Redundante Constraints und Heuristiken zum effizienten Lösen von Problemen der Ablaufplanung mit CHIP[1]

Hans-Joachim Goltz
GMD – Forschungszentrum Informationstechnik GmbH
Forschungsinstitut für Rechnerarchitektur und Softwaretechnik
GMD-FIRST, Rudower Chaussee 5, 12489 Berlin
e-mail: goltz@first.gmd.de

Zusammenfassung

Methoden zum Erzeugen optimaler Lösungen für $n \times m$ Maschinenbelegungsprobleme mit der constraintlogischen Programmiersprache CHIP werden beschrieben, wobei CHIP als "Blackbox"-System betrachtet wird. Zuerst werden redundante Constraints angegeben, durch die der Suchraum verkleinert werden kann. Da das Erzeugen aller möglichen redundanten Constraints nicht möglich ist, werden Heuristiken benötigt, die bestimmen, für welche Aufgabenmengen welche redundante Constraints erzeugt werden. Außerdem werden Heuristiken für die schrittweise Erzeugung der Bearbeitungsreihenfolge der Arbeitsaufgaben diskutiert. Ziel unserer Untersuchungen ist dabei, Methoden zu entwickeln, die in unserem Planungungstool integriert werden können, so daß in den relevanten Fällen bessere Lösungen als mit den allgemeineren Suchmethoden erzeugt werden können.

1 Einleitung

Die constraint-logische Programmierung (CLP) ist eine Erweiterung der logischen Programmierung um die effiziente Verarbeitung von Constraints und vereinigt die Deklarativität der Logischen Programmierung mit der Effizienz der Constraint-Programmierung. Die constraint-logische Programmierung mit linearen Constraints über endlichen Domänen (CLP(FD)) kann als praktisches Tool für das Lösen diskreter Planungsprobleme erfolgreich eingesetzt werden (siehe z.B. [DSvH90, AB93, BL94b, CL94]).
Die $n \times m$ Maschinenbelegungsprobleme bilden eine Klasse von Ablaufplanungsproblemen, die intensiv untersucht wurden. Das von *J.F. Muth* und *G.L. Thompson* in [MT63] definierte 10×10 Maschinenbelegungsproblem wird oft als Benchmark verwendet. Obwohl dieses Problem bereits 1963 formuliert und seitdem umfangreich untersucht wurde, konnte erst 1989 in [CP89] ein Algorithmus für die Erzeugung der optimalen Lösung repräsentiert werden. Inzwischen existieren verschiedene Veröffentlichungen, die Methoden für das Finden optimaler Lösungen dieser Problemklasse beschreiben. In [CL94] und [Col96] sind beispielsweise solche Methoden für eine CLP-Implementation enthalten.

[1] CHIP ist ein Produkt der Firma COSYTEC, Frankreich

In praktischen Anwendungen existieren sehr verschiedenartige Probleme der Ablaufplanung mit unterschiedlichen Optimierungszielen. Das Hauptziel unserer Untersuchungen ist deshalb die Entwicklung möglichst allgemeiner Methoden und Heuristiken für die Erzeugung guter Lösungen für Probleme der Ablaufplanung, d.h. Lösungen, die nahe der optimalen Lösung sind. In [Gol95] haben wir die Strategie der Domänenreduzierung für die Lösungssuche von Problemen, die in CLP(FD) formuliert sind, vorgestellt. Die Grundidee besteht darin, daß in einer bestimmten Reihenfolge die Domänen der Variablen eingeschränkt werden, anstatt ihnen Werte zuzuordnen. Dieses Suchverfahren kann für Probleme verschiedener Klassen gute Lösungen erzeugen. Andererseits wäre es auch vorteilhaft, wenn Methoden zur Verfügung stehen, die es für bestimmte Problemklassen ermöglichen, noch bessere Lösungen zu erhalten.

Methoden zum Erzeugen optimaler Lösungen für $n \times m$ Maschinenbelegungsprobleme mit der constraintlogischen Programmiersprache CHIP stellen wir in dieser Arbeit vor, wobei CHIP als "Blackbox"-System betrachtet wird. Die Erfahrungen aus diesen Untersuchungen werden wir für das allgemeinere Planungungstool, das wir in CHIP implementierten, nutzen. Diese Methoden können beispielsweise auch bei allgemeineren Problemen der Ablaufplanung für den Fall angewendet werden, daß eine Einzelmaschine ein Engpaß des Problems ist. Die ersten experimentellen Ergebnisse zeigten, daß die von uns vorgeschlagenen Methoden gute Ergebnisse liefern. Wegen den Voraussetzungen an die Implementation war aber nicht zu erwarten, daß unsere Ergebnisse besser als die in [CL94] dargestellten sind, da dort eine spezielle Implementation für die effiziente Verwaltung von Taskintervallen integriert wurde.

Im nächsten Abschnitt werden Probleme der Ablaufplanung kurz charakterisiert und die grundlegende Methode der Modellierung von $n \times m$ Maschinenbelegungsproblemen mit CLP(FD) angegeben. Redundante Constraints, die im Abschnitt 3 diskutiert werden, sind zusätzliche Constraints, die logisch aus den Constraints der Problembeschreibung folgen. Solche Constraints werden hinzugefügt, um die Propagationseigenschaften zu verbessern und den Suchraum einzuschränken. Zu den angegebenen redundanten Constraints gehören sowohl Constraints, die prinzipiell bereits in anderen Arbeiten beschrieben wurden, als auch Constraints, die wir neu vorschlagen. Im Abschnitt 4 werden Heuristiken diskutiert, die für die Lösungssuche erforderlich sind. Einige Schlußfolgerungen werden im letzten Abschnitt angegeben.

2 Ablaufplanung mit CLP(FD)

Die Ablaufplanung ist die zeitliche und ressourcenbezogene Planung der Ausführung einer Menge von Aufträgen. Probleme der Ablaufplanung können sehr unterschiedlich sein. Eine Problemklasse der Ablaufplanung kann wie folgt definiert werden. Auf k Maschinen werden m Aufträge verteilt, die jeweils aus n linear abhängigen Arbeitsaufgaben (bzw. Arbeitsschritten) für verschiedene Maschinen bestehen, wobei die folgenden Bedingungen gelten: zu jedem Zeitpunkt kann eine Maschine nur eine Arbeitsaufgabe ausführen, die Arbeitsaufgaben eines Auftrages sind in der gegebenen Reihenfolge ohne Unterbrechung auszuführen (die nachfolgende Arbeitsaufgabe kann erst beginnen, wenn die vorausfolgende Arbeitsaufgabe abgeschlossen ist) und die Aufträge sind untereinander

unabhängig. Ein solches Problem wird als $n \times m$ Maschinenbelegungsproblem bezeichnet, wenn zusätzlich die Arbeitsaufgaben eines Auftrages jeweils verschiedene Maschinen benötigen. Das Ziel der Optimierung besteht darin, das kürzeste Zeitintervall zu finden, in der die gegebenen Aufträge erfüllt werden können. Für die folgenden Untersuchungen ist die Voraussetzung, daß die Arbeitsaufgaben eines Auftrages linear geordnet sind, nicht erforderlich. Beispielsweise könnten die Bedingungen der Reihenfolge auch von der Art sein, wie sie in dem bekannten "Bridge"-Problem vorkommen ([Hen89]).
Modellierungen von Problemen der Ablaufplanung in CLP(FD) und Strategien zur Lösung solcher Probleme werden beispielsweise in [Hen89, DSvH90, BL94b, BL94a, CL94] diskutiert. Durch Constraints über Startzeitvariablen der Arbeitsaufgaben können insbesondere $n \times m$ Maschinenbelegungsprobleme leicht formuliert werden, wobei die Domänen dieser Variablen endliche Mengen natürlicher Zahlen sind. Constraintlogische Programmiersprachen sind aber auch für die Formulierung anderer Probleme der Ablaufplanung gut geeignet.

Für die Modellierung eines $n \times m$ Maschinenbelegungsproblems setzen wir voraus, daß eine Menge von Aufträgen und ein Zeitintervall, in dem diese Aufträge einzuplanen sind, gegeben sind. Wir nehmen dabei an, daß das gegebene Zeitintervall mit dem Zeitpunkt 0 beginnt. Die Startzeiten der Arbeitsaufgaben werden als Domänenvariablen betrachtet und die Constraints werden über diese Variablen formuliert. Die Bedingungen der Reihenfolge der Arbeitsaufgaben eines Auftrages und die Bedingung der Erfüllung eines Auftrages innerhalb des gegebenen Zeitintervalls können durch Ungleichungen augedrückt werden.
Für eine Arbeitsaufgabe t sei $start(t)$ die Startzeitvariable und $d(t)$ die gegebene Bearbeitungsdauer, die für die Erfüllung dieser Aufgabe erforderlich ist. Für zwei Arbeitsaufgaben t_1 und t_2, die auf einer Maschine zu bearbeiten sind, muß die folgende Bedingung gelten:

$$\texttt{entweder} \quad start(t_1) + d(t_1) \leq start(t_2) \quad \texttt{oder} \quad start(t_2) + d(t_2) \leq start(t_1)$$

Diese exklusive Oder-Beziehung kann durch Hinzufügen einer Booleschen Variable als Konjunktion dargestellt werden (siehe beispielsweise [AC91] oder [BL94b]). Für diese Art der Modellierung werden wir die Bezeichnung *XORBool* verwenden.

Die Bedingung, daß eine Maschine zu jedem Zeitpunkt nur für eine Arbeitsaufgabe benutzt werden kann, kann auch durch globale Constraints ausgedrückt werden. In CHIP kann diese Bedingung durch das globale Constraint *"cumulative"* oder durch das globale Constraint *"diffn"* ausgedrückt werden (siehe [BC94]). Durch das Constraint *cumulative* kann die Beschränkung der gleichzeitigen Ausführung von Arbeitsaufgaben durch eine vorhandene maximale Ressourcenverfügbarkeit repräsentiert werden. Die Bedingung der überlappungsfreien Plazierung einer Menge von n-dimensionalen Rechtecken innerhalb eines n-dimensionalen Rechtecks kann mittels des globalen Constraints *diffn* formuliert werden. Wenn diese globale Constraints in CHIP genutzt werden, sind die Implementationen und damit die Propagationseigenschaften eine "Blackbox". Entsprechend unserer Erfahrungen sind bei diesen Problemen die Propagationseigenschaften von *diffn* besser als die von *cumulative* .

3 Redundante Constraints

Im folgenden sei M eine Menge von Arbeitsaufgaben, die auf einer gegebenen Maschine einzuplanen sind. Für eine solche Menge M werden die folgenden Bezeichnungen eingeführt:

minstart(M) — frühester Startzeitpunkt der Arbeitsaufgabe aus M, die vor allen anderen Aufgaben aus M begonnen wird;

maxstart(M) — spätester Startzeitpunkt der Arbeitsaufgabe aus M, die vor allen anderen Aufgaben aus M begonnen wird;

minend(M) — frühester Fertigstellungszeitpunkt der Arbeitsaufgabe aus M, die nach allen anderen Aufgaben aus M beendet wird;

maxend(M) — spätester Fertigstellungszeitpunkt der Arbeitsaufgabe aus M, die nach allen anderen Aufgaben aus M beendet wird;

start(M) — Startzeit (Domänenvariable) der Arbeitsaufgabe aus M, die vor allen anderen Aufgaben aus M begonnen wird;

end(M) — Fertigstellungszeit (Domänenvariable) der Arbeitsaufgabe aus M, die nach allen anderen Aufgaben aus M beendet wird;

d(M) — die Summe der Dauer der Arbeitsaufgaben, die zu M gehören.

Für eine Arbeitsaufgabe werden die analogen Bezeichnungen verwendet. Für *minstart(M)* und *maxend(M)* gelten offenbar die Beziehungen:

$$minstart(M) = minimum\_of \{ minstart(t) : \ t \in M \}$$
$$maxend(M) = maximum\_of \{ maxend(t) : \ t \in M \} .$$

Für den spätesten Zeitpunkt des Beginns der Arbeitsaufgaben, die zu M gehören, gilt:

$$maxstart(M) \leq maxend(M) - d(M)$$

Die Zahl *maxstart(M)* kann auch echt kleiner als die angegebene Differenz sein. Berechnungen und Beispiele dazu sind in [Col96] enthalten. Wenn die Berechnung von *maxstart(M)* einen Wert ergibt, der kleiner als *minstart(M)* ist, können die Arbeitsaufgaben nicht eingeplant werden. Analoge Beziehungen gelten für *minend(M)* und *maxend(M)*. Um die Arbeitsaufgaben aus M erfolgreich einplanen zu können, müssen also zuerst die folgenden Bedingungen erfüllt sein:

$$\boxed{minstart(M) \leq maxstart(M) \quad \text{und} \quad minend(M) \leq maxend(M) \qquad (1)}$$

Sei $d_1(M)$ die Summe der Dauer der Arbeitsaufgaben, die nicht zu M gehören und zwischen den Zeitpunkten *maxstart(M)* und *minend(M)* auf der gleichen Maschine (wie die Aufgaben aus M) auszuführen sind. Die Startzeit *start(M)* der ersten Arbeitsaufgabe von M kann als Domänenvariable mit Werten zwischen *minstart(M)* und *maxstart(M)* definiert werden. Entsprechend kann die Fertigstellungszeit *end(M)* aller Arbeitsaufgaben von M als eine Domänenvariable mit Werten zwischen *minend(M)* und *maxend(M)* eingeführt werden. Um die Arbeitsaufgaben aus M erfolgreich einplanen zu können, müssen die folgenden Constraints gelten:

$$\mathit{start(M)} = \mathit{minimum\_of}\,\{\, \mathit{start(t)} : t \in M\} \tag{2}$$
$$\mathit{end(M)} = \mathit{maximum\_of}\,\{\, \mathit{end(t)} : t \in M\} \tag{3}$$
$$\mathit{start(M)} + \mathit{d(M)} + d_1(M) \leq \mathit{end(M)} \tag{4}$$

Die Teilmenge *first(M)* von M wird durch die Arbeitsaufgaben $t \in M$ definiert, die folgende Bedingungen erfüllen:

$$\mathit{minstart(t)} \leq \mathit{maxstart(M)}$$
$$\mathit{minstart(t)} + \mathit{d(M)} + d_1(M) \leq \mathit{maxend}(\,M \setminus \{t\}\,)$$

Eine Arbeitsaufgabe aus M kann nur dann als erste Arbeitsaufgabe dieser Menge ausgeführt werden, wenn sie zu *first(M)* gehört. Analog wird die Menge *last(M)* der Arbeitaufgaben $t \in M$, die als letzte Arbeitsaufgabe der zu M gehörenden Aufgaben ausgeführt werden können, durch die beiden folgenden Bedingungen definiert:

$$\mathit{minend(M)} \leq \mathit{maxend(t)}$$
$$\mathit{minstart}(\,M \setminus \{t\}\,) + \mathit{d(M)} + d_1(M) \leq \mathit{maxend(t)}$$

Wenn die Menge *first(M)* oder die Menge *last(M)* kein Element enthält, kann offenbar keine Lösung existieren. Wenn eine dieser beiden Menge einelementig ist, muß die entsprechende Arbeitsaufgabe vor bzw. nach allen anderen Arbeitsaufgaben aus M ausgeführt werden.

$$\mathrm{card}(\mathit{first(M)}) > 0 \qquad \text{und} \qquad \mathrm{card}(\mathit{last(M)}) > 0 \tag{5}$$

Zusätzlich zu den bisher genannten Bedingungen und Constraints, die prinzipiell auch in verschiedenen Arbeiten beschrieben wurden, schlagen wir vor, auch Constraints hinzuzufügen, die ausdrücken, daß genau eine Arbeitsaufgabe aus *first(M)* als erste Aufgabe und genau eine Arbeitsaufgabe aus *last(M)* als letzte Aufgabe bezüglich M einzuplanen sind. Im folgenden werden nur diese Constraints für *first(M)* angegeben. Die entsprechenden Constraints für die Menge *last(M)* sind analog zu formulieren.
Sei *first(M)* = $\{\, t_1, \ldots, t_m \,\}$ und seien $B_1, \ldots, B_m$ Boolesche Variablen, d.h., die Domänen dieser Variablen enthalten nur die Zahlen 0 und 1. Außerdem sei *big* eine hinreichend große Zahl (z.B. der späteste Fertigstellungstermin aller Arbeitsaufgaben). Für jedes $i \in \{1, \ldots, m\}$ und jedes $j \in \{1, \ldots, m\} \setminus \{i\}$ muß das folgende Constraint erfüllt sein:

$$\mathit{start}(t_i) + \mathit{d}(t_i) \leq \mathit{start}(t_j) + \mathit{big} * B_i \tag{6}$$

Außerdem muß das Constraint

$$\boxed{B_1 + \ldots + B_m = m - 1 \qquad (7)}$$

gelten. Somit muß genau für ein $i_0 \in \{ 1, \ldots n \}$ $B_{i_0} = 0$ sein und nur für dieses i_0 sind die Constraints aus (6) relevant. Um für diesen Fall auszudrücken, daß die ausgewählte Arbeitsaufgabe auch vor den anderen zu M gehörenden Arbeitsaufgaben auszuführen ist, wird für jedes $i \in \{1, \ldots, m\}$ ein bedingtes Constraint erzeugt:

`if` $B_i = 0$	
`then` für jedes $t \in M \setminus \mathit{first(M)}$: $\mathit{start}(t_i) + d(t_i) \leq \mathit{start}(t)$	(8)

Die angegebenen Bedingungen und Constraints können in CHIP leicht formuliert werden. Für die Constraints (2) und (3) sind das `minimum`-Constraint und das `maximum`-Constraint von CHIP zu verwenden. Die Constraints der Form (8) können mit dem bedingten Constraint ausgedrückt werden. Alle anderen Constraints (bzw. Bedingungen) sind Ungleichungen oder Gleichungen.

4 Heuristiken für die Suche

Auch wenn redundante Constraints erzeugt werden, können i. allg. die Variablendomänen nicht so eingeschränkt werden, daß sich eine Lösung ergibt. Folglich ist eine Lösungssuche erforderlich, bei der für jede Maschine die Festlegung der Bearbeitungsreihenfolge der Arbeitsaufgaben entscheidend ist. Somit werden Heuristiken für die schrittweise Erzeugung dieser Reihenfolgen benötigt. Durch die Verwendung von redundanten Constraints kann der Suchraum eingeschränkt werden. Die Constraints aus dem vorangegangenen Abschnitt gelten für jede Teilmenge von Arbeitsaufgaben, die auf einer Maschine zu erfüllen sind. Das Erzeugen der redundanten Constraints für alle diese Mengen in jedem Suchschritt ist aber nicht effizient. Also werden Heuristiken benötigt, die bestimmen, für welche Aufgabenmengen welche redundante Constraints erzeugt werden.

Zuerst ist also festzulegen, für welche Aufgabenmengen redundante Constraints erzeugt werden. In [CL94] wird die Betrachtung von Taskintervallen vorgeschlagen. Bei Verwendung von Taskintervallen ist eine effiziente Problemlösung nur dann möglich, wenn die dynamischen Änderungen der Taskintervalle während der Suche effizient implementiert werden können (siehe [CL94]). Eine solche effiziente Implementation wird aber von CHIP nicht unterstützt.
Als Ausgangspunkt unserer Heuristik wählten wir einen Ansatz, der bereits in [CP89] und [Col96] verwendet wurde. Für jede Maschine wird eine Menge *NotOrd* betrachtet, die aus den Arbeitsaufgaben der Maschine besteht, für die die Bearbeitungsreihenfolge noch nicht vollständig bestimmt ist. Eine Arbeitsaufgabe gehört genau dann zu *NotOrd*, wenn eine weitere Arbeitsaufgabe existiert, die auf der gleichen Maschine einzuplanen ist, und die Reihenfolge der Bearbeitung zwischen diesen beiden Arbeitsaufgaben noch

nicht feststeht. Offensichtlich kann eine solche Menge nicht einelementig sein. Während der Suche ändert sich natürlich die Menge *NotOrd*. Die in [Col96] angegebene Verwaltung der Ordnungsbeziehungen zwischen den Arbeitsaufgaben ermöglicht eine effiziente Berechnung dieser Menge für jede Maschine. Diese Verwaltung der Ordnungsbeziehungen wird aber durch CHIP bisher nicht direkt unterstützt. In CHIP könnte eine Verwaltung der relevanten Ordnungsbeziehungen mittels der Booleschen Variablen organisiert werden, die entstehen, wenn die exklusiven Oder-Beziehungen durch *XORBool* modelliert werden (siehe Abschnitt 2). Da wir diese Modellierung aber nicht zwingend verwenden wollten, verzichteten wir auf eine explizite Verwaltung der relevanten Ordnungsbeziehungen.
Nach jedem Suchschritt muß also *NotOrd* für jede Maschine erstmal wieder neu berechnet werden. Da eine Arbeitsaufgabe, die nicht mehr zu *NotOrd* gehört, auch in den folgenden Suchschritten nicht zu dieser Menge gehören kann, kann sie markiert werden. Im Falle des Backtrackings muß eine solche Markierung wieder aufgehoben werden.
Für den Test, ob für zwei Arbeitsaufgaben t_1 und t_2 die Reihenfolge bereits festgelegt ist, wählten wir nur eine hinreichende Bedingung:

$$minend(t_1) < maxstart(t_2) \quad \text{oder} \quad minend(t_2) < maxstart(t_1)$$

Eine exaktere, aber zeitintensivere Methode für den Test besteht darin, das Absetzen der Ungleichung $start(t_1) + d(t_1) \leq start(t_2)$ und der analogen Ungleichung schrittweise zu "probieren".

Um die Anzahl der Überprüfungen etwas zu verringern und den Suchraum einzuschränken, verwendeten wir erfolgreich die folgende Heuristik: für Aufgaben mit kurzer Bearbeitungsdauer wird keine Ordnung direkt festgelegt. Diese Arbeitsaufgaben werden nicht bei der Bestimmung von *NotOrd* berücksichtigt (aber bei der Berechnung von $d_1(NotOrd)$, siehe Abschnitt 3). Durch die Wirkung der grundlegenden Constraints (siehe Abschnitt 2) finden diese Arbeitsaufgaben ihren "Platz" im Plan. Diese Heuristik sollte aber nur für wenige Aufgaben angewendet werden, wobei eine Konzentration der Zuordnung zu einer Maschine bzw. eines Auftrages vermieden werden sollte.

Die redundanten Constraints aus Abschnitt 3 werden nicht für jede berechnete Menge *NotOrd* erzeugt, um den Aufwand bei der Suche zu reduzieren. Sie werden nur dann erzeugt, wenn sich die Anzahl der Elemente von *NotOrd* gegenüber der Berechnung vor dem entsprechenden Suchschritt verringert hat und größer als 2 ist.

Nach Heuristiken für das Erzeugen redundanter Constraints werden nun Heuristiken für die schrittweise Erzeugung der Bearbeitungsreihenfolge der Arbeitsaufgaben auf jeder Maschine diskutiert. In unseren ersten Experimenten wählten wir die folgenden Heuristiken:

1. Die Bearbeitungsreihenfolge wird maschinenweise vollständig festgelegt. Die Reihenfolge der Maschine wird durch eine Engpaßanalyse bestimmt. Grundlage der Engpaßanalyse sind die Anforderungsprofile der Maschinen (siehe [SF96]).

2. Wenn für die ausgewählte Maschine die entsprechende Menge *NotOrd* nur zwei Elemente enthält, wird eine der beiden möglichen Reihenfolgen gewählt. Im anderen Fall wird entweder *first(NotOrd)* oder *last(NotOrd)* betrachtet, wobei *last(NotOrd)* nur dann ausgewählt wird, wenn diese Menge weniger Elemente enthält als *first(NotOrd)*.

3. Wenn die Menge *first(NotOrd)* (bzw. *last(NotOrd)*) ausgewählt wurde, wird eine der Booleschen Variablen aus den zugeordneten Constraints (6) und (7) mit 0 identifiziert. Die Reihenfolge der Auswahl dieser Booleschen Variable wird bisher einfach aus dem Minimum der Domänen der zugehörigen Startzeitvariablen bestimmt.

Gegenüber den vorgeschlagenen Heuristiken werden in [CP89], [CL94] und [Col96] jeweils nur eine Ordnung zwischen zwei ausgewählten Arbeitsaufgaben festgelegt. Diese Aufgaben werden analog aus den entsprechenden Mengen *"first"* oder *"last"* ausgewählt, wobei für diese Auswahl verschiedene Heuristiken verwendet werden. Die Engpaßanalyse erfolgt in allen drei Arbeiten unterschiedlich und auch anders als in unserem Vorschlag. Außerdem sind in [CL94] Taskintervalle die Grundlage der Betrachtungen.

Bisher führten wir keine systematischen Untersuchungen zu Verbesserungen unseres Ansatzes durch. Die ersten Experimente zeigten, daß unsere Heuristiken erfolgreich verwendet werden können. Mit CHIP und den beschriebenen Heuristiken konnte eine optimale Lösung des bekannten 10×10 Maschinenbelegungsproblems aus [MT63] nach 25 Backtrackingschritten gefunden werden. Für den Beweis der Optimalität dieser Lösung wurden 2583 Backtrackingschritte benötigt. Auf einem 150 MHz-Pentium-Rechner unter Window-NT wurden für das Erzeugen einer optimalen Lösung 7 Sekunden und für den Nachweis der Optimalität 7 Minuten benötigt. Bessere Laufzeiten sind zu erwarten, wenn für die Berechnung der Mengen *NotOrd* ein effizienteres Verfahren gewählt werden könnte. Außerdem wurde für die schrittweise Festlegung der Bearbeitungsreihenfolgen auf den Maschinen bisher relativ einfache Heuristiken gewählt. In der folgenden Tabelle werden Ergebnisse aus der Literatur für den Beweis der Optimalität dieses 10×10 Maschinenbelegungsproblems dargestellt. Dabei bezeichnet BT die Anzahl der erforderlichen Backtrackingschritte und CPU die Laufzeit in Sekunden.

	[CP89] PRIME 2655	[AC91] Sparc 1	[CL94] Sparc 1	[BPN95] RS 6000	[Col96] Sparc 5	[Wür97] Sparc 20	– *hier* – Pentium 150
BT	17 982	16 055	6 728	4 181	10 991	2 564	2 583
CPU	14 680	372	730	59	116	81	420

Die Anzahl der erforderlichen Backtrackingschritte für die Erzeugung einer Lösung und damit die Laufzeit hängt auch wesentlich von der gewählten Problemmodellierung ab. Für die oben genannte Erzeugung einer optimalen Lösung waren bei verschiedenen Modellierungen (siehe Abschnitt 2) und zwei verschiedenen Heuristiken die folgenden Backtrackingschritte notwendig:

cumulative + diffn	*diffn*	*cumulative*	*XORBool*
399	429	10 584	22 488
25	25	467	5 665

Untersuchungen zur Verbesserung der vorgeschlagenen Heuristiken sind geplant. Das grundsätzliche Vorgehen soll dabei erhalten bleiben. Zunächst werden auch bestimmte Teilmengen von *NotOrd* in die Betrachtungen einbezogen. Insbesondere soll dies der Fall sein, wenn diese Menge in disjunkte Teilmengen zerlegt werden kann. Wenn beispielweise für eine Maschine eine Arbeitsaufgabe bereits vollständig eingeplant ist (d.h., die Ordnungsbeziehung zu den anderen Aufgaben dieser Maschine ist bestimmt) und diese Aufgabe zeitlich innerhalb der entsprechenden Menge *NotOrd* auszuführen ist, dann kann *NotOrd* in zwei disjunkten Teilmengen so zerlegt werden, daß jede Aufgabe der einen Menge immer vor jeder Aufgabe der anderen Menge auszuführen ist. Weiterhin sollen verschiedene Varianten der Heuristiken zur Festlegung der Bearbeitungsreihenfolgen untersucht werden. Dies betrifft insbesondere Heuristiken für die Reihenfolge der Auswahl der Booleschen Variable, die mit 0 bewertet wird.

5 Schlußfolgerungen

Ein Verfahren wurde vorgestellt, mit dem optimale Lösungen von $n \times m$ Maschinenbelegungsproblemen in relativ kurzer Zeit mit der constraintlogischen Programmiersprache CHIP erzeugt werden können. Dabei wurde CHIP als "Blackbox" betrachtet und für Suchraumeinschränkungen wurden nur die Propagationseigenschaften der Constraints zwischen den Startzeitvariablen (endliche Domänenvariablen) genutzt. Durch systematische Untersuchungen von verschiedenen Varianten der Heuristiken sind noch Verbesserungen zu erwarten. Die vorgeschlagenen Methoden können in einem allgemeineren Planungstool, das in CHIP implementiert ist, integriert werden. In den Fällen, in denen diese Methoden anwendbar sind, können bessere Lösungen erzeugt werden als mit den allgemeineren Suchmethoden. Um die Einsatzmöglichkeiten dieser Methoden in dem allgemeineren Planungstool noch zu verbessen, sind weitere Untersuchungen geplant. Insbesondere sollen die Propagationseigenschaften der vorgeschlagenen redundanten Constraints mit dem Aufwand ihrer Erzeugung vergliechen werden. Durch solche Untersuchungen erwarten wir bessere Heuristiken für Bedingungen, wann welche redundante Constraints während der Suche erzeugt werden sollten. Das Ergebnis der Untersuchungen könnte auch eine abgeschwächte Variante sein, die zwar etwas schlechtere Propagationseigenschaften besitzt, aber wesentlich schneller einen Suchschritt überprüft. Außerdem sind Untersuchungen zur Kombination der hier diskutierten Methoden mit der Methode der Domänenreduzierung ([Gol95]) vorgesehen.

Literatur

[AB93] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *J. Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[AC91] D. Applegate and W. Cook. A computational study of job-shop scheduling. *Operations Research Society of America*, 3(2), 1991.

[BC93] F. Benhamou and A. Colmerauer, editors. *Constraint Logic Programming (Selected Research)*. MIT Press, Cambridge, London, 1993.

[BC94] E. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *J. Mathematical and Computer Modelling*, 20(12):97–123, 1994.

[BL94a] S. Breitinger and H.C.R. Lock. Improving search for job-shop-scheduling with CLP(FD). In M. Hermenegildo and J. Penjam, editors, *Programming Language Implementation and Logic Programming*, volume 844 of *Lecture Notes in Computer Science*, pages 277–291, Berlin, Heidelberg, 1994. Springer.

[BL94b] S. Breitinger and H.C.R. Lock. Modelling and scheduling in CLP(FD). In L. Sterling, editor, *Proc. Internat. Conf. on the Practical Application of Prolog*, pages 95–110. London, 1994.

[BPN95] P. Baptiste, C. Le Pape, and W. Nuijten. Constraint-based optimization and approximation for job-shop scheduling. In *Proc. AAAI-Sigman Workshop on Intelligent Manufacturing Systems, IJCAI-95, Montreal*, 1995.

[CL94] Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In P. Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming*, pages 369–383. MIT Press, 1994.

[Col96] Y. Colombani. Constraint programming: an efficient and practical approach to solving the job-shop problem. In E.C. Freuder, editor, *Principles and Practice of Constraint Programming – CP'96*, volume 1118 of *Lecture Notes in Computer Science*, pages 149–163, Berlin, Heidelberg, 1996. Springer-Verlag.

[CP89] J. Carlier and E. Pinson. An algorithm for solving the job shop problem. *Management Science*, 35(2):164–176, 1989.

[DSvH90] M. Dincbas, H. Simonis, and P. van Hentenryck. Solving large combinatorial problems in logic programming. *J. Logic Programming*, 8:75–93, 1990.

[Gol95] H.-J. Goltz. Reducing domains for search in CLP(FD) and its application to job-shop scheduling. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming – CP'95*, volume 976 of *Lecture Notes in Computer Science*, pages 549–562, Berlin, Heidelberg, 1995. Springer-Verlag.

[Hen89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge (Mass.), London, 1989.

[JM94] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *J. Logic Programming*, 19/20:503–581, 1994.

[MT63] J.F. Muth and G.L. Thompson. *Industrial Scheduling*. Prentice-Hall, Englewood Cliffs, NJ, 1963.

[SF96] N. Sadeh and M. Fox. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86:1–41, 1996.

[Wür97] J. Würtz. Constraint-based scheduling in Oz. In U. Zimmermann, U. Derigs, W. Gaul, R. Möhrig, and K.-P. Schuster, editors, *Operations Research Proceedings 1996*, pages 218–223. Springer-Verlag, Berlin, Heidelberg, New York, 1997. Selected Papers of the Symposium on Operations Research (SOR 96), Braunschweig, Germany, September 3–6, 1996.