



**Systemhaus für intelligente
EDV-Anwendungen**

Richard-Wagner-Straße 91
67655 Kaiserslautern
Telefon +49 631 65566
Telefax +49 631 65464
E-Mail: sieda@sieda.com
WWW: <http://www.sieda.com>

Optimierung mit Constraintverfahren

Harald Meyer auf'm Hofe

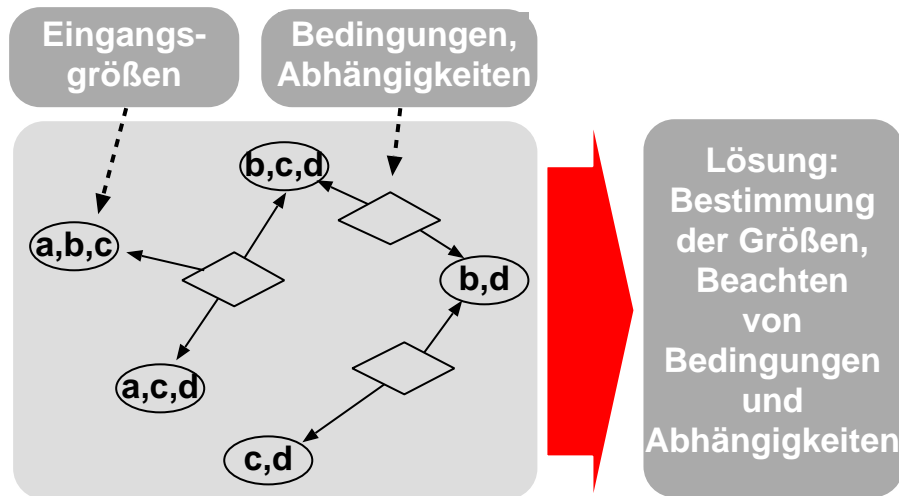
SIEDA T-2000/1

Technischer Bericht

Inhaltsverzeichnis

1	Anwendungsfelder	3
1.1	Ablaufplanung	3
1.2	Personaleinsatzplanung	5
1.3	Terminplanung	5
1.4	Budgetplanung	6
2	Wie arbeiten Constraintverfahren	8
2.1	Constraintpropagierung	9
2.2	Suchbäume	10
2.3	Lokale Suche	12
2.4	»Weiche« Constraints	13
3	ORBIS · Dienstplan: Anwendung von Constraintverfahren im Detail	15
4	Eigenschaften ConSolve	18

Constraintverfahren Kompakt



Bei der Lösung dieser Probleme bietet Ihnen Constrainttechnologie viele Vorteile, die sich im wesentlichen aus der abstrakten Modellierung des Problems durch ein *Constraintmodell* ergeben.

Vorteile der Constrainttechnologie

- Flexible Problemmodellierung:**
 Im Gegensatz zu klassischen OR-Verfahren können durch Constraints sehr viele Probleme adäquat modelliert werden.
- Kompakte Problemmodellierung:**
 Im Vergleich zu OR-Verfahren¹ ist die Erstellung und Pflege einer Problemmodellierung durch Constraints mit deutlich weniger Aufwand verbunden.
- Flexibler Einsatz:**
 Constraintverfahren eignen sich sowohl für autonome Systeme als auch für Assistenzsysteme.

¹OR: Operations research.

Kapitel 1

Anwendungsfelder

Constraintprobleme sind kombinatorische Probleme, bei denen Variablen Werte zugewiesen werden, die bestimmten Bedingungen genügen müssen oder sollen. Bestimmte Kombinationen von Werten sind verboten. Diese sehr allgemeine Problemstellung tritt in vielen Anwendungen auf, in denen *die Verwaltung von Abhängigkeiten* eine wichtige Rolle spielt.

Die Constrainttechnologie hat sich seit vielen Jahren bei der Lösung kombinatorischer Probleme bewährt und zeichnet sich durch vielseitige Anwendbarkeit aus. Die folgenden Abschnitte zeigen exemplarisch eine kleine Auswahl erprobter Anwendungsfelder.

In jedem Abschnitt werden zunächst die Randbedingungen beschrieben, die zu beachten sind. Danach wird die Struktur eines dieser Aufgabenstellung entsprechenden *Constraintmodells* beschrieben, in dem die Variablen, ihre Wertebereiche und die Constraints zwischen den Variablen identifiziert werden.

1.1 Ablaufplanung

Einfache Ressourcenplanung: Es sollen Ressourcen bestimmten Aufgaben zugeordnet werden (*resource allocation problem*). Als (einfaches) Beispiel sei z.B. die Verteilung von Facharbeitern auf Arbeitsplätze nach Abb. 1.1 genannt. Gestrichelte Pfeile stellen akzeptable Zuordnungen (schmale Linien) bzw. erwünschte Zuordnungen (fette Linien) dar, durchgezogene Linien repräsentieren *eine* Lösung des Problems. In einem entsprechenden Constraintmodell kann jeweils ein Arbeitsplatz durch eine Constraintvariable repräsentiert werden, während die für den jeweiligen Arbeitsplatz qualifizierten Facharbeiter die Menge möglicher Belegungen bilden. Sehr ähnliche Problemstellungen ergeben sich z.B. bei der Verteilung von Lehrkräften auf Kurse und viele anderen Anwendungen.

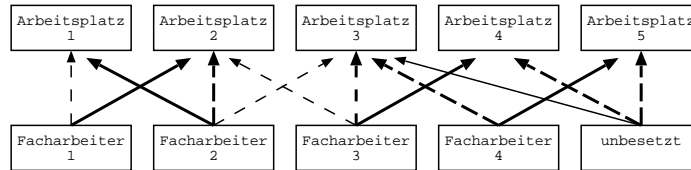


Abbildung 1.1: Beispiel zur einfachen Ressourcenplanung.

Constraints wiederum stellen sicher, dass kein Facharbeiter zwei Arbeitsplätze besetzen muss, und dass möglichst nur erwünschte Zuordnungen getroffen werden. Erwünschte Eigenschaften einer Lösung – Präferenzen – lassen sich durch »weiche« Constraints unterschiedlicher Wichtigkeit ausdrücken. Anders als Verfahren der linearen Optimierung können Constraintmodelle leicht auf Variationen in der Aufgabenstellung angepasst werden, wenn z.B. einige Facharbeiter mehrere Arbeitsplätze bedienen können oder Aufsichten für Lehrlinge eingeplant werden müssen usw.

Ablaufplanung - Ressourcen und Zeit: Bei den meisten dieser Anwendungen spielt die Zeit eine wesentliche Rolle. Abb. 1.2 zeigt eine Lösung zu einer Ablaufplanung aus der Produktionsplanung. Auf fünf Maschinen M1 bis M5 können unterschiedliche Arbeitsschritte durchgeführt werden, die eine gewisse Arbeitszeit beanspruchen. Um ein Werkstück zu fertigen, müssen i.d.R. mehrere Maschinen benutzt werden. In Abb. 1.2 werden die Arbeitsgänge für zwei Werkstücke durch unterschiedliche Schraffuren dargestellt. Pläne wie der aus Abb. 1.2 sollen optimiert werden, indem z.B. die nachgefragten Werkstücke so schnell wie möglich gefertigt werden. Hierbei sind Bedingungen bzgl. der Reihenfolge der Arbeitsschritte zu beachten. Zusätzlich müssen häufig Rüst- und Transportzeiten beachtet werden. Die Planung muss häufig sicherstellen, dass die Bedienung der Maschinen durch geeignet qualifizierte Facharbeiter erfolgt und geeignetes Material zur Fertigung bei den Maschinen vorhanden ist. Zuweilen sind auch absolute Zeitabstände zwischen den Arbeitsschritten relevant, und die gleichen Arbeitsschritte können auf unterschiedlichen Maschinen unterschiedlich viel Zeit benötigen und unterschiedliche Kosten verursachen.

Diese Anwendungen sind die erprobteste Domäne von Constraintverfahren. Im allgemeinen Falle wird ein Arbeitsschritt durch drei Variablen für die ausführende Maschine, den Startzeitpunkt des Arbeitsschrittes und die Dauer des Arbeitsschrittes beschrieben. Alle auftretenden Formen von Bedingungen an eine Lösung können durch Bedingungen über diese Variablen ausgedrückt werden.

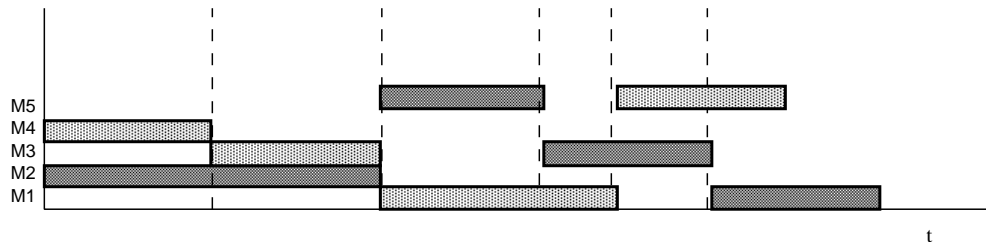


Abbildung 1.2: Ablaufplanung: Ressourcen und Zeit.

1.2 Personaleinsatzplanung

Personaleinsatzplanung kann zum unüberschaubaren Problem werden, wenn in einem Mehrschichtbetrieb verschiedene Arbeitsplätze in Abhängigkeit vom zu erwartenden Arbeitsaufkommen unter Beachtung unterschiedlicher Qualifikationen besetzt werden sollen, wobei die Möglichkeit des flexiblen Einsatzes durch Arbeitszeitkonten ausgenutzt werden soll. Wer unter diesen Umständen keine Schwierigkeiten bei der Zuweisung von Arbeitsschichten hat, beschäftigt wahrscheinlich zu viele Arbeitskräfte.

Abb. 1.3 verdeutlicht einen Schicht- oder Dienstplan. In einem Constraintmodell zu diesen Anwendungen wird für jeden Mitarbeiter an jedem Arbeitstag eine Constraintvariable vorgesehen, die mit den möglichen Arbeitsschichten belegt werden darf (in der Abbildung durch Frühschicht F1, Spätschicht S1 und Nachtschicht N2 usw. bezeichnet). Diverse Constraints, die jeweils die Variablen in einer Zeile des Dienstplanes betreffen, führen das Arbeitszeitkonto (AZK), garantieren ergonomische Schichtrhythmen und beachten arbeits- und tarifrechtliche Vorschriften. Constraints, die jeweils die Variablen in einer Spalte des Dienstplanes betreffen, stellen die Präsenz von Arbeitskräften sicher. Modellierung von Arbeitsgruppen, unterschiedlichen Qualifikationen, Betreuung von Auszubildenden und sonstigen Aufsichtspflichten ist möglich.

Die Optimierung des Dienstplanes ermöglicht auch den Einsatz z.B. von Teilzeitarbeitern in Bereichen, in denen eine Präsenz von Arbeitskräften garantiert werden muss (wichtig z.B. bei Altersteilzeitmodellen).

1.3 Terminplanung

Terminkalender sind fast immer gut gefüllt. Daher sind Terminabsprachen für ein Treffen von drei oder mehr Personen häufig ein aufreibendes Geschäft. Constraintmodelle können hier Abhilfe schaffen, indem für jeden vereinbarten Termin eine

	So	Mo	Di	Mi	Do	Fr	Sa	So	AZK
Meier	-	S1	N1	N1	N1	N2	N2	-	+30h
Vohwinkel	S1	S1	S1	S1	S1	S1	-	-	-10h
Müller	-	S1	S1	S1	S1	S1	-	-	+18h
Schulz	F2	F1	F1	F1	F1	F1	-	-	+ 5h
Lohmeyer	N2	N2	-	-	S1	S1	S2	S2	+11h
Gronau	-	F2	F2	F2	F2	F2	F2	F2	- 1h
Droste	-	N1	N1	N2	N2	-	-	N2	+ 1h
	F: 1 S: 1 N: 1	F: 2 S: 3 N: 2	F: 2 S: 2 N: 2	F: 2 S: 2 N: 2	F: 2 S: 2 N: 2	F: 2 S: 2 N: 2	F: 1 S: 1 N: 1	F: 1 S: 1 N: 1	Besetzung

Abbildung 1.3: Schichtplan bzw. Dienstplan.

Constraintvariable vorgesehen wird, die mit dem Zeitpunkt der Zusammenkunft (Datum und/oder Uhrzeit) belegt werden kann. Constraints zwischen den Terminen verhindern sich überlappende Verabredungen und berücksichtigen Präferenzen. Die Möglichkeit, »weiche« Bedingungen unterschiedlicher Priorität spezifizieren zu können, ist auch in diesen Anwendungen von besonderer Bedeutung.

1.4 Budgetplanung

Tabellenkalkulationsprogramme enthalten eine Constraintverarbeitung *der einfachsten Form*. Der Wert einer Zelle wird berechnet, wenn eine Funktion für diese Zelle spezifiziert wurde, und diese Funktion ihre Argumente kennt. Sind für jede Zelle noch zwei oder mehr Werte möglich, so passiert *nichts*.

Tabellenkalkulation wird häufig für eine *Was-wäre-wenn-Rechnung* verwandt, indem so lange an den Werten der Zelle »gedreht« wird, bis bestimmte Vorgaben erfüllt sind. In solchen Fällen spart es sehr viel Zeit, wenn die Tabellenkalkulation die *Möglichkeiten* für die Belegung einer Zelle anzeigt, anstatt mit der Arbeit solange zu warten, bis ein *eindeutiges Ergebnis* berechnet werden kann.

Abb. 1.4 zeigt, dass so etwas möglich ist, wenn die Werte einer Zelle durch eine Constraintvariable verwaltet werden. Zwischen den Zellen bestehen dann keine Funktionen zur Berechnung einzelner Werte sondern Constraints, die *mit Mengen möglicher Werte rechnen* können. Mögliche Werte werden im Beispiel durch Intervalle beschrieben. In Abb. 1.4 hat der Planer nun erweiterte Möglichkeiten, um seine Prognosen über die Kosten eines Projekts mit den Prognosen über den erreichbaren Projektetat in Einklang zu bringen. In der Abbildung ergibt sich etwa eine Schätzung von »Jahreskosten« für ein Software-Projekt aus dem durch-

	A	B	C	D
1		Schätzung		
2		Personalkosten	Laufzeit	
3	Anzahl Ent	Gehalt (D.)	[0.75 ; 1.0]	
4	[4.0 ; 5.0]	[75.0 ; 95.0]		
5		Jahreskosten		
6		[300.0 ; 466.66]		
7				
8		Projekt		
9		[225.0 ; 350.0]		

	A	B	C	D
1		Schätzung		
2		Personalkosten	Laufzeit	
3	Anzahl Ent	Gehalt (D.)	[0.75 ; 0.96]	
4	[4.0 ; 5.0]	[90.0 ; 95.0]		
5		Jahreskosten		
6		[360.0 ; 466.66]		
7				
8		Projekt		
9		[270.0 ; 350.0]		

Abbildung 1.4: Beispiel: Constraints und Tabellenkalkulation.

schnittliche Gehalt »Gehalt (D)« der Entwickler und der »Anzahl Entwickler«. Multipliziert mit der »Laufzeit« des Projekts ergeben sich unter den Stichwort »Projekt« die gesamten Personalkosten des Projekts. Beim Übergang von der linken auf die rechte Tabelle wird der Bereich der durchschnittlichen Personalkosten verändert. Folge:

1. Die Untergrenze für die »Jahreskosten« steigt.
2. Die Untergrenze für die »Projektkosten« steigt.
3. Die Obergrenze für die Projektlaufzeit fällt, da die »Projektkosten« nach oben beschränkt sind.

An jeder Stelle können die möglichen Werte durch Eingaben eingeschränkt werden. Durch Veränderungen in den anderen Zellen werden sofort die Folgen dieser Einschränkung sichtbar. Der Planer erstellt wesentlich schneller einen realistischen Kostenansatz, der den zu erwartenden Projektrahmen nicht übersteigt.

Kapitel 2

Wie arbeiten Constraintverfahren

Constraintmodelle sind abstrakt, übersichtlich, vergleichsweise schnell zu erstellen und gut zu pflegen. Die Idee der Constraintverarbeitung besteht darin, den Weg vom Constraintmodell zur effektiven Problemlösung durch Verfügbarkeit leistungsfähiger Algorithmen zu unterstützen. Dabei sind zwei Ansätze möglich:

- Die Suche nach einer passenden Belegung der Constraintvariablen wird im wesentlichen programmiert, wobei Constraintalgorithmen zur Verfügung gestellt werden. Diese Lösung erfordert viel Erfahrung bei der Erstellung und Pflege einer Problemlösung.
- Für die Suche nach einer passenden Belegung steht eine Palette von Algorithmen bereit, von denen einer ausgewählt und ggf. parametrisiert werden muss. Treten Performanzprobleme auf, so wird entweder ein anderer Algorithmus gewählt oder das Constraintmodell verändert.

Der zweite Weg ist der praktischere, weil Erstellung und Pflege einer Problemlösung wesentlich einfacher sind. Die aufwendig zu erstellenden Teile der Problemlösung, die Constraintalgorithmen, können einfach ausgetauscht werden. Bei schwierigen Optimierungsproblemen muss man jedoch leider häufig einer Mischform beider Leitbilder folgen. Daher sind Constraintverfahren durch Parameter anzupassen und an bestimmten Stellen anwendungsspezifisch erweiterbar.

Wie sehen nun diese Constraintalgorithmen aus?

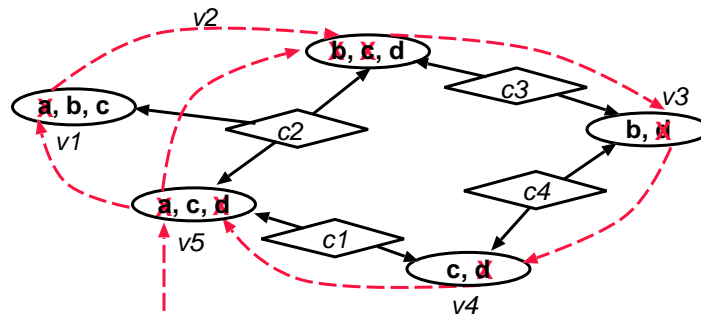


Abbildung 2.1: Filterung von Wertebereichen durch Constraintpropagierung.

2.1 Constraintpropagierung

Wie im Anwendungsszenario »Budgetplanung« aus Abschnitt 1.4 besonders deutlich wurde, besteht die Stärke von Constraintverfahren darin, mit ganzen Wertebereichen anstatt nur mit Werten zu rechnen. Dies geschieht folgendermaßen: Für jeden Constrainttyp gibt es in der Constraintbibliothek Filtermethoden, die unverträgliche Werte aus Wertebereichen entfernen können. Jeder Constrainttyp implementiert eine Form der Abhängigkeit zwischen Variablen wie zum Beispiel die Gleichheitsrelation.

Die Methoden zu einem Constrainttyp entfernen in einer Kettenreaktion nach Abb. 2.1 diejenigen Werte aus den Wertebereichen, die in keiner Lösung des Constraintproblems vorkommen können. Angenommen, Constraint c_1 in Abb. 2.1 ist ein Gleichheitsconstraint. Dann entfernen die Filtermethoden bei der Propagierung von c_1 zunächst Wert a aus Variable v_5 , weil dieser Wert in Variable v_4 nicht vorkommt. Diese Filterung zieht in der Abbildung weitere Streichungen derjenigen Werte nach sich, die nur in Verbindung mit soeben gefilterten Werten durch ein Constraint erlaubt waren. Diese Folgefilterungen sind in Abb. 2.1 durch gestrichelte Pfeile angedeutet. Es entsteht eine Kettenreaktion, die einen Großteil der Alternativen aus den Wertebereichen entfernen kann — die Suche nach einer Belegung der Variablen wird einfacher.

Durch Constraintpropagierung geht keine Lösung des Problems verloren. In den verkleinerten Wertebereichen bleiben alle Werte vorhanden, die auch in einer der möglichen Lösungen des Problems auftauchen. In der Regel ist jeder der in einem verkleinerten Wertebereich verbleibende Wert auch in einer möglichen Lösung des Problems vorhanden. Dieser Effekt wird im Anwendungsszenario »Budgetplanung« ausgenutzt.

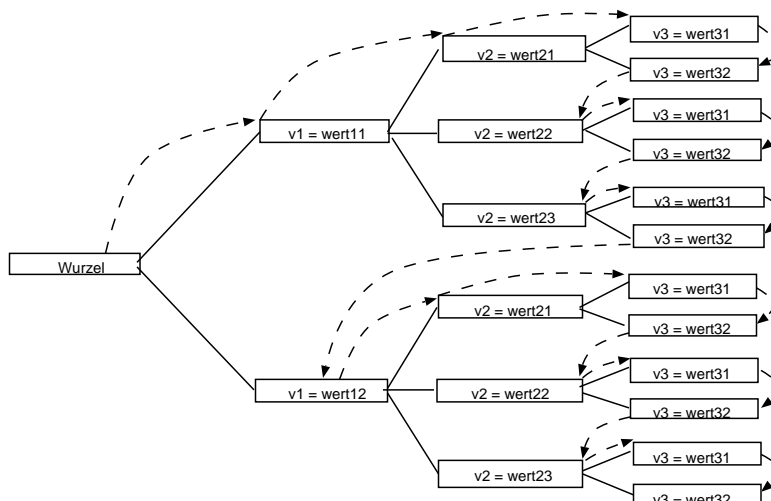


Abbildung 2.2: Prinzip der Baumsuche: Bilde alle Kombinationen von Werten.

2.2 Suchbäume

Mit der Constraintpropagierung allein lässt sich keine Lösung eines Constraintproblems berechnen, da diese aus einer Belegung *aller* Variablen mit genau *einem* Wert aus dem Wertebereich besteht, so dass alle Constraints erfüllt sind. Um eine solche Lösung zu finden, versuchen *Suchalgorithmen* einzelne Variablenbelegungen in einer Weise zu kombinieren, bis diese schließlich eine Lösung des Constraintproblems bilden.

Probiert man für jede Constraintvariable alle Werte nacheinander aus und testet danach auf Verletzung von Constraints (Bedingungen), so entstehen Suchbäume wie in Abb. 2.2. Eine Variable v_1 wird ausgewählt. Dort wird ein Wert $wert_{1,1}$ festgehalten. Danach wird eine Variable v_2 ausgewählt, wo ein $wert_{2,1}$ betrachtet wird usw. So wird nach und nach eine vollständige Belegung aller Variablen durchgeführt. Während dieses Kombinationsprozesses werden Constraints propagiert, um zu testen, inwieweit die ausgewählten Belegungen der Variablen mit den Anforderungen übereinstimmen. Werden (zu viele) Constraints nicht erfüllt, so wird ein Teil der aktuellen Belegungen rückgängig gemacht und die Suche fortgesetzt. Durch dieses *backtracking* entsteht der Suchbaum.

Suchbäume können sehr groß werden. Die Constraintpropagierung, wie im Abschnitt 2.1 beschrieben, kann jedoch die Größe der zu bildenden Suchbäume effektiv verkleinern, indem

- Wertebereiche noch zu belegender Variablen gefiltert werden,

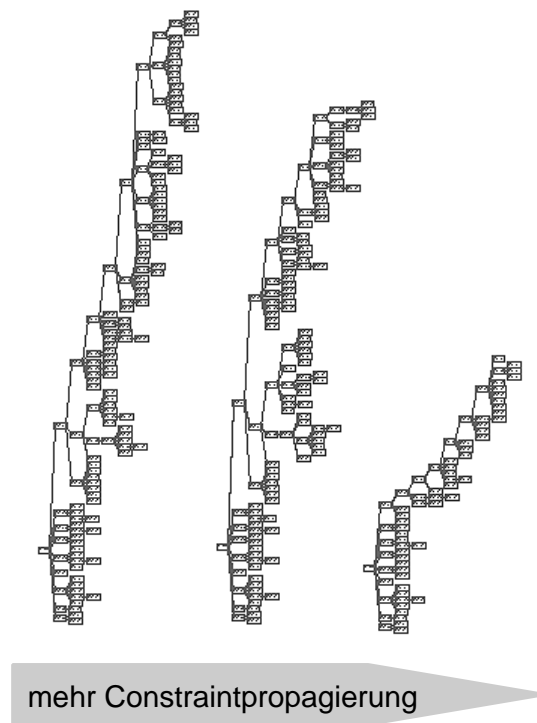


Abbildung 2.3: Mehr Constraintpropagierung — kleinerer Suchbaum.

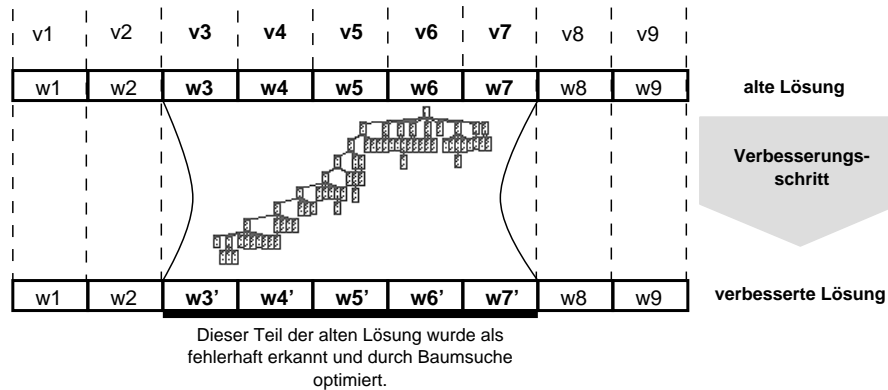


Abbildung 2.4: Schema der lokalen Suche.

- die Reihenfolge, in der die Variablen belegt werden, optimiert wird und
- die Reihenfolge, in der die Werte zugewiesen werden, optimiert wird.

Abb. 2.3 zeigt den Effekt, den die Constraintpropagierung auf die Anzahl der ausprobierten Variablenbelegungen (Knoten im Suchbaum) hat. Die Abbildung zeigt Suchbäume zu dem gleichen Problem, wobei von links nach rechts aufwendigere Formen der Constraintpropagierung verwendet wurden. Die Constraintpropagierung selbst beansprucht auch mehr oder weniger Rechenzeit — es ist eine Abwägung zwischen eigentlicher Suche und der Constraintpropagierung notwendig.

2.3 Lokale Suche

Baumsuche berechnet zwar optimale Lösungen, ist aber häufig zu aufwendig. In diesem Fall versucht man entweder, mehr Information in das Constraintmodell zu stecken (vgl. die Abschnitt 2.4), oder man wählt ein Näherungsverfahren. Näherungsverfahren können nicht die Optimalität einer Lösung nachweisen, garantieren aber mit zunehmender Berechnungsdauer immer bessere Lösungen. Verschiedene Näherungsverfahren sind bekannt, die alle nach dem gleichen Prinzip der *lokalen Suche* arbeiten. Dabei wird eine *initiale Belegung* aller Variablen erzeugt, die normalerweise noch viele Constraints verletzt, also nicht alle Anforderungen an eine Lösung erfüllt. Danach wird nach dem Schema aus Abb. 2.4 vorgegangen. Für jede Constraintverletzung wird eine Menge von Variablen ausgewählt, deren Belegungen für das Problem verantwortlich gemacht werden. Über diese Belegungen wird die Baumsuche aktiviert und, wenn möglich, dadurch eine Verbesserung der in-

itiales Belegung erzielt. Diese Verbesserungsschritte werden durchgeführt, bis die gefundene Lösung akzeptabel ist.

Zwei Näherungsverfahren namens *mincon* und *minconwalk* arbeiten automatisch ohne weitere Festlegungen. Häufig kann man allerdings leicht erkennen, welche Teile einer Belegung aller Variablen für die Verletzung von Constraints verantwortlich sind. Ist ein solches Verfahren bekannt, so kann es zur Verfeinerung einer programmierbaren Version der lokalen Suche verwendet werden, die in der Regel sehr gute Ergebnisse erzielt.

Ein wichtiger Vorteil dieser Verfahren ist, dass der aktuelle Lösungsvorschlag während der Suche zu *jedem Zeitpunkt* verändert werden kann. Daher kann es durch Bereitstellung einer geeigneten Benutzeroberfläche einem Anwender ermöglicht werden, aktiv in den Problemlösungsprozess einzugreifen. Gerade bei der Lösung von Planungsaufgaben ist dies ein nicht zu unterschätzender Vorteil.

2.4 »Weiche« Constraints

In den Anwendungsfeldern aus Abschnitt 1 traten häufig *weiche Constraints* auf, um u.a. folgende Zusammenhänge zu beschreiben:

- Planung von Fertigungsabläufen: Facharbeiter 2 arbeitet möglichst an Arbeitsplatz 2 (vgl. Abschnitt 1.1).
- Personaleinsatzplanung: Erreibe so oft wie möglich die präferierte Größe einer Arbeitsgruppe (vgl. Abschnitt 1.2).
- Terminplanung: Halte möglichst 20 Minuten zwischen zwei Terminen frei. Für Termin »Gruppenbesprechung« soll der Raum 372 bevorzugt werden.
- ...

Diese weichen Constraints werden mit »Strafpunkten« versehen, die ein Maß für Kosten oder Präferenz darstellen. Die Lösung eines Constraintproblems darf zwar weiche Constraints verletzen. Die verletzten Constraints sollen jedoch so wenige Strafpunkte wie möglich aufweisen. Weiche Constraints bringen also zu den absoluten Anforderungen den Aspekt der Optimierung ins Spiel.

Zwei Formen von Strafpunkten lassen sich unterscheiden:

Hierarchieebene: Weiche Constraints werden in Hierarchieebenen eingeteilt. Jedes Constraint einer höheren Hierarchieebene ist wichtiger als *alle* Constraints niedrigerer Hierarchie zusammengekommen.

Gewicht: Innerhalb einer Hierarchieebene wird weichen Constraints ein Gewicht zugeordnet. Die Suchalgorithmen minimieren die Gewichtssumme der verletzen Constraints.

Wozu diese Unterscheidung? Constraintgewichte eignen sich zur Darstellung von Kosten in Constraintmodellen. Mit ihnen kann auch spezifiziert werden, so viele Anforderungen wie möglich zu erfüllen. Constraintgewichte stellen eine *graduelle* Unterscheidung der Wichtigkeit von Anforderungen dar.

Hierarchieebenen sind für kategorische Unterscheidungen geeignet. Nehmen wir an, Sie haben eine Anwendung, in der es sowohl *kostenrelevante* Anforderungen als auch »*nice to have*« Wünsche gibt. Die Optimierung der Kosten geht vor, so dass die kostenrelevanten Anforderungen in eine höhere Hierarchieebene gruppiert werden.

Hierarchieebenen sind ebenfalls wichtig, um die Suche effizienter zu machen. Angenommen, für die Planungsaufgabe in Ihrer Anwendung sind (Teil-)lösungen bekannt, die häufig auftreten. Wenn sie weiche Constraints in das Constraintmodell integrieren, die die Verwendung dieser Teillösungen präferieren, wird *jeder* Suchalgorithmus zunächst die Routinelösungen ausprobieren. Logischerweise ist die Verträglichkeit mit Routinelösungen weniger wichtig als *jede* Anforderung, so dass Routinelösungen in eine eigene und unwichtige Hierarchieebene eingeordnet werden.

Kapitel 3

ORBIS · Dienstplan: Anwendung von Constraintverfahren im Detail

Das ORBIS · Dienstplan-System¹ ist ein prototypisches Beispiel für eine erfolgreiche Anwendungslösung, deren Funktionalität ohne den Einsatz von Constraintverfahren nicht realisiert werden kann.

ORBIS · Dienstplan ist eine Dienstplanungssoftware unter WINDOWS 95TM und WINDOWS NTTM, die folgende Leistungsmerkmale besitzt:

- Automatische Dienstplanung,
- Istdaten-Erfassung einschließlich Ruf- und Bereitschaftsdienste für den Pflegebereich,
- Führen von Zeit- und Überstundenkonten,
- Berechnung relevanter Arbeitszeiten für die Lohn- und Gehaltsabrechnung,
- Einbettung in die bestehende EDV-Umgebung.

Als erstes Produkt in Deutschland führt **ORBIS · Dienstplan** eine **automatische Dienstplanung** durch, indem es aus gegebenen Mitarbeiter- und Stationsdaten mit einem in der Software enthaltenen »Expertenwissen« Dienstpläne »auf Knopfdruck« erzeugt. Die erzeugten Dienstpläne müssen dabei einer Anzahl formaler und qualitativer Kriterien genügen:

¹Früherer Name: SIEDAplan.

Dienstplangenerierung											
Station: Station2		von: 01.06.1997 bis: 30.06.1997									
	ZK	So 1	Mo 2	Di 3	Mi 4	Do 5	Fr 6	Sa 7	So 8	Mo 9	Di 10
Baumann Fathimeh	1h54	N2	*	-	-	-	-	-	-	-	-
Braun Svetlana	-29h36	-	-	-	S2	S2	S2	S2	S2	T2	T2
Conradi Doris	41h18	S2	F2	F2	*	F2	F2	S2	S2	S2	S2
Ehrenbrecht Susanna	26h27	-	T2	T2	T2	T2	T2	-	-	T2	T2
Kaufmann Andrea	-22h42	S2	T2	T2	T2	*	T2	-	-	N2	N2
Kober Rita	-16h12	-	S2	S2	S2	S2	S2	-	-	-	-
Kunstmann Roswitha	-15h24	-	T2	T2	-	T2	T2	-	-	T2	T2
Lieser Monika	-1h12	-	T2	T2	T2	T2	T2	-	-	S1	S1
Marovski Michaela	-7h42	-	F2	F2	F2	-	N2	-	-	S2	S2
Melchowski Katharin	14h54	N2	*	N2	N2	-	-	-	N2	*	-
Müller Claudia	-5h24	F2	S2	S2	S2	S2	S2	N2	-	F2	F2
Panterra Sieglinde	7h42	F2	*	-	F2	F2	F2	-	F2	F2	F2
Paskovic Eva	13h30	-	N2	N2	N2	N2	N2	N2	N2	*	-
Schmidt Rolf	-35h15	S2	*	-	T2	T2	-	F2	S2	N2	N2
Weidmann Ulla	11h10	-	N2	-	-	N2	-	F2	F2	*	-
Winter Ramona	27h16	-	S1	S1	-	-	-	S1	-	T2	T2

Station2: Besetzungsdifferenz										
Anzeige	1	2	3	4	5	6	7	8	9	10
F	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0

Abbildung 3.1: Anzeige des Dienstplanes und der Differenz der tatsächlichen zur erwünschten Besetzung (Ausschnitt).

- Sämtliche gesetzlichen Vorgaben und tarifvertraglichen Regelungen sind einzuhalten (z.B. Arbeitszeitgesetz, Mutterschutzgesetz, ...).
- Die Besetzungsvorgaben sind zu erreichen, sofern dieses mit der gegebenen Personalstärke möglich ist.
- Ein Ausgleich der Zeitkonten der Mitarbeiter ist anzustreben.
- Eine Vielzahl qualitativer Kriterien wie Arbeitsrythmus, Berücksichtigung von Mitarbeiterwünschen oder die harmonische Belastung der Mitarbeiter sind zu beachten.

Für die Beurteilung der Qualität von Dienstplänen lassen sich nicht ausschließlich objektive Kriterien nennen. Zu divergent sind die Anforderungen unterschiedlicher Benutzer an die generierten Pläne. Auch unterscheidet sich die Priorität der

oben genannten Kriterien je nach Benutzer und Einsatzbereich des Systems erheblich. Diesen Problemen begegnet **ORBIS · Dienstplan**, indem sich die Generierung nach den Wünschen des Benutzers konfigurieren lässt. Dies bedeutet, dass die Priorität der Dienstplanungskriterien geändert werden kann und neue Kriterien einfach in das bestehende System integrierbar sind.

Neben einem automatischen Modus besitzt **ORBIS · Dienstplan** einen **Anweisungsmodus** und einen **manuellen Modus**. Im Anweisungsmodus können Dienstplanänderungen vom Benutzer durchgeführt werden. Aufgabe der Software ist es anschließend, aus den Änderungen resultierende notwendige Umstellungen im Dienstplan vorzunehmen wie z.B. Besetzungsanpassungen, um einen korrekten Dienstplan wiederherzustellen. Im Anweisungsmodus sind unterschiedliche Korrekturstrategien der Software denkbar wie z.B. minimale Umstellung des Dienstplanes oder bestmögliche (ausgewogenste bezüglich der Zeitkonten).

Im manuellen Modus sind sämtliche automatischen Funktionen der Software abgeschaltet. Das Verhalten entspricht nun einem herkömmlichen Dienstplanungssystem, das lediglich eine Planungsunterstützung durch Konsistenztests bietet.

Zusammenfassend lassen sich folgende Vorteile von **ORBIS · Dienstplan** gegenüber herkömmlichen Dienstplanungssystemen nennen:

- Das System kann korrekte und optimierte Dienstpläne innerhalb von 5 bis 10 Minuten generieren.
- Sofern der Benutzer nicht den manuellen Modus verwendet, werden ausschließlich korrekte Dienstpläne erzeugt, d.h. die generierten Dienstpläne genügen sämtlichen gesetzlichen Bestimmungen und tarifvertraglichen Regelungen. So können z.B. Dienstpläne erzeugt werden, in denen bereits Ausgleichstage für Sonntagsarbeit und für Feiertagsarbeit (sofern der Mitarbeiter Freizeitausgleich wünscht) unter Wahrung der Ausgleichsfrist eingetragen sind.
- Bei der Generierung werden sämtliche Kriterien der Dienstplanung beachtet. Die Gefahr, Anforderungen beim Erstellen des Dienstplanes zu vergessen, wie z.B. ein bestimmter Mitarbeiterwunsch, besteht nicht.
- Durch die automatische Planung werden wesentliche Zeitgewinne erreicht.
- Änderungen im Dienstplan, z.B. bei kurzfristigen Krankmeldungen, werden durch den Anweisungsmodus erleichtert.

Kapitel 4

Eigenschaften *ConSolve*

Features:

- Stellt folgende Typen von Constraintvariablen zur Verfügung:
 - Endliche, symbolische.
 - Intervalle von Fließkommazahlen.
- Harte, sogenannte weiche *und unscharfe* Constraints ermöglichen die Repräsentation praktisch aller Optimierungsprobleme.
- Folgende Constraintalgorithmen sind verfügbar:
 - Berechnung von mit harten [2] *und weichen* Constraints [16] kantenkonsistenten Domänen.
 - *Branch&Bound* Baumsuche erweitert um diverse Constraintverfahren [3].
 - lokale Suchalgorithmen *mincon* [14] und *minconwalk* [17].
 - Eine programmierbare lokale Suche [13].
- Einbindung in C/C++ oder JAVA (JNI) als DLL- bzw. ELF-Bibliothek.

Ausblick: Gegenwärtig findet eine vollständige Neuentwicklung des Moduls zur Constraintoptimierung in **ORBIS · Dienstplan** (vgl. Kapitel 3) statt. Hauptsächlich Zielrichtungen:

- Flexiblere Repräsentation der kombinatorischen Problemstellung durch Konzeptbeschreibungen [10].

- Integration effizienter Verfahren für die Optimierung von Problemen spezieller Eigenschaften [11].

Resultat wird ein flexibles System für Planungs- und Optimierungsprobleme sein, dass aufgrund neuer Techniken noch effektiver arbeitet als das gegenwärtig eingesetzte System.

Literaturverzeichnis

- [1] A. Abecker, H. Meyer auf'm Hofe, J. P. Müller und J. Würtz (Hrsg.). *Notes on the DFKI-Workshop: Constraint-Based Problem Solving*, Document D-96-02. Deutsches Forschungszentrum für Künstliche Intelligenz, 1996. 20
- [2] Y. Deville und P. Van Hentenryck. An Efficient Arc Consistency Algorithm for a Class of CSP-Problems. Aus: *Proceedings of the 12th IJCAI*, Sydney, 1991. 18
- [3] Eugene C. Freuder und Richard J. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21-70, 1992. 18
- [4] Michael Jampel, Eugene Freuder und Michael Maher (Hrsg.). *Workshop Notes CP95 Workshop on Over-Constrained Systems*, Cassis, France, 1995. 21
- [5] Harald Meyer auf'm Hofe und Andreas Abecker. Zur Verarbeitung "weicher" Constraints. *KI – Künstliche Intelligenz*, (4):31-36, 1997.
- [6] Harald Meyer auf'm Hofe und Andreas Abecker. ConPlan: Solving Scheduling Problems Represented by Soft Constraints. Aus: *ISIAC-98: Proceedings of the Int' Symp' on Intelligent Automation and Control*, Seiten: 245-250, Anchorage, Alaska, USA, 1998.
- [7] Harald Meyer auf'm Hofe. Representation of Requirements Through Preference Orderings of Soft Constraints. Aus: [1], Januar 1996.
- [8] Harald Meyer auf'm Hofe. ConPlan/ SIEDAplan: Personnel Assignment as a Problem of Hierarchical Constraint Satisfaction. Aus: *PACT-97: Proceedings of the Third International Conference on the Practical Application of Constraint Technology*, Seiten: 257-272, Practical Application Expo, London, April 1997.
- [9] Harald Meyer auf'm Hofe. Finding Regions for Local Repair in Hierarchical Constraint Satisfaction. Research Report RR-97-05, Deutsches Forschungszentrum für Künstliche Intelligenz, Dezember 1997.
- [10] Harald Meyer auf'm Hofe. *ConStruct*: Combining Concept Languages with a Model of Configuration Processes. Aus: Boi Faltings, Eugene Freuder, Gerhard Friedrich und Alexander Felfernig (Hrsg.), *Configuration: Papers from the 1999 AAAI Workshop*, Seiten: 17-22. AAAI Press, Menlo Park, CA, 1999. 18
- [11] Harald Meyer auf'm Hofe. Benefits and Problems of Using Cycle-Cutset Within Iterative Improvement Algorithms. Aus: *ECAI-2000 Workshop: PuK-2000 – Planen und Konfigurieren*, Berlin, August 2000. 19
- [12] Harald Meyer auf'm Hofe. *Kombinatorische Optimierung mit Constraintverfahren – Problemlösung ohne anwendungsspezifische Suchstrategien*, Band 242 aus DISKI

- *Dissertationen zur Künstlichen Intelligenz*. Infix, akademische Verlagsgesellschaft, 2000. ISBN 3-89838-242-7.
- [13] Harald Meyer auf'm Hofe. Solving Rostering Tasks as Constraint Optimization. Aus: *PATAT-2000: Practical Applications and Theory of Automated Time-Tabling*, Seiten: 280–298, Konstanz, August 2000. [18](#)
- [14] Steven Minton, Mark Johnston, Andrew Philips und Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction problem and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992. [18](#)
- [15] Harald Meyer auf'm Hofe und Enno Tolzmann. ConPlan/ SIEDAplan: Personaleinsatzplanung als Constraintproblem. *KI – Künstliche Intelligenz*, (1):37–40, 1997.
- [16] Paul Snow und Eugene C. Freuder. Improved Relaxation and Search Methods for Approximate Constraint Satisfaction with a Maximin Criterion. Aus: *Proc. of the 8th biennial conf. of the canadian society for comput. studies of intelligence*, Seiten: 227–230, Mai 1990. [18](#)
- [17] Richard J. Wallace und Eugene C. Freuder. Heuristic Methods for Over-Constrained Constraint Satisfaction Problems. Aus: [\[4\]](#), Seiten: 97–101, 1995. [18](#)