & \text{if } e = 0. \end{cases}$$

where $n(C)$ is the number of clauses which C makes true, $e \geq 0$ and a are some constants, $a \in \{0\} \cup [1,3]$, $|P|$ is the size of the population P and:

$$\overline{P} = \frac{1}{|P|} \sum_{C_i \in P} n(C_i) \ , \quad P_\sigma = \sqrt{\frac{\sum_{C_i \in P} (n(C_i) - \overline{P})^2}{|P| - 1}}$$

2 ... and results

Since the parameters of a genetic algorithm are very important for a successful run, we tried to find an optimal parameters setting. There are a lot of parameters on which a GA run may depend, but we concentrated on the following: population size, number of crossover points, crossover probability, mutation probability, size of tournament in selection routine and exponent in fitness function. As test examples we used three randomly generated sets of clauses that are considered *hard* in [3]: the first one with 50 variables and 210 clauses (`t50210`), the second one with 75 variables and 322 clauses (`t75322`), and the third one with 100 variables and 430 clauses (`t100430`). We chose those because, since GAs are stochastic processes, one has to take multiple runs (in our case, 30) for each parameter setting and choosing larger clause sets would be very time consuming. We performed tests in the following order: number of crossover points, population size, crossover probability, mutation probability, tournament size, exponent in fitness function, then again population size, crossover probability, mutation probability etc. Of course, we can't give a parameter setting for the genetic part of the algorithm that will solve *every* clause

[2] Not quite true, see below.

[3] where k is the number of variables.

set optimally (this is an approximative algorithm), but our aim is to find one that will, *in average*, perform well. Parallel to those (one–parameter–at–a–time) tests, we also ran GA for multi–parameter optimization of genetic algorithm parameters similar to those in [2], but only on the second clause set described above, i.e. `t75322`. Our main results are:

- Uniform crossover performs better then other crossover types;
- Crossover probability is a less important parameter then the mutation probability;
- Tournament as selection rule performs better then roulette wheel;
- Optimal population size is directly proportional to the number of variables;
- Other parameters tested are not so important, that is, the most important parameters are the number of crossover points, the mutation probability and the tournament size.

The first result deserves some further explanation: as already said, finding an optimal solution to a satisfiability problem is different from solving other problems because we need exact result. That often means that we have to change values of almost all variables to come from the current (suboptimal) assignment to the optimal one. As uniform crossover is more disruptive then other crossover operators, it is sometimes the right method when the population is big enough. Since larger tournament means higher selection pressure, combination with uniform crossover makes some kind of equilibrium.

The above results are given only for the genetic part of the algorithm *without* any greedy optimization or problem–specific knowledge. Our experience with the problem–specific optimization methods described above is that in general they make the process converge faster (unfortunately sometimes too fast — to some local optimum). Hill–climbing and uniform crossover are performing two opposite functions, but together they can improve our search for an optimal assignment. However, one has to be careful when combining them. Another point is that local hill–climbing is a very *expensive* operation that takes more run–time per chromosome then other operators. If we run it on every chromosome, run–time will be very long, and the program performance will decrease. It is also not necessary to apply it to every chromosome, because selection takes care of distribution of good schemata.

The conclusion is that although GAs are not the best possible method for obtaining solutions to satisfiability problem, they can, however, be used for this purpose. But, one has to be very careful in balancing parameters.

References

[1] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley, 1989.

[2] J. J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms, in *IEEE Transaction on Systems, Man and Cybernetics*, 16(**1986**), pp. 122-128.

[3] D. Mitchell, B. Selman and H. Levesque. Hard and Easy Distributions of SAT Problems, in *AAAI–92*, 1992, pp. 459–465.

[4] B. Selman and H. A. Kautz. An Empirical Study of Greedy Local Search for Satisfiability Testing, to appear in *AAAI-93*.

[5] B. Selman, H. Levesque and D. Mitchell. A New Method for Solving Hard Satisfiability Problems, in *AAAI–92*, 1992, pp. 440-446.

Reparatur von Plänen durch fallbasiertes Schließen

Jürgen Dorn, Mario Girch und Wolfgang Slany

Christian Doppler Labor für Expertensysteme, Technische Universität Wien

Verschiedene Experimente im Bereich der Künstlichen Intelligenz mit constraint-basiertem Scheduling haben gezeigt, daß eine effiziente Problemlösung darauf beruht, zuerst mit einfachen Methoden (linearer Aufwand) einen Plan zu erzeugen, und später eventuelle Verletzungen der Einschränkungen (constraints) durch Umstellungen von Jobs iterativ zu beheben (Minton et al. 1993) und (Zweben et al. 1990). Diese Feststellung ist allerdings nicht neu und so wurde bereits in Arbeiten im Bereich des Operations Research sogenannte Pertubationsverfahren entwickelt (Barker and McMahon 1985) und (Widmer and Hertz 1987), die darauf beruhen, daß ein Plan verbessert wird, indem einzelne Jobs vertauscht werden. Dieser Ansatz scheint in vielen Anwendungen deswegen effizienter, weil mögliche Lösungen des Problems nicht gleichverteilt im Lösungsraum verteilt liegen. Existiert nur eine akzeptable Lösung (die optimale), dann kann der Aufwand genauso groß sein wie bei anderen Verfahren.

Wir haben ein Expertensystem entwickelt, das diese Form des reparaturbasierten Schedulings auf Basis von Einschränkungen benutzt (Dorn and Shams 1991). Das System plant Schmelzen in einem Stahlwerk der Böhler GmbH in Kapfenberg, Steiermark ein. In einer theoretischen Arbeit haben wir die Methode weiterentwickelt und die Repräsentation von Einschränkungen auf Fuzzy-Sets abgebildet (Dorn and Slany 1993). Wir können damit einen graduellen Erfüllungsstand einer Einschränkung darstellen und haben eine Möglichkeit durch Kombination aller Verletzungen einen Wert für die Güte eines Planes zu berechnen. Während Minton et al. versuchen, durch Reparatur die Anzahl der Konflikte (Verletzungen von Einschränkungen) zu verringern, versuchen wir den Evaluationswert durch Reparatur zu verbessern. Um im folgenden darzustellen, wie wir durch fallbasiertes Schließen die Reparatur unterstützen, vereinfachen wir unser Problem etwas.

Ein Job sei im folgenden durch eine bestimmte Dauer zur Bearbeitung (möglicherweise nur ungenau bekannt), u.U. durch eine geforderte Fertigstellungszeit (due date) und gewissen Qualitätsanforderungen[1] gekennzeichnet. Die Qualitäten von zwei Jobs lassen sich vergleichen und es existiert eine Relation „subsume" die besagt, daß die eine Qualität in der anderen enthalten ist. Aus der geplanten Zeit eines Jobs und seiner Fertigstellungszeit, läßt sich entweder eine Verspätung oder eine Verfrühung berechnen. Mit der Festlegung einer Reihenfolge der Jobs sei automatisch der Beginn jedes einzelnen Jobs gegeben.

Drei prinzipielle Verletzungen von Einschränkungen werden unterschieden. Einstellige Einschränkungen schränken die Zeit ein, in der ein Job produziert werden soll. Eine Verspätung wird stärker bewertet als eine Verfrühung. Der Grad der Verletzung wird auf einen Wert zwischen 0 und 1 abgebildet, wobei eine 1 keine Verletzung bedeutet. Kompatibilitäten sind

---

1 Im Stahlwerk sind dies enthaltene chemische Elemente wie Chrom, Kobalt, ... Für manche Elemente existieren Mindestwerte, die relativ leicht einzuhalten sind, für andere existieren aber Höchstgrenzen, deren Erfüllung ungleich schwerer sind, da von Vorgängerchargen Verunreinigungen verursacht werden können.

zweistellige Einschränkungen und drücken aus wie gut zwei Jobs nacheinander passen. Die Erfüllung dieser Einschränkung ist abhängig von den Qualitäten der Jobs und wird auch wieder auf einen Wert zwischen 0 und 1 abgebildet. Zeitliche Einschränkungen zwischen zwei Jobs dienen u.a. für Sicherheitsabstände zwischen Jobs. Die Erfüllung dieser Einschränkung wird auch wieder auf einen Wert zwischen 0 und 1 abgebildet.

Wenn eine einstellige Einschränkung verletzt wird, kann diese behoben werden durch

- Herausnehmen des Jobs und das Einplanen an anderer Stelle oder
- bei Verfrühung, durch Einplanen eines zusätzlichen Jobs vor dem aktuellen Job und
- bei Verspätung, durch Herausnehmen eines früheren Jobs.

Eine Kompatibilitätsverletzung kann behoben werden, indem entweder

- ein weiterer Job zwischen beiden Jobs eingefügt wird,
- einer von beiden Jobs entfernt wird oder
- einer von beiden Jobs mit einem anderen vertauscht wird.

Die Verletzung einer zweistelligen zeitlichen Einschränkung kann dadurch behoben werden, daß entweder

- ein Job zwischen beiden Jobs eingefügt oder herausgenommen wird,
- einer von beiden Jobs entfernt wird oder
- einer von beiden Jobs mit einem anderen vertauscht wird.

Welcher dieser Methoden gewählt wird ist problemabhängig und kann die Verletzung von bisher nicht verletzten Einschränkungen hervorrufen.

Eine Reparatur eines Planes wird auch dann notwendig, wenn sich in der laufenden Produktion eine Abweichung vom Plan oder eine Störung ergibt. Mögliche Abweichungen bzw. Störungen können sein:

- Verzögerung oder Beschleunigung der Ausführung eines Jobs,
- Verletzung der Qualitätsanforderung an ein Produkt,
- Auftreten eines neuen Jobs, der auch einplant werden soll oder
- der Ausfall einer Maschine für eine gewisse Zeit.

Durch diese Ursachen können sich Verletzungen von Einschränkungen ergeben, die wiederum behoben werden sollten. Bei der Behebung einer Verletzung stehen, wie wir gesehen haben, mehrere Methoden zur Verfügung. Welche gewählt wird, hängt von der Gesamtkonstellation ab. Auch wenn zwei Jobs vertauscht werden sollen, muß noch entschieden werden, welche zwei Jobs vertauscht werden. Bei der der reparaturbasierten Strategie gehen wir davon aus, daß verschiedene Möglichkeiten generiert werden, und nur nach einer Verbesserung der Evaluierungsfunktion für den gesamten Plan, diese Vertauschung durchgeführt wird. Prinzipiell existieren bei n Jobs im Plan, $2 * (n\text{-}1)$ Möglichkeiten zur Vertauschung. Im aufwendigsten Fall existiert zwischen je zwei Jobs eine Einschränkung, so daß der Aufwand zum Finden der besten Vertauschungsposition sehr aufwendig sein kann bei großem n. Außerdem

kann es bei der reparaturbasierten Strategie passieren, daß man in ein lokales Maximum der Evaluierungsfunktion gerät.

Fallbasiertes Schließen ist eine Technik, um Problemlösungen aus alten Erfahrungen abzuleiten (Slade 1991). Fallbasiertes Schließen kann benutzt werden, um die Reparatur von Plänen durch alte Erfahrungen zu unterstützen. In der Arbeit von Miyashita und Sycara (1992) wird die interaktive Reparatur eines Benutzers als Fall repräsentiert. Bei uns ist ein Fall eine automatisch vorgenommene Reparatur, die zu einem verbesserten Evaluationswert geführt hatte. Wenn eine neue Reparatur notwendig ist, wird in einer Fallbasis nach einem ähnlichen Fall aus der Vergangenheit gesucht. Dieser Fall muß normalerweise an das aktuelle Problem adaptiert werden, da er nicht genau übereinstimmt. Der adaptierte Fall wird ausgeführt und der so reparierte Plan wird bewertet. Liefert die Evaluierung keinen guten Wert, war die Reparatur also ungenügend, wird dieses als negative Erfahrung in der Fallbasis abgespeichert. Nun wird versucht, diesen Fehler zu erklären und u.U. den Fall selbst zu reparieren. Kann keine Erklärung oder Reparatur gefunden werden, wird die inkrementelle Methode benutzt, um eine Reparatur durchzuführen.

Der beschriebene Ansatz führt zu einer effizienteren Problemlösungsstrategie und kann vor allem für die Anpassung an Änderungen in der Produktionsumgebung benutzt werden.

Referenzen

Barker, J.R. and McMahon, G.B. (1985) Scheduling the General Job Shop, Management Science 31 (5) pp. 247–264.

Dorn, J. and Shams, R. (1991) An Expert System for Scheduling in a Steelmaking Plant, *Proceedings of the 1st World Congress on Expert Systems*, Orlando Fla, Pergamon Press.

Dorn, J., Slany, W. and Stary, Ch. (1992) Uncertainty Management by Relaxation of Conflicting Constraints in Production Process Scheduling, *Proceedings of the AAAI Spring Symposium on "Practical Approaches to Scheduling and Planning"*, pp. 62–67.

Minton, S., Johnston, M., Philips, A. and Laird, P. (1990) Minimizing Conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, *Proceedings of the Artificial Intelligence* 58, pp. 161–205.

Miyashita, K. and Sycara, K. (1992) Case-Based Incremental Schedule Revision, *Proceedings of the AAAI Workshop on Knowledge-Based Production Planning, Scheduling and Control*, pp. 78–91.

Slade, S. (1991) Case-Based Reasoning: A Research Paradigm, *AI Magazine* **12** (1), pp. 42–55.

Widmer, M. and Hertz, A. (1987) *A new Approach for Solving the Flow Shop Sequencing Problem*, ORWP 87/15, Ecole Polytechnique Federale de Lausanne, Department de Mathemathiques.

Zweben, M., Deale, M. and Gargan, R. (1990) Anytime Rescheduling, *Proceedings of the DARPA Workshop on Innovative Approaches to Planning and Scheduling*, pp. 251–259.

Optimisation by Constrained Approximation

Alexander Herold, Mark Wallace and Volker Küchenhoff
ECRC, Arabellastr. 17, 81925 München

Abstract

Constraint logic programming supports search over a solution space structured into a tree, some of whose leaves are feasible solutions. The constraints allow some of the branches, whose leaves include no feasible solutions, to be pruned away in advance during the search. Many problems require not just a feasible solution but an optimal one, assuming some function associating a cost with each solution. To find the optimum some kind of enumeration of the feasible solutions is required. Traditionally branch & bound techniques are advocated in constraint logic programming to improve the enumeration efficiency. Unfortunately for certain classes of large problems enumeration techniques turn out to be computationally impractible. In case it is impractible to find the actual optimum by enumeration, approximation algorithms can be used to find, with a high probability, a good solution, by exploring only a part of the solution space. For many hard search problems, such as the travelling salesman problem, assembly-line sequencing, and scheduling, approximation algorithms have been used very successfully. Such methods are often viewed as an alternative to constrained search. However, constraints can be used with approximation algorithms in exactly the same way they are used with enumeration algorithms: the choice of approximation or enumeration is independent of the use of constraints. Experiments we have carried out show that constraints bring the same benefits of simplicity and efficiency when used with approximation algorithms that they do with enumeration algorithms.

Ein Ansatz zur Modellauswahl und -integration in Entscheidungsunterstützungssystemen

Matthias Lachmann
Institut für Wirtschaftstheorie und Operations Research
Universität Karlsruhe
Kaiserstrasse 12
7500 Karlsruhe 1

Abstract

Die im Operations Research (OR) entwickelten Modelle und Methoden sind im allgemeinen für die Lösung gut strukturierter Probleme geeignet, d.h. solcher Probleme, bei denen alle relevanten Variablen, Parameter und Ziele sowie deren Beziehungen untereinander bekannt sind. Zur Lösung praktischer Probleme ist es jedoch meistens erforderlich, das vorhandene Problem zunächst zu strukturieren, um es in einem anschliessenden Schritt mit geeigneten Methoden zu lösen. Sofern zur Lösung OR-Methoden eingesetzt werden können, steht der Anwendung jedoch die Erfahrung entgegen, dass hierzu spezielles Wissen über die Modelle, die Lösungsmethoden und deren Eigenschaften bei der Anwendung notwendig ist. Um solches OR-Expertenwissen für Nicht-Experten zugänglich zu machen, bietet sich der Einsatz eines Entscheidungsunterstützungssystems (EUS) an, das mit einer entsprechenden Modellbank von OR-Modellen ausgestattet ist. Um die so verfügbaren OR-Modelle einsetzen zu können, gibt des Systembenutzer sein jeweiliges Problem zunächst in das EUS ein. Die Beschreibung des Problems erfolgt mit Hilfe einer auf erweiterten Entity-Relationship-Modellen beruhenden graphischen Modellierungssprache. Anschliessend werden, passend zu dem eingegebenen Problem, aus der Modellbank mehrere Modelle ausgewählt, die jeweils einen Teil des gesamten Problems erfassen und mit deren Hilfe das jeweilige Teilproblem gelöst werden kann. Das EUS enthält zu diesem Zweck neben der Modellbank eine Fallbasis, in der Anwendungsschemata der OR-Modelle aus der Modellbank enthalten sind. Durch die Untersuchung von Ähnlichkeiten zwischen den Anwendungsschemata und der Problembeschreibung des Systembenutzers werden die OR-Modelle gefunden, die für die Lösung verschiedener Teile des Gesamtproblems eingesetzt werden können. In einem weiteren Schritt werden die Modelle der Teilprobleme zu einem Gesamtmodell integriert, das einen möglichst grossen Teil des Ausgangsproblems abbildet. Anschliessend kann die Lösung des Problems durch eine durch die Modellintegration bestimmte Ausführung verschiedener zu den Teilmodellen bekannten Algorithmen berechnet werden.

Iterative Tiefensuche auf massiv parallelen MIMD-Systemen

Alexander Reinefeld
PC^2 – Paderborn Center for Parallel Computing
33 095 Paderborn
ar@uni-paderborn.de

1 Einleitung

Suche und *Wissen* bilden die Grundpfeiler moderner Problemlösungssysteme. Heuristische Baum-Suchverfahren finden sich sowohl in planenden Systemen der Künstlichen Intelligenz (z.B. Robotersteuerungen, Theorembeweisern, Expertensysteme), als auch in kombinatorischen Optimierungsprogrammen der Operations Research (Vertex-Cover, TSP, etc.). Im Kern dieser Anwendungen steht ein Entscheidungsbaum, dessen Knoten Systemzustände repräsentieren und dessen Kanten mögliche Transformationen von einem Zustand in einen Folgezustand darstellen.

Die Problemlösung besteht in der Lokalisierung eines kostenoptimalen Pfades von der Wurzel (dem Ausgangszustand) zu einem Lösungsknoten (einem angestrebten Zielzustand). Im Fall der genannten Robotersteuerung könnte eine Lösung im Auffinden einer Folge von Operatoren (= Bewegungsabläufen) liegen, die einen gegebenen Ist-Zustand mit möglichst wenig Transformationen in einen gewünschten Soll-Zustand überführen.

Baumsuchprobleme sind in der Regel NP-hart, da die Größe des Suchraums exponentiell mit der Suchtiefe wächst. Unter Realzeitbedingungen ist eine exakte Lösung häufig nur durch Einsatz massiv paralleler Hardware möglich. Die Entwicklung effizienter paralleler Suchalgorithmen ist daher eine der großen Herausforderungen, von der die weiteren Fortschritte der KI und OR wesentlich abhängen.

2 Iterative-Deepening Search, IDA*

Die heuristische iterative Tiefensuche führt eine Reihe kostenbeschränkter Tiefensuchen durch, bei denen der Suchhorizont jeweils sukzessiv um ein kleines Stück erweitert wird. Sie "simuliert" gewissermaßen eine Breitensuche bei reduziertem Speicherbedarf der Größenordnung $O(depth)$, der allerdings mit einer erhöhten Knotenzahl erkauft wird.

Die iterative Tiefensuche hat sich in Optimierungsproblemen bewährt, für die keine effektive obere Suchschranke angegeben werden kann, wo sich also der Einsatz des Branch-and-Bound-Verfahrens verbietet, weil es einen zu großen Suchraum aufspannen würde. Bekannte Beispiele aus dem KI- und OR-Bereich sind das 15-Puzzle oder das Cutting-Stock-Problem.

In Analogie zum bekannten A*-Suchalgorithmus verwendet man auch in der iterativen Tiefensuche Heuristiken zur Einschränkung des Suchraums. Das resultierende Verfahren heißt *Iterative-Deepening A**, oder kurz *IDA**.

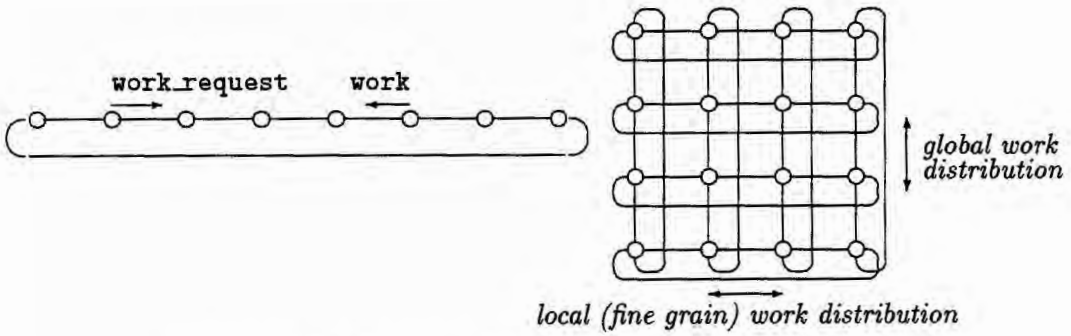

Abbildung 1: Lastverteilungsverfahren auf einem Ring bzw. Torus

3 Paralleles IDA*

Wir haben IDA* auf einem massiv parallelen (MIMD-) Transputersystem mit über eintausend Prozessoren implementiert. Unser Suchalgorithmus benutzt ein Daten-Partitionierungsverfahren mit dynamischer Auftragsanziehung. Die initiale Lastverteilung wird durch parallele (redundante) Expansion der ersten Iterationen (= oberen Baumebenen) auf allen Prozessoren erzielt. Im weiteren Verlauf der Suche arbeiten die Prozessoren auf ihren lokalen Stacks und kommunizieren nur, um überschüssige Arbeitspakete abzugeben oder neue Arbeit anzufordern. Das Lastverteilungsverfahren sorgt im Laufe der Suche automatisch für eine zunehmend bessere Lastbalancierung, weil die Datenpakete den Eigentümer dauerhaft wechseln, und die Lastverteilung so von Iteration zu Iteration immer mehr verfeinert wird.

Damit die Synchronisation zwischen den einzelnen Iterationen nicht zum Engpaß wird, haben wir eine abgeschwächte Barrier-Synchronisation eingeführt, die in ihren Grundzügen dem Terminationserkennungsalgorithmus von Dijkstra [1] folgt.

4 Effizienzergebnisse

Für unsere Leistungsmessungen stand ein Parsytec GCel-Transputersystem mit 1024 Prozessoren zur Verfügung. Als Anwendung haben wir das 15-Puzzle[1] gewählt, das mit 10^{13} Zuständen einen nicht-trivialen Suchraum aufspannt. Abbildung 2 zeigt den auf bi-direktionalen Ringen mit 32, 64 und 128 Prozessoren erzielten Speedup. Jeder Graph entspricht einer zu lösenden Random-Konfiguration des Puzzles. Zur Vermeidung von Speedup-Anomalien haben wir jeweils die gesamte Suchfront der letzten Iteration (in der die kostenoptimalen Lösungen liegen) komplett expandieren lassen.

Die Ergebnisse zeigen, daß sich trotz des großen Kommunikationsdurchmessers auch in Ring-Topologien durchaus respektable Geschwindigkeitsgewinne mit Effizienzen von über 90% erzielen lassen. Vergleichbare empirische Untersuchungen von Kumar und Rao [3] haben auf einem 128-Prozessor Intel-Hypercube einen maximalen Speedup von 25 ergeben, allerdings mit einem schlechteren Lastverteilungsverfahren. Unser Algorithmus weist eine maximale Lastdifferenz von nur 5% auf.

Nach den Voruntersuchungen auf einfachen Prozessor-Ringen verschiedener Größe haben wir unseren Suchalgorithmus auch auf enger vermaschten Torus-Topologien erprobt. Abbildung 3 zeigt die Leistungsmessungen auf Tori mit bis zu $32 \times 16 = 512$ Prozessoren. Im Mittel über 20 verschiedene Einzelversuche (hier sind nur 10 abgebildet) wurde eine Effizienz von $\approx 90\,\%$ gemessen. Die beobachtete stärkere Streuung der Einzellaufzeiten ist auf

[1] Auch als "Schiebefax" bekannt: 15 Steinchen sind auf einem 4×4-Feld horizontal oder vertikal so zu verschieben, daß mit minimaler Zuganzahl eine gegebene Zielkonfiguration erreicht wird. Das $n \times n$-Puzzle ist NP-vollständig.

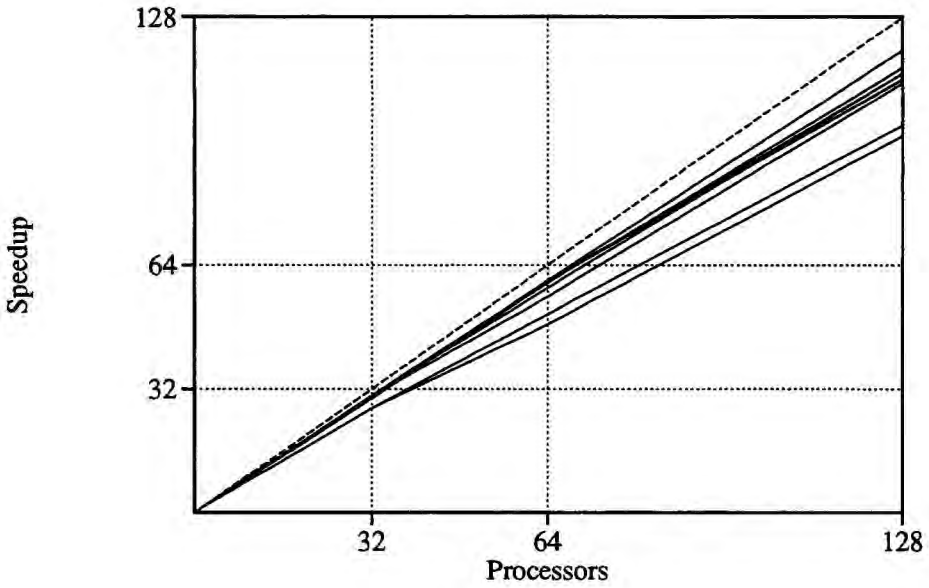

Abbildung 2: Speedup auf bi-direktionalen Ringen mit 32, 64 und 128 Knoten

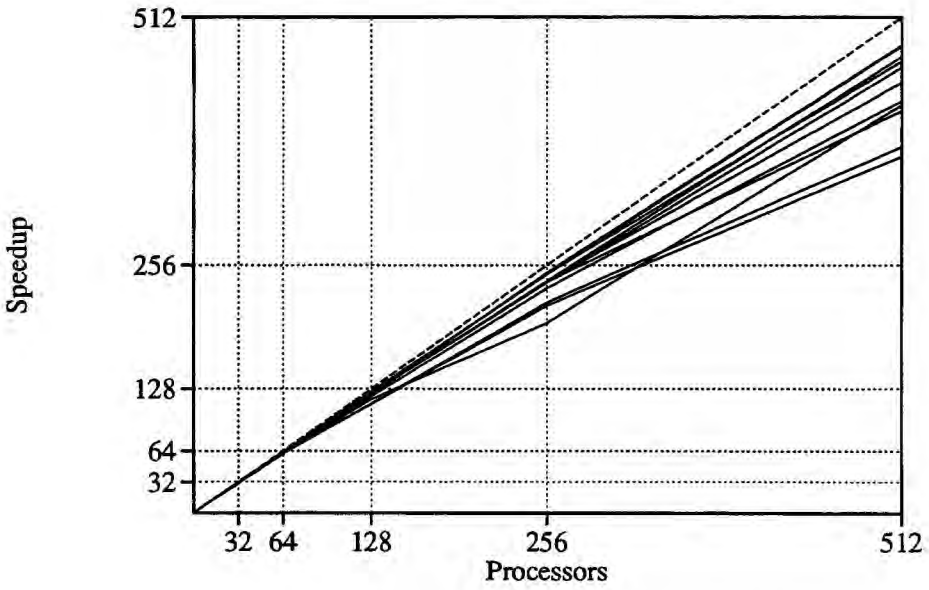

Abbildung 3: Speedup auf Tori mit bis zu 512 Prozessoren

den zunehmenden Einfluß der initialen Lastverteilungsphase zurückzuführen, in der sämtliche Prozessoren dieselben oberen Baumebenen expandieren.

5 Zusammenfassung

Wir haben einen parallelen IDA* Algorithmus mit dynamischem Auftragsanziehungsverfahren auf einem MIMD-System implementiert und auf verschiedenen Topologien erprobt:

- Selbst auf einfachen Ring-Topologien mit bis zu 128 Prozessoren (Abb. 2) liegt die Effizienz (gemittelt über 20 Random-Bäume) bei $\approx 90\,\%$.
- Auf Tori mit bis zu 512 Knoten (d.h. 16 horizontale und 32 vertikale Ringe) wurden ebenfalls über 90 Prozent Effizienz gemessen (Abb. 3).

Vergleichbare Untersuchungen auf SIMD-Systemen mit 4 bis 32 K Prozessoren [5, 4] weisen größere Synchronisationsverluste und Geschwindigkeitseinbußen durch Lastverteilung und Kommunikation auf. Die ausgefeilten Lastverteilungsverfahren in unserer Implementation haben einen wesentlichen Anteil an der Gesamtleistung. Sie sorgen bei geringem Kommunikationsaufwand für eine gleichmäßige Auslastung der Prozessoren (max 5% Lastunterschied). Effizienzergebnisse für das komplette System mit 1024 Prozessoren werden in Kürze verfügbar sein.

Literatur

[1] E. Dijkstra, W.H.J. Feijen, A.J.M. van Gasteren. *Derivation of a termination detection algorithm for distributed computation.* Information Processing Letters 16 (1983), 217 - 219.

[2] R.E. Korf. *Depth-first iterative-deepening: An optimal admissible tree search.* Artificial Intelligence 27(1985), 97–109.

[3] V. Kumar, V.N. Rao. *Scalable parallel formulations of depth-first search.* In: Kumar, Gopalakrishnan, Kanal (eds.), Parallel Algorithms for Machine Intelligence and Vision, Springer-Verlag (1990), 1–41.

[4] A. Mahanti, C.J. Daniels. *A SIMD approach to parallel heuristic search.* Artificial Intelligence 60(1993), 243–282.

[5] C. Powley, C. Ferguson, R.E. Korf. *Depth-first heuristic search on a SIMD machine.* Artificial Intelligence 60(1993), 199–242.

Prädikatenlogisches Beweisen mit gemischt ganzzahliger Optimierung
Ein tableaubasierter Ansatz

Klaus Ries, Reiner Hähnle*

Universität Karlsruhe, Fakultät für Informatik
Institut für Logik, Komplexität und Deduktionssysteme
Kaiserstr. 12, 76128 Karlsruhe
{ries|haehnle}@ira.uka.de

Zusammenfassung

Ausgehend von einem optimierten Algorithmus, der Aussagenlogik in ganzzahlige Optimierungsprobleme übersetzt, wird ein einfacher Beweiser für Prädikatenlogik entwickelt. Das dabei verwendete Verfahren ist stark an Tableauverfahren angelehnt und kompiliert γ-Formeln in eine Art Prozedur aus. Besonderer Wert wird dabei auf die Reduktion der Anzahl der zusätzlich eingeführten Variablen gelegt. Anhand von *Links* wird gezeigt, wie man mithilfe der gemischt ganzzahligen Optimierung Probleme im Bereich des automatischen Beweisens lösen kann, die einer rein logischen Modellierung nur schlecht zugänglich sind. Abschließend wird diskutiert, welche Algorithmen zur gemischt ganzzahligen Optimierung sich für die Verwendung in einem prädikatenlogischen Beweiser eignen.

1 Übersetzung von Aussagenlogik

Die Übersetzung der klassischen Aussagenlogik in ganzzahlige Ungleichungssysteme wurde bereits in [1] beschrieben und in ihrer Korrektheit und Vollständigkeit bewiesen. Als neue Elemente sollen hier die Verwendung der Verzweigungsvariablen feiner kontrolliert und eine besondere Behandlung von Disjunktionen beschrieben werden. Zur Vereinfachung der Darstellung seien die Formeln in Negationsnormalform gegeben und der Übersetzungalgorithmus wird als Regelsystem formuliert.

Definition 1 (gemischtes Boolesches Constraintssytem und Erfüllbarkeit)
Ein gemischtes Boolesches Constraintsystem ist eine Menge, die aus

- *linearen Ungleichungen in aussagenlogischen Atomen und*
- *signierten aussagenlogischen Formeln*

besteht. Eine signierte Boolesche Formel $\boxed{\geq i,I}\, F$ *besteht aus einer linearen Funktion i in aussagenlogischen Atomen, einer unendlichen Menge von Verzweigungsvariablen I, die eine Teilmenge der aussagenlogischen Atome darstellt und einer gewöhnlichen Booleschen Formel F in*

*Diese Arbeit wurde durch die DFG im Schwerpunkt Deduktion gefördert.

Negationsnormalform. Ein gemischtes Boolesches Constraintsystem heißt erfüllbar gdw. es ein aussagenlogisches Modell gibt, sodaß alle linearen Ungleichungen im Constraintsystem gelten[1] *und für alle signierten Booleschen Formeln* $\boxed{\geq i,I}\, F$ *mit* $i = 1$ *in dieser Belegung* F *wahr ist.*

Beobachtung 1 *Eine aussagenlogische Formel* F *ist erfüllbar gdw.* $\{\boxed{\geq 1,I}\, F\}$ *für ein beliebiges unendliches* I*, das keine Variablen mit* F *gemeinsam hat, erfüllbar ist.*

Die Erfüllbarkeit eines gemischten Boolschen Constraintsystem ohne signierte Formeln kann man mit den Methoden der ganzzahligen Optimierung testen.

Das nun folgende Regelsystem schreibt ein gemischtes Boolesches Constraintsystem solange um, bis es nur noch lineare Ungleichungen enthält. Die Korrektheit und Vollständigkeit der Regeln folgt unmittelbar aus [1]. *var* bezeichnet jeweils ein aussagenlogisches Atom, I_1 und I_2 sind unendliche Mengen von Verzweigungsvariablen, atomare aussagenlogische Formeln können als Disjunktionen der Länge 1 angesehen werden.

$$\frac{\mathcal{CS} \dot\cup \{\boxed{\geq i,I_1 \dot\cup I_2}\, \phi_1 \wedge \phi_2\}}{\mathcal{CS} \cup \{\boxed{\geq i,I_1}\, \phi_1, \boxed{\geq i,I_2}\, \phi_2\}} \qquad \frac{\mathcal{CS} \dot\cup \{\boxed{\geq i,\{j\} \dot\cup I}\, \bigvee(\phi_1 \dot\cup \phi_2)\}}{\mathcal{CS} \cup \{\boxed{\geq i-j,I}\, \bigvee \phi_1, \boxed{\geq i+j-1,I}\, \bigvee \phi_2\}} \qquad \frac{\mathcal{CS} \dot\cup \{\boxed{\geq i,I}\, \bigvee\{\phi\}\}}{\mathcal{CS} \cup \{\boxed{\geq i,I}\, \phi\}}$$

$$\frac{\mathcal{CS} \dot\cup \{\boxed{\geq i,I}\, \bigvee(\phi \dot\cup \{var\})\}}{\mathcal{CS} \cup \{\boxed{\geq i-var,I}\, \bigvee \phi\}} \qquad \frac{\mathcal{CS} \dot\cup \{\boxed{\geq i,I}\, \bigvee(\phi \dot\cup \{\neg var\})\}}{\mathcal{CS} \cup \{\boxed{\geq i+var-1,I}\, \bigvee \phi\}} \qquad \frac{\mathcal{CS} \dot\cup \{\boxed{\geq i,I}\, \bigvee\{\}\}}{\mathcal{CS} \cup \{0 \geq i\}}$$

Die Regeln für die Disjunktion weisen dabei folgende Eigenschaften auf

- jede CNF Formel wird in ihr "natürliches" Ungleichungssystem nach [2] umgeschrieben.
- Verzweigungsvariablen, die in einem Ast einer Disjunktion benutzt wurden, können in anderen Ästen wiederverwendet werden; dadurch wird nicht nur das Constraintsystem kompakt gehalten sondern auch die Anzahl der erfüllbaren Belegungen, die einen Ast repräsentieren.
- lange Disjunktionen können geeignet behandelt werden.
- man kann eine einfache und effiziente Implementierung der obigen Optimierungen angeben.

2 Übersetzung von Prädikatenlogik

Wir gehen davon aus, daß alle prädikatenlogischen Formeln oBdA in Negationsnormalform vorliegen und keine Existenzquantoren enthalten. Dazu wird die Definition 1 verallgemeinert zur Definition 2.

Definition 2 (Direkte Erfüllbarkeit) *Ein gemischtes prädikatenlogisches Constraintsystem ist eine Menge, die aus*

- *linearen Ungleichungen in Prädikaten und*
- *signierten prädikatenlogischen Formeln und*
- γ*-Regeln* $\gamma(X, V, \mathcal{CS})$

[1] Falls ein aussagenlogisches Atom p in einem Modell als Zahl interpretiert wird, gilt: $p = 1$, wenn p wahr ist, $p = 0$, wenn p falsch ist.

besteht. Eine signierte Boolesche Formel $\boxed{\geq^{i,I}} F$ *besteht aus einer linearen Funktion* i *in Prädikaten, einer unendlichen Menge freier Verzweigungsvariablen* I*, einer Teilmenge der Prädikate und einer gewöhnlichen prädikatenlogischen Formel* F *in Negationsnormalform ohne Existenzquantoren. Eine* γ*-Regel* $\gamma(X, V, \mathcal{CS})$ *besteht aus*

- *einer Menge* X *von Variablen für Terme*
- *einer Menge* V *von Verzweigungsvariablen*
- *einem gemischten prädikatenlogischen Constraintsystem* $\mathcal{CS}$

Ein gemischtes prädikatenlogisches Constraintsystem heißt direkt erfüllbar gdw. es ein prädikatenlogisches Modell gibt, sodaß alle linearen Ungleichungen im Constraintsystem gelten und für alle signierten prädikatenlogischen Formeln $\boxed{\geq^{i,I}} F$ *mit* $i = 1$ *in dieser Belegung auch* F *gilt.*

Wie bei der aussagenlogischen Erfüllbarkeit kann die direkte Erfüllbarkeit durch gemischt ganzzahlige Optimierung getestet werden, sofern keine signierten prädikatenlogischen Formeln mehr im Constraintsystem vorkommen. Ein Regelsystem, daß beliebige gemischte prädikatenlogische Constraintssysteme in solche ohne signierte Formeln übersetzt, erhält man, indem man neue Regeln für Allquantoren einführt und zuläßt, daß alle Regeln auch auf die Constraintssysteme anwendbar sind, die in γ-Regeln vorkommen. Dabei sollen $\vec{x}, \vec{y}$ für Vektoren von Variablen für Terme stehen, $\vec{x}\vec{y}$ sei die Konkatenation von $\vec{x}$ und $\vec{y}$.

$$\frac{\mathcal{CS} \dot{\cup} \{\boxed{\geq^{i,I}} \forall \vec{x} : A(\vec{x})\}}{\mathcal{CS} \cup \{\gamma(\vec{x}, I, \{\boxed{\geq^{i,I}} A(\vec{x})\})\}} \qquad \frac{\mathcal{CS} \dot{\cup} \{\boxed{\geq^{i,I}} \forall \vec{x} : \forall \vec{y} : A(\vec{x}, \vec{y})\}}{\mathcal{CS} \cup \{\boxed{\geq^{i,I}} \forall \vec{x}\vec{y} : A(\vec{x}\vec{y})\}}$$

$$\frac{\mathcal{CS} \dot{\cup} \{\boxed{\geq^{i,I}} \forall \vec{x} : A(\vec{x}) \wedge B(\vec{x})\}}{\mathcal{CS} \cup \{\boxed{\geq^{i,I}} \forall \vec{x} : A(\vec{x}), \boxed{\geq^{i,I}} \forall \vec{x} : B(\vec{x})\}}$$

Die direkte Erfüllbarkeit ist jedoch noch kein hinreichendes Kriterium, um die Unerfüllbarkeit einer prädikatenlogischen Formel später beweisen zu können.

Definition 3 (Erfüllbarkeit) *Ein gemischtes prädikatenlogisches Constraintsystem ist unerfüllbar gdw.*

- *es nicht direkt erfüllbar ist oder*
- *es unter einer Substitution der (freien) Variablen für Terme unerfüllbar ist oder*
- *es eine Anwendung einer* γ*-Regel gibt, sodaß das entstehende Constraintssystem unerfüllbar ist.*

Es ist erfüllbar gdw. es nicht unerfüllbar ist. Bei der Anwendung einer γ*-Regel wird ein Constraintsystem um die Menge der Constraints* $\mathcal{CS}$ *aus der* γ*-Regel erweitert, wobei die in der* γ*-Regel angegebenen Verzweigungsvariablen und Variablen für Terme durch "neue" Instanzen ersetzt werden.*

Durch eine einfach implementierbare Abwandlung der γ-Regel kann man ebenso wie im aussagenlogischen Fall eine große Anzahl von Verzweigungsvariablen einsparen.

Beobachtung 2 *Man kann jetzt einen Beweiser für Prädikatenlogik programmieren, denn*

- *eine Formel F ohne freie Variablen ist unerfüllbar gdw. das gemischte prädikatenlogische Constraintsystem $\{\boxed{\geq 1,I}\, F\}$ unerfüllbar ist*
- *die Regeln zur Umschreibung von signierten prädikatenlogischen Formeln verändern die Unerfüllbarkeit des Constraintsystems nicht*
- *Ein gemischtes prädikatenlogisches Constraintsystem ohne signierte prädikatenlogische Formeln ist unerfüllbar gdw. das folgende indeterministische Programm* **unerfüllbar** *ausgeben kann:*
  1. *Falls das Constraintsystem nicht direkt erfüllbar ist, halte mit der Meldung* **unerfüllbar** *an*
  2. *Sonst gibt es eine Belegung der in der Formel vorkommenden atomaren prädikatenlogischen Formeln, die das Constraintsystem erfüllt. Wähle deterministisch eine solche Belegung aus. Wende indeterministisch eine der folgenden Regeln an*
     - *falls es zwei unifizierbare atomaren prädikatenlogischen Formeln mit unterschiedlicher Belegung gibt, wähle indeterministisch ein solches Paar aus und führe die Substitution mit dem Unifikator auf dem gesamten Constraintsystem durch; sonst gebe* **möglicherweise erfüllbar** *aus*
     - *wende indeterministisch eine γ-Regel an; falls es keine γ-Regeln gibt, gebe* **möglicherweise erfüllbar** *aus*
  3. *gehe zu 1.*

Jeder der hier vorgeführten indeterministischen Schritte kann optimiert werden. Um Vollständigkeit zu erreichen, kann man level-saturation Strategien benutzen.

Dieses Verfahren, um Prädikatenlogik zu treiben, kann man mit jedem aussagenlogischen Verfahren kombinieren, da sich die direkte Erfüllbarkeit als SAT Problem formulieren läßt. So haben [3] ähnliche Verfahren für Shannon-Graphen angewendet.

3 Gemischt ganzzahlige Optimierung

Ausnutzung der Ausdrucksfähigkeit Die Benutzung von gemischt ganzzahliger Optimierung als Basis eines Beweisers für Prädikatenlogik kann folgende Vorteile bieten

- Erweiterung auf mehrwertige Logiken [1].
- Erweiterung der Sprache der Logik um einfache Bedingungen, die direkt als lineare Ungleichungen formulierbar sind.
- Ausnutzung der erweiterten Ausdrucksmöglichkeiten für die Steuerung des indeterministischen Programmes beim Beweisen in Prädikatenlogik.

In diesem Abschnitt soll es um ein Beispiel für letzteres gehen. In vielen Beweisern spielt der Begriff des *Links* eine wichtige Rolle. Ein Link ist ein unifizierbares Paar von atomaren prädikatenlogischen Formeln, wobei die eine in der Ausgangsformel negiert vorkam, die andere nicht[2]. In der Regel gibt es mehr als eine erfüllende Belegung für das Constraintsystem. Gesucht sei jetzt die Belegung mit den wenigsten verletzten Links, also der Links, bei denen die am Link beteiligten atomaren prädikatenlogischen Formeln unterschiedliche Belegung haben.

[2]Es wären hier einige Einschränkungen bezüglich der sinnvollen Links möglich, dies soll jedoch vernachlässigt werden

Dies läßt sich als Minimierungsproblem über die Summe der rationalen oder reelen *Linkvariablen* formulieren, wobei für jeden Link mit den Formeln $q(x)$ und $q(y)$ eine Linkvariable k und lineare Ungleichungen $k \geq q(x) - q(y), k \geq q(y) - q(x)$ eingeführt werden. Diese Summe entspricht im Optimum genau der Anzahl der verletzten Links und die Lösung des Optimierungsproblems ist genau die geforderte Belegung. Aus dem Ausgangsproblem, das noch ein ganzzahliges Erfüllbarkeitsproblem war, wird so ein ganzzahliges bzw. ein gemischt ganzzahliges Optimierungsproblem, wenn man die Forderung nach Ganzzahligkeit der Linkvariablen stellt bzw. fallenläßt[3]. Bei der Verwendung von Verzweigungsverfahren ist es sinnvoll, die Linkvariablen nicht als ganzzahlige Variablen anzugeben, da dies zum Verzweigen über Linkvariablen führen könnte.

Lösen der (gemischt) ganzzahligen Optimierungsprobleme Beim Studium des Beweisers für Prädikatenlogik fällt auf, daß der Test auf direkte Erfüllbarkeit mit einer stetig wachsenden Menge von linearen Ungleichungen geführt wird. Es liegt daher nahe, die Lösbarkeitsprüfung für solche Systeme inkrementell zu gestalten.

Cutting Plane Verfahren implementieren eine solches inkrementelles Verfahren sehr natürlich und erlauben auch die inkrementelle Lösung des Optimierungsproblems.

Verzweigungsverfahren sind von Hause aus nicht inkrementell. Wenn man aber den letzten Suchpfad, der zu einer Lösung des Problems führt, abspeichert, kann man auch in diesem Verfahren den Test auf Unerfüllbarkeit inkrementell machen, denn bereits abgelaufene Knoten, die unerfüllbar waren, können durch das Hinzufügen neuer Knoten nicht erfüllbar werden. Schwieriger ist das inkrementelle Lösen des Optimierungsproblems.

In der Praxis hat es sich gezeigt, daß für die allgemeine gemischt ganzzahlige Optimierung Verzweigungsverfahren den Cutting Plane Verfahren in der Regel überlegen sind. Übertragen auf die Welt des automatischen Beweisens ähneln Cutting Plane Verfahren Resolutionsverfahren, während Verzweigungsverfahren eine gewisse Verwandschaft mit der Davis-Putnam Prozedur aufweisen [2], die nach [4] hervorragende Resultate für das SAT Problem aufweisen.

Literatur

[1] Hähnle, Reiner, *A New Translation from Deduction into Integer Programming*, Proc. Conf. on Artificial Intelligence and Symbolic Mathematical Computations, Karlsruhe, Springer LNCS, 1992

[2] Hooker, John N., *A Quantitative Approach to Logical Inference*, Decision Support Systems **4**, 45-69, 1988

[3] Posegga, Joachim, und Ludäscher, Bertram, *Towards First-order Deduction Based on Shannon Graphs*, Proceedings 16^{th} German Workshop on Artificial Intelligence (GWAI), Bonn, Springer Lecture Notes in AI, 1992

[4] Büning, Kleine H. und Buro, M., *Report on a SAT competition*, Bericht 110, Reihe Informatik, 1992

[3] Die Linkvariablen haben im Optimum stets eine ganzzahlige Lösung.

Lösung von Constraint-Erfüllungs-Problemen mit Hilfe von Join-Trees

Martin Zahn *

Zusammenfassung

Es wird ein Constraint-Erfüllungs-System vorgestellt, das vor der Bestimmung von Lösungen einen Join-Tree ermittelt und die gerichtete strenge k-Konsistenz etabliert. Besonders sinnvoll ist die Anwendung dieses Verfahrens, wenn viele Lösungen des Constraint-Erfüllungs-Problems zu bestimmen sind.

1 Einführung

Es sei eine Menge von Variablen gegeben und jede Variable v habe die *Domäne* $\delta(v)$, welche die endliche Menge der Werte ist, die v annehmen darf. Eine *Belegung* ist eine Abbildung B, die jeder Variablen ein Element aus ihrer Domäne zuweist, d.h. $B(v) \in \delta(v)$. Ein *Constraint* C ist ein Paar $(\bar{v}, R)$, wobei $\bar{v} = (v_1, \ldots, v_n)$ ein Tupel von Variablen und $R \subseteq \delta(v_1) \times \cdots \times \delta(v_n)$ eine Relation ist; $var(C) := \{v_1, \ldots, v_n\}$ ist die Menge der Constraint-Variablen. Im folgenden sei stets ein *Constraint-Netz* N gegeben, dies ist eine endliche Menge von Constraints; $var(N) := \bigcup_{C \in N} var(C)$ ist die Menge der Constraint-Netz-Variablen. Eine Lösung des Constraint-Erfüllungs-Problems ist eine solche Belegung B, daß für alle Constraints $((v_1, \ldots, v_n), R)$ aus dem Constraint-Netz $(B(v_1), \ldots, B(v_n)) \in R$ gilt.

Ein *Hypergraph* ist ein Paar (K, H), wobei K eine endliche Menge und H eine Menge von Teilmengen von K ist; K ist die *Knotenmenge* und H die Menge der *Hyperkanten*. $\mathcal{H}_N := (var(N), \{var(C) \mid C \in N\})$ ist ein Hypergraph.

2 Wenn ein Join-Tree existiert ..

Die Bestimmung eines Join-Trees ist einfach, falls einer existiert (siehe [5]). Es gibt Verfahren zur Lösungsberechnung, deren „worst-case"-Komplexitäten nur exponentiell in k sind, sofern kein Constraint mehr als k Variablen enthält und ein Join-Tree existiert. In [3] ist eine Methode hierfür, welche wir in [6] verfeinerten, und zwar unter Verwendung der *gerichteten strengen k-Konsistenz*, die eine Verallgemeinerung der gerichteten Kanten- und Pfadkonsistenz aus [2] darstellt.

3 Wann existiert ein Join-Tree?

Es existiert genau dann ein Join-Tree, wenn die Graham-Reduktion (Algorithmus 1), angewandt auf den Hypergraphen $\mathcal{H}_N$, einen leeren Hypergraphen zurückgibt (d.h. $K = \emptyset$), siehe [1] und [5], wo

*IPF, Universität Karlsruhe, E-mail: zahn@ipf.bau-verm.uni-karlsruhe.de

man auch weitere Sätze findet, welche die Eigenschaft der Existenz eines Join-Trees charakterisieren.

procedure *Graham_reduction* $(K, H$ **in out**$)$
Wiederhole die folgenden Operationen bis
keine mehr angewandt werden kann:
(a) Existieren zwei Hyperkanten $h_1, h_2 \in H$
mit $h_1 \subseteq h_2$, dann entferne h_1 aus H.
(b) Existiert für einen Knoten $v \in K$ genau
eine Hyperkante $h \in H$ mit $v \in h$, dann
entferne v aus h und K.

Algorithmus 1: Graham-Reduktion

4 Wenn kein Join-Tree existiert ...

Durch das Hinzufügen von Hyperkanten bzw. Constraints kann man das Constraint-Netz so modifizieren, daß ein Join-Tree existiert. Eine Heuristik hierfür ist in [3], durch welche aber Hyperkanten ergänzt werden können, die unnötig viele Knoten enthalten; dieses ist ein entscheidender Nachteil. Wir entwickelten deshalb unter Verwendung des im Abschnitt 3 aufgeführten Satzes den Algorithmus 2.

procedure *extended_Graham_reduction* $(K, H$ **in**$)$
while $K \neq \emptyset$
Graham_reduction (K, H)
if $K \neq \emptyset$ **then**
Wähle $v \in K$, so daß die Hyperkante
$h' \leftarrow \bigcup_{h \in H, h \ni v} h$
möglichst wenig Elemente hat.
$H \leftarrow H \cup \{h'\}$

Algorithmus 2: Erweiterte Graham-Reduktion

Algorithmus 2 fügt dann eine Hyperkante zum Hypergraphen (K, H) hinzu, wenn weder die Reduktion (a) noch die Reduktion (b) des Algorithmus 1 durchgeführt werden kann. Nach dem Hinzufügen einer Kante kann die Graham-Reduktion stets nach mehrmaliger Anwendung der Reduktion (a) mit der Reduktion (b) fortfahren. Man beachte, daß die Erweiterung der Hyperkantenmenge um eine Hyperkante mit weniger als $\mathrm{card}(h')$ Elementen zu keinem Erfolg führen würde, d.h. Reduktion (b) bliebe unanwendbar; denn es kann genau dann die Reduktion (b) bei der nachfolgenden Graham-Reduktion durchgeführt werden, wenn eine Hyperkante hinzugefügt wird, die für ein $v \in K$ eine Obermenge von $\bigcup_{h \in H, h \ni v} h$ ist. Für die „worst-case"-Komplexität dieses Algorithmus gilt $O(\mathrm{card}(K) \cdot \mathrm{card}(H)^2)$, falls die Anzahl der Knoten pro Hyperkante durch eine Konstante beschränkt ist, siehe [6].

Algorithmus 2 haben wir für Hypergraphen formuliert. Das Constraint-Netz wird so modifiziert, daß man für jede Hyperkante, die durch den Algorithmus 2 bei der Anwendung auf den Hypergraphen $\mathcal{H}_N$ ergänzt wurde, einen Constraint zum Constraint-Netz hinzufügt, der „nichts verbietet" und dessen Constraint-Variablen-Menge gleich der hinzugefügten Hyperkante ist.

In Anbetracht von Abschnitt 2 bezeichnen wir eine solche Constraint-Netz-Erweiterung als *optimal*, wenn der Constraint mit den meisten Variablen so wenige wie möglich enthält. Daß das Problem der Berechnung einer optimalen Erweiterung NP-vollständig ist, zeigten wir in [6]. Hier findet man auch einen „branch-and-bound"-Algorithmus zur optimalen Lösung dieses Problems.

Literatur

[1] Beeri, C.; Fagin, R.; Maier, D.; Yannakakis, M.: *On the Desirability of Acyclic Database Schemes*, Journal of the ACM 30(3), 1983, 479–513

[2] Dechter, R.; Pearl, J.: *Network-Based Heuristics for Constraint-Satisfaction Problems*, Artificial Intelligence 34, 1988, 1–38

[3] Dechter, R.; Pearl, J.: *Tree-Clustering Schemes for Constraint-Processing*, Proceedings of the Seventh National Conference on Artificial Intelligence, 1988, 150–154, Menlo Park, Calif., American Association for Artificial Intelligence

[4] Freuder, E. C.: *Synthesizing Constraint Expressions*, Communications of the ACM 21(11), 1978, 958–966

[5] Maier, D.: *The Theory of Relational Databases*, Computer Science Press, Rockville, Maryland, 1983

[6] Zahn, M.: *Entwurf und Implementierung eines Expertensystemkerns zur Konfiguration – unter Verwendung eines zu entwickelnden Constraint-Erfüllungs-Systems*, Diplomarbeit am Institut für Prozeßrechentechnik und Robotik der Universität Karlsruhe, 1993