Ausarbeitung zum Thema

„On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimization“

Diese Ausarbeitung wurde im Rahmen des Seminars
„Optimierung industrieller Anwendungsprobleme“
und des Studienprojekts „Fahrplanoptimierung“ erstellt.

Autor: Bodo Schenk

Diese Arbeit basiert auf dem gleichnamigen Papier vom Hooker J., Ottosson G., Thorsteinsson E. und Kim H., 1999.

Inhaltsverzeichnis

1. Einleitung *3*
1.1 Wozu werden LP und CP verwendet ? 3
1.2 Prinzipielle Funktionsweise des LP-Verfahrens 4
1.3 Prinzipielle Funktionsweise des CP-Verfahrens 6

2. Das Maschinen-Zuordnungs-Problem (MZP) *7*
2.1 MZP anschaulich 7
2.2 MZP formal 8
2.3 MZP als LP-Problem 9
2.4 MZP als CP-Problem 10
2.5 Abschließender Vergleich von LP und CP 11

3. Mixed Logical/Linear Programming *12*
3.1 Vorstellung des Modells 12
3.2 MLLP formal 12
3.3 Der Lösungs-Algorithmus 13
3.4 MZP als MLLP-Problem 15

4. Constraint Stores *16*
4.1 Grundüberlegungen 16
4.2 Constraint Stores in MLLP 16
4.3 Zugriff auf den CP-Store 17

1. Einleitung

Für die Optimierung industrieller Anwendungsprobleme - wie zum Beispiel Arbeitszuordnungsprobleme oder den „Traveling Salesman“ - wurden in der Vergangenheit hauptsächlich zwei Verfahren eingesetzt: *Linear Programming* (LP) und *Constraint Propagation* (CP).

Es gab auch einige Ansätze, diese Verfahren miteinander zu verbinden, um die Vorteile beider Techniken nutzen zu können. Diese Ansätze beschränkten sich aber im Allgemeinen darauf, entweder beide Verfahren parallel ablaufen zu lassen, oder das eine in das andere einzubauen.

Diese Arbeit beschäftigt sich mit dem Versuch, ein völlig neues Modell zu erstellen. Dieses Modell soll sowohl die verschiedenen Aspekte von LP wie auch die von CP berücksichtigen.

1.1 Wozu werden LP und CP verwendet ?

Beide Verfahren haben ein Ziel: es soll eine möglichst optimale Lösung für ein komplexes Problem gefunden werden, für das es keine vorgefertigten analytischen Lösungsalgorithmen gibt, oder für das diese Algorithmen nicht in polynomieller Zeit abgearbeitet werden können (NP-Probleme). Dabei wird bewusst in Kauf genommen, dass nicht die beste Lösung gefunden wird, sondern nur eine möglichst gute Lösung.

Bei solchen Problemen handelt es sich oftmals um Zuordnungsprobleme: eine bestimmte Menge von Aufgaben soll einer bestimmten Menge von Leuten/Maschinen zugeordnet werden, oder in eine bestimmte Reihenfolge gebracht werden, oder beides. Dazu zählen beispielsweise sämtliche Varianten des „Traveling Salesman Problem“.

Zur Lösung eines solchen Problems sind immer mindestens die folgenden Angaben nötig:

- Die beteiligten Variablen
- Die zu minimierende (oder zu maximierende) Funktion, die von einigen dieser Variablen abhängt („Bewertungsfunktion“)
- Die Bedingungen, denen diese Variablen genügen müssen (z.B. untere und obere Grenzen). Diese Bedingungen sehen je nach Lösungsverfahren (LP oder CP) anders aus.

1.2 Prinzipielle Funktionsweise des LP-Verfahrens

Bei dem LP-Verfahren müssen die Bedingungen für die Variablen in Form von linearen Gleichungen gegeben werden. Die Angabe von komplexeren Bedingungen - beispielsweise eine Art *If...Then...Else* - ist nicht direkt möglich.

Ein LP-Problem wird allgemein so formuliert:

$$\begin{aligned} &\text{min. } cx \\ &\text{subject to } Ax = b \\ &x \geq 0 \end{aligned}$$

wobei $x = (x_1, ..., x_n)$
$c = (c_1, ..., c_n)$
$b = (b_1, ..., b_m)$
A ist eine Matrix mit n Spalten und m Zeilen

Ein Beispiel wäre:

$$\begin{aligned} \text{min.} \quad & x_1 + 2x_2 + x_3 \\ \text{s.t.} \quad & x_1 + 2x_2 - x_3 = 10 \\ & 2x_1 + 2x_2 + 2x_3 = 20 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Es gibt mehrere Verfahren, solche Probleme zu lösen. Das bekannteste davon ist der Simplex-Algorithmus. Dieser Algorithmus geht davon aus, dass die linearen Gleichungen im Suchraum einen Körper mit Ecken und Kanten bilden. Die optimale Lösung muß auf einer der Ecken liegen. Also versucht der Algorithmus, ausgehend von einer Ecke, entlang der Kanten zu den anderen Ecken zu kommen.

Dazu wird ein sog. *Tableau* benutzt. In der ersten Zeile steht die zu minimierende Funktion mit dem Initialergebnis 0. In den folgenden Zeilen stehen die einzelnen Bedingungen.

Initialisierung

	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$
0	1	2	1
10	1	2	-1
20	2	2	2

Nun muss eine erste Lösung, also eine Ecke des Körpers im Suchraum, gefunden werden. Dazu wird eine Basis aus linear unabhängigen Spalten gebildet. In diesem Beispiel ist dies offensichtlich mit den Spalten x_1 und x_2 möglich. Die Spalte x_3 hingegen ist linear abhängig von dieser Basis.

Die Variablen, die die Basis bilden, bekommen die Werte in der linken Ergebnisspalte zugewiesen. Die Variablen, die nicht in der Basis sind, bekommen den Wert 0 zugewiesen.

Durch elementare lineare Umformungen kommt man zur ersten Lösung:

Lösung 1

Basis(x_1, x_2)	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$
10	0	0	-2
10	1	0	3
0	0	1	-2

Ergebnis: $x_1 = 10$, $x_2 = 0$, $x_3 = 0$, min. 10

Nun wird versucht, eine Spalte in die Basis zu übernehmen, und eine andere dafür zu entfernen. Durch elementare lineare Umformungen wird x_2 durch x_3 ersetzt.

Lösung 2

Basis(x_1, x_3)	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$
10	0	- 1	0
10	1	3/2	0
0	0	- 1/2	1

Ergebnis: $x_1 = 10$, $x_2 = 0$, $x_3 = 0$, min. 10

Lösung 3

Basis(x_2, x_3)	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$
50/3	2/3	0	0
20/3	2/3	1	0
10/3	1/3	0	1

Ergebnis: $x_1 = 0$, $x_2 = 20/3$, $x_3 = 10/3$, min. $50/3 \approx 16{,}6$

Wie lange dieser Algorithmus abläuft, welche Probleme es z.B. mit Endlosschleifen gibt, und wie man herausfindet, welche Variable in die Basis kommt und welche dafür entfernt wird, ist nicht Thema dieser Ausarbeitung und kann in entsprechender Literatur über lineare Programmierung nachgelesen werden.

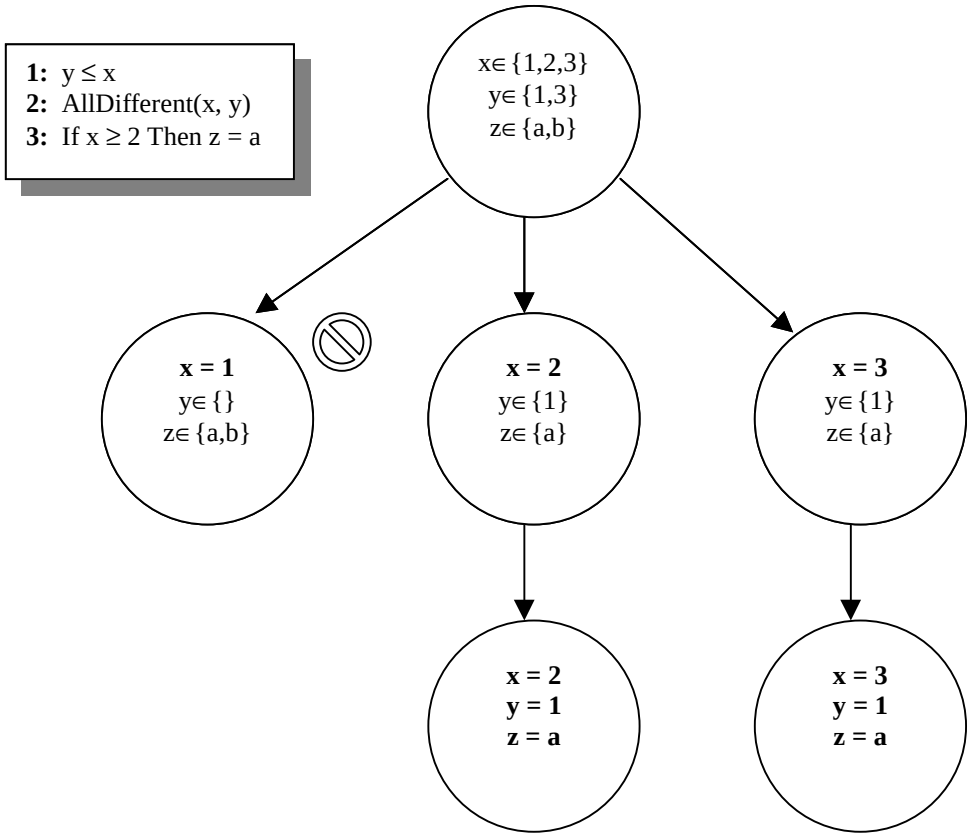
Hinweis: dieser Algorithmus funktioniert nur mit reellen Zahlen. Beim ILP (Integer Linear Programming) können auch diskrete (ganze) Zahlen verwendet werden. Ein ILP-Problem zu lösen ist jedoch wesentlich schwerer, als ein LP-Problem zu lösen.

1.3 Prinzipielle Funktionsweise des CP-Verfahrens

Bei CP kann man die Bedingungen genauso wie bei LP durch Gleichungen und Ungleichungen ausdrücken. Außerdem kann man logische Operatoren, komplexere Konstrukte wie *If...Then...Else* und einige vordefinierte Bedingungen (z.B. *alldifferent-Constraint*) verwenden.

Bei CP beginnt die Suche in einem Suchbaum. Am Wurzelknoten sind alle Variablen noch unbekannt. Jede Variable hat aber eine sog. *Domain*, eine (endliche) Menge von gültigen Werten, die man dieser Variablen zuweisen kann.

Dann wird eine Variable ausgewählt und alle Werte, die diese Variable aufgrund der zugeordneten Domain annehmen kann, aufgezählt. Für jeden Wert wird ein neuer Knoten gebildet, wobei die Domains modifiziert werden.



Nach einer Weile werden die Domains so klein, dass sie entweder nur noch einen einzigen Wert enthalten oder leer sind. Wurden die in der Bewertungsfunktion vorkommenden Variablen auf diese Art bestimmt, kann die Lösung bewertet werden.

Bei einigen Knoten kann die Suche abgebrochen werden, da entweder eine leere Domain - also eine ungültige Lösung - entstanden ist oder die Bewertungsfunktion dieses Knotens ein schlechteres Ergebnis liefert als die beste bisher gefundene Lösung. Da sich unterhalb eines Knotens die Domains noch mehr verkleinern, könnte sich die Lösung niemals weiter verbessern.

(Hinweis: bei diesem Beispiel wurde die Frage der Optimierung weggelassen. Ziel war es, eine beliebige, gültige Lösung zu finden.)

2. Das Maschinen-Zuordnungs-Problem (MZP)

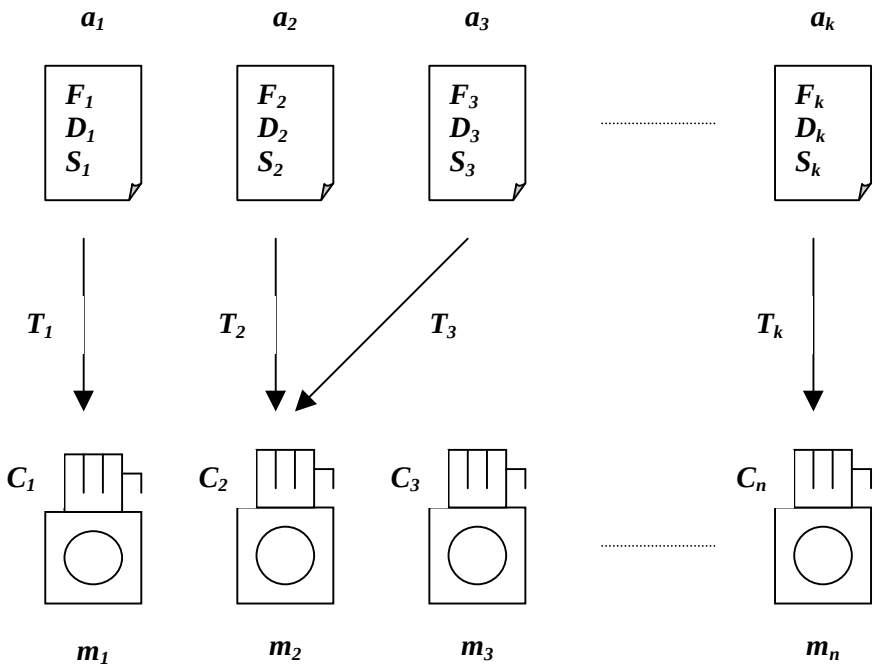
Im folgenden soll ein typisches Anwendungsproblem geschildert werden, das mit beiden Techniken gelöst werden kann. Anhand dieses Beispiels sollen die Vor- und Nachteile der verschiedenen Techniken demonstriert werden.

2.1 MZP anschaulich

In einer Fabrik arbeiten eine Reihe von Maschinen unabhängig voneinander. Immer morgens werden die für den Tag geplanten Aufträge auf diese Maschinen verteilt. Eine Maschine m läuft entweder den ganzen Tag (und verursacht damit konstante Kosten C_m), oder bleibt ausgeschaltet (und verursacht keine Kosten).

Jeder Auftrag a kann frühestens zum Zeitpunkt F_a begonnen werden, dauert D_a und muß bis spätestens zum Zeitpunkt S_a beendet sein. Der Zeitpunkt, an dem der Auftrag a dann tatsächlich ausgeführt wird, sei T_a.

Ziel ist es, die durch die Maschinen auflaufenden Kosten zu minimieren.



2.2 MZP formal

gegeben: $MZP = (A, M, D, F, S, C)$

$A = \{a_1 \dots a_k\}$	Menge von Aufträgen
$M = \{m_1 \dots m_n\}$	Menge von Maschinen
$F, D, S: A \rightarrow Z^+$	Früheste Startzeit, Dauer und späteste Beendigungszeit eines Auftrags
$C: M \rightarrow Z^+$	Kosten, die eine Maschine verursacht, sobald sie mind. einmal benutzt wird

gesucht:

$\sigma: A \rightarrow M$	Ordnet einen Auftrag einer Maschine zu
$T: A \rightarrow Z^+$	Der Zeitpunkt, an dem ein Auftrag startet

Bedingungen:

1. Ein Auftrag i kann frühestens bei F(i) gestartet werden
   $\forall i=1...k: T_i \geq F_i$

2. Ein Auftrag i muss bis spätestens S(i) beendet sein
   $\forall i=1...k: T_i + D_i \leq S_i$

3. Zwei Aufträge auf der gleichen Maschine dürfen sich nicht überschneiden
   $\forall i=1...k, j=1...k, i \neq j: (\sigma_i = \sigma_j) \rightarrow (T_i + D_i \leq T_j) \vee (T_j + D_j \leq T_i)$

Minimierung:

$$\forall m=1...n: f_m = \begin{cases} C_m, \text{ falls } \exists i=1...k: \sigma_i = m \\ 0, \text{ sonst} \end{cases}$$

$$\text{min. } B = \sum_m f_m$$

2.3 MZP als LP-Problem

Zur Lösung des Problems werden folgende Integer-Variablen definiert, die alle entweder den Wert 0 oder den Wert 1 haben können.

Wenn die beiden Aufträge i und j ($i \neq j$) auf der gleichen Maschine ausgeführt werden, und der Auftrag i vor dem Auftrag j ausgeführt wird, dann sei x_{ij}=1, ansonsten 0. Wenn der Auftrag i auf der Maschine m ausgeführt wird, dann sei y_{im}=1, ansonsten 0. Wenn die Maschine m benutzt wird, dann sei z_m=1, ansonsten 0.

Die zu minimierende Funktion ist dann offensichtlich die Summe der Kosten aller benutzten Maschinen:

$$\text{min: } \sum_{m} C_m * z_m$$

Außerdem müssen folgende Bedingungen eingehalten werden:

1. Eine Maschine wird dann benutzt, wenn ihr mindestens ein Auftrag zugeordnet ist

$$\forall m,i: z_m \geq y_{im}$$

2. Ein Auftrag wird genau einer Maschine zugeordnet

$$\forall i: \sum_{m} y_{im} = 1$$

3. Ein Auftrag kann frühestens zu einem bestimmten Zeitpunkt begonnen werden

$$\forall i: T_i \geq F_i$$

4. Ein Auftrag muss bis zu einem bestimmten Zeitpunkt beendet sein

$$\forall i: T_i + D_i \leq S_i$$

5. Zwei Aufträge auf der gleichen Maschine müssen nacheinander abgearbeitet werden

$$\forall i \neq j: T_i + D_i \leq T_j + M(1 - x_{ij})$$

6. Wenn zwei Aufträge der gleichen Maschine zugeordnet sind, kommt entweder der eine vor dem anderen oder andersherum

$$\forall m, i \neq j: x_{ij} + x_{ji} \geq y_{im} + y_{jm} - 1$$

An diesem Beispiel sind die folgenden LP-typischen Eigenschaften zu beobachten:

- Die in LP benutzten Variablen sind oft mehrfach indiziert. In diesem Beispiel haben die Variablen x und y jeweils zwei Indizes. Bei komplexeren Problemen kann die Zahl der Indizes so groß werden, dass das Programm völlig unverständlich wird.

- Verwendung eines „Big-M constraints“ in Bedingung 5. Im Fall x_{ij}=1 ist die Bedingung eindeutig. Im Fall x_{ij}=0 muß die Bedingung immer wahr sein, weil der Auftrag i nicht vor dem Auftrag j ausgeführt werden muß. Nur wenn die Konstante M groß genug ist, kann dies immer erfüllt sein.

- In LP können sehr effiziente Bedingungen angegeben werden, die die Zahl der möglichen Lösungen stark einschränken, z.B. die Bedingungen 1 und 2. Andererseits verursacht die zweifach indizierte Variable x eine ungewollte Vergrößerung des Suchraumes.

- Da Integervariablen verwendet werden, handelt es sich um ein ILP-Problem. Dieses ist bei weitem nicht so effizient lösbar wie ein reines LP-Problem.

2.4 MZP als CP-Problem

Wenn die Maschine m benutzt wird, sei die Variable z_m=1, ansonsten 0. Außerdem sei m_i die Maschine, auf der Auftrag i ausgeführt wird.

Die zu minimierende Funktion ist dann wieder:

$$\text{min: } \sum_{m} C_m * z_m$$

Es müssen folgende Bedingungen eingehalten werden:

1. Eine Maschine wird genau dann benutzt, wenn ihr mindestens ein Auftrag zugeordnet ist

$$\forall m\text{: If } atleast(m, m_1 \ldots m_n, 1) \text{ Then } z_m=1 \text{ Else } z_m=0$$

2. Ein Auftrag kann frühestens zu einem bestimmten Zeitpunkt begonnen werden

$$\forall i\text{: } T_i \geq F_i$$

3. Ein Auftrag muss bis zu einem bestimmten Zeitpunkt beendet sein

$$\forall i\text{: } T_i + D_i \leq S_i$$

4. Zwei Aufträge auf der gleichen Maschine müssen nacheinander abgearbeitet werden

$$\forall i,j\text{: If } m_i = m_j \text{ Then } (T_i + D_i \leq T_j) \vee (T_j + D_j \leq T_i)$$

An diesem Beispiel sind die folgenden typischen CP-Eigenschaften zu beobachten:

- Alle Bedingungen können kurz, übersichtlich und leicht verständlich angegeben werden. Mehrfach indizierte Variablen sind in der Regel nicht nötig.

- Durch die komplexeren Ausdrücke wird die Auswertung der Bedingungen wahrscheinlich weniger effizient ablaufen als bei einem LP-Problem (aber wahrscheinlich schneller als bei einem ILP-Problem).

2.5 Abschließender Vergleich von LP und CP

Die folgenden Punkte sollen die in den ersten Kapiteln dargestellten Sachverhältnisse zusammenfassen:

- LP und CP sind unterschiedliche Verfahrensweisen, um optimale Lösungen für komplexe Probleme zu finden.

- Beide Verfahren versuchen, eine Menge von Variablen derart zu belegen, dass eine gegebene Funktion minimal (oder maximal) wird, und die Variablenbelegungen innerhalb gewisser Grenzen bleiben.

- Die Art, in der diese Grenzen in Form von Bedingungen angegeben werden, ist bei LP und CP unterschiedlich.

- LP erlaubt es, sehr effizient zu bearbeitende Bedingungen in Form linearer Gleichungen zu stellen. Allerdings ist das auch die einzige Möglichkeit, Bedingungen zu definieren, was die Freiheit des Benutzers stark einschränkt und die Ausdrücke sehr unübersichtlich macht.

- In CP können darüber hinaus komplexere Bedingungen gestellt werden, die aber nicht so effizient und zielsicher abgearbeitet werden können.

In den folgenden Kapiteln wird versucht, die Vorteile beider Techniken zu nutzen. Kapitel 3 verfolgt die Idee, ein völlig neues Modell zu definieren, in dem Elemente aus LP und CP vorhanden sind. Kapitel 4 erweitert dieses Modell um sog. *„Constraint Stores“*, die auch schon in früheren Ansätzen benutzt wurden.

3. Mixed Logical/Linear Programming

Der in dieser Arbeit weiterverfolgte Ansatz versucht, ein neues Modell für die Beschreibung der Probleme zu entwickeln.

3.1 Vorstellung des Modells

Das hier vorgestellte Modell „*Mixed Logical/Linear Programming*“ (MLLP) trennt die Variablen eines Problems in diskrete Variablen $\boldsymbol{y_1}..\boldsymbol{y_n}$ und kontinuierliche Variablen $\boldsymbol{x_1}...\boldsymbol{x_m}$.

Ein sog. Constraint wird in diesem Modell durch folgenden Ausdruck gegeben:

$$h(\boldsymbol{y_1}..\boldsymbol{y_n}) \rightarrow \boldsymbol{a_1x_1} + ... + \boldsymbol{a_mx_m} \geq \boldsymbol{b}$$

h(y) ist dabei ein prädikatenlogischer Ausdruck, der aufgrund der Variablenbelegung der diskreten Variablen entweder erfüllt ist oder nicht. Wenn *h(y)* erfüllt ist, dann tritt die lineare Ungleichung auf der rechten Seite in Kraft.

Diese lineare Ungleichung ist hier in der allgemeinen Form gegeben. Es müssen natürlich nicht alle X-Variablen darin vorkommen. Genauso hängt *h(y)* nicht von allen diskreten Y-Variablen ab.

Lineare Ungleichungen, die immer gelten sollen, können durch folgende Bedingung angegeben werden:

$$(0 = 0) \rightarrow \boldsymbol{a_1x_1} + ... + \boldsymbol{a_mx_m} \geq \boldsymbol{b}$$

Logische Ausdrücke, die immer erfüllt sein sollen, können durch folgende Bedingung angegeben werden:

$$\neg\, h(\boldsymbol{y_1}...\boldsymbol{y_n}) \rightarrow (1 = 0)$$

3.2 MLLP formal

gegeben:	MLLP = (Y, D, X, c, C)	
	$Y = \{y_1 ... y_m\}$	Menge diskreter Variablen
	$D: Y \rightarrow 2^Z$	Domains dieser Variablen
	$X = (x_1 ... x_n)$	Vektor kontinuierlicher Variablen
	$c = (c_1 ... c_n)$	Kostenfaktoren
	$C = (C_1 ... C_k)$	Menge von Constraints (s.o.)
gesucht:	$\sigma: Y \rightarrow Z$	Belegung der Y-Variablen
	$x \in R^n$ mit cx minimal	Belegung der X-Variablen
Bedingungen:	$\forall i=1...m: \sigma_i \in D_i$	
	$\forall j=1...k: C_j$ ist erfüllt	

3.3 Der Lösungs-Algorithmus

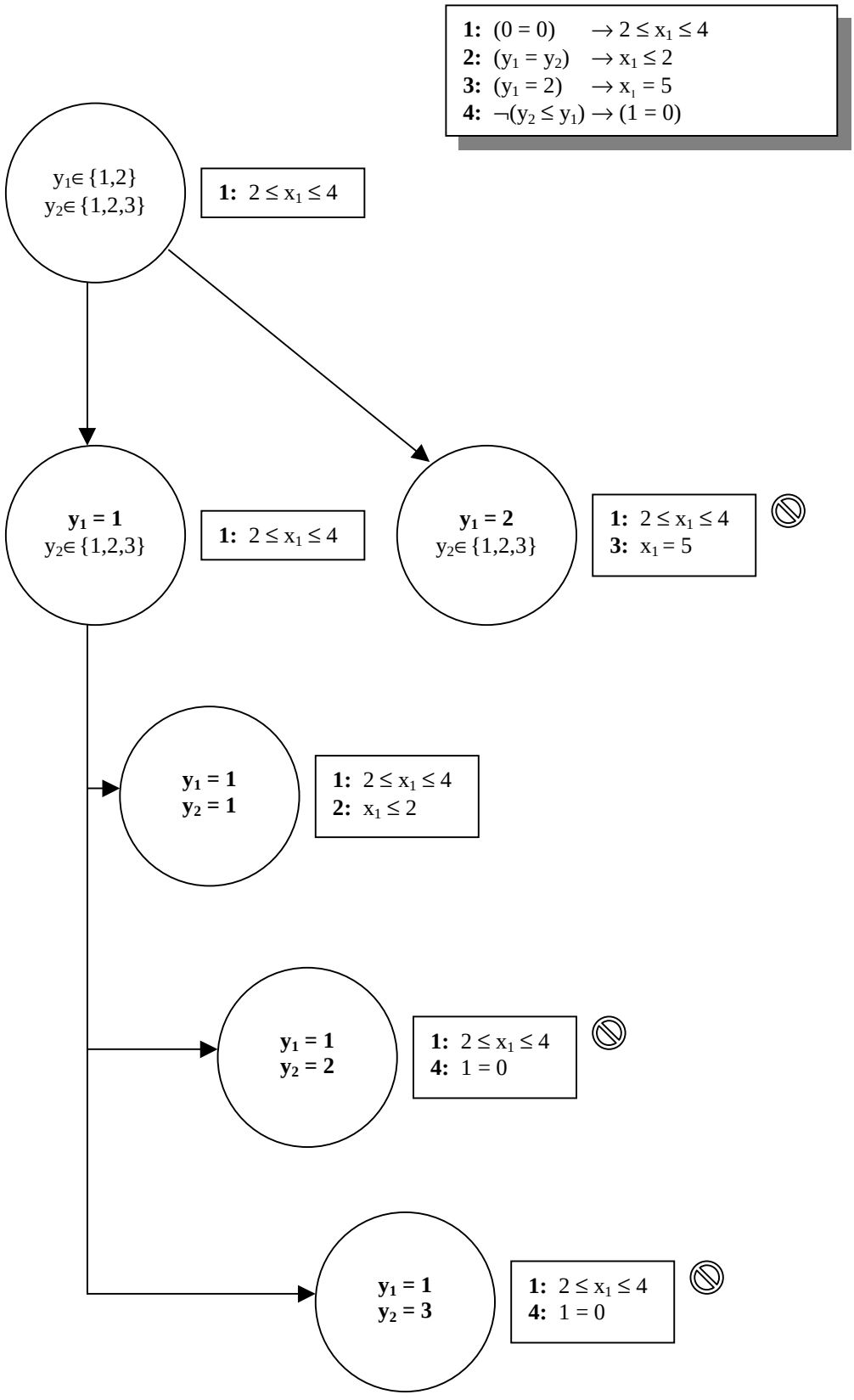
Die Suche im MLLP-Verfahren soll - wie bei CP - in einem Suchbaum stattfinden.

Am Wurzelknoten sind alle Variablen noch unbekannt. Jede diskrete Variable hat wieder eine sog. *Domain*, eine (endliche) Menge von gültigen Werten, die man dieser Variablen zuweisen kann.

Es wird eine der diskreten Variablen ausgewählt und alle Werte, die diese Variable aufgrund der zugeordneten Domain annehmen kann, aufgezählt. Für jeden Wert wird ein neuer Knoten gebildet.

An jedem Knoten wird nun mittels LP das Problem gelöst, das sich durch alle linearen Ungleichungen ergibt, für die die Bedingung aufgrund der Variablenbelegung wahr ist. Sollte dieses Gleichungssystem nicht lösbar sein oder ist die Lösung schlechter als die beste bisher gefundene Lösung, muß dieser Knoten nicht weiter untersucht werden.

Durch die Bildung neuer Unterbäume unter einem Knoten bleiben die geltenden Ungleichungen dieses Knotens bestehen, und es werden wahrscheinlich sogar noch neue hinzukommen. Deshalb muss unter einem als unlösbar oder schlecht lösbar erkannten Knoten nicht weitergesucht werden.



An den gekennzeichneten Knoten kann die Suche abgebrochen werden - das lineare Gleichungssystem kann nicht gelöst werden. Dies ist vor allem bei dem Knoten $y_1=2$ vorteilhaft, da man sich die Berechnung des kompletten weiteren Unterbaums spart.

3.4 MZP als MLLP-Problem

Sei m_i die Maschine, auf der Auftrag i ausgeführt wird. f_m sind die Kosten, die Maschine m verursacht.

min: $\sum_{m} f_m$

Es müssen folgende Bedingungen eingehalten werden:

1. Eine Maschine verursacht genau dann Kosten, wenn ihr mindestens ein Auftrag zugeordnet ist

$$\forall m: atleast(m, m_1 \ldots m_n, 1) \rightarrow f_m = C_m$$
$$\forall m: \neg atleast(m, m_1 \ldots m_n, 1) \rightarrow f_m = 0$$

2. Ein Auftrag kann frühestens zu einem bestimmten Zeitpunkt begonnen werden

$$\forall i: (0 = 0) \rightarrow T_i \geq F_i$$

3. Ein Auftrag muss bis zu einem bestimmten Zeitpunkt beendet sein

$$\forall i: (0 = 0) \rightarrow T_i + D_i \leq S_i$$

4. Zwei Aufträge auf der gleichen Maschine müssen nacheinander abgearbeitet werden

$$\forall i,j: ((m_i = m_j) \wedge (T_i < T_j)) \rightarrow T_i + D_i \leq T_j$$
$$\forall i,j: ((m_i = m_j) \wedge (T_i = T_j)) \rightarrow (1 = 0)$$

An diesem Beispiel sind die folgenden typischen MLLP-Eigenschaften zu beobachten:

- Die Programme sind so einfach und übersichtlich wie bei CP
- Da an jedem Knoten des Suchbaums nur die linearen Ungleichungen mittels LP gelöst werden, für die die Bedingung *h(y)* erfüllt ist, ergeben sich sehr kleine und effizient zu lösende Gleichungssysteme. Da alle X-Variablen kontinuierlich sind (also reelle Zahlen), handelt es sich immer um "echte" LP-Probleme (keine ILP-Probleme)

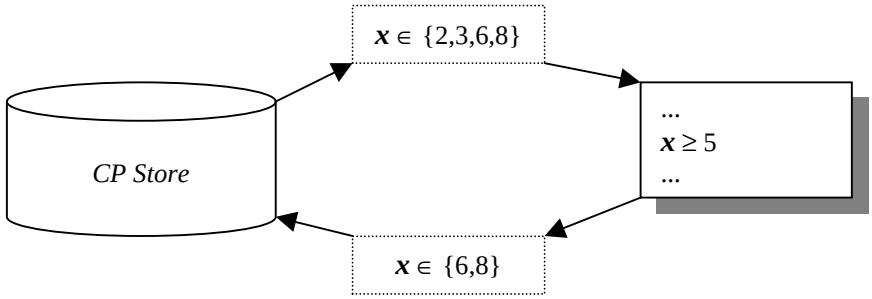
4. Constraint Stores

4.1 Grundüberlegungen

In der Theorie über CP gibt es das Konzept der „*Constraint Stores*“.

Man unterscheidet dabei zwischen *einfachen Bedingungen* und *komplexen Bedingungen*. Die einfachen Bedingungen lassen sich dabei effizient (in polynomieller Zeit) bearbeiten. Die einfachen Bedingungen sind in CP die Domains der einzelnen Variablen.

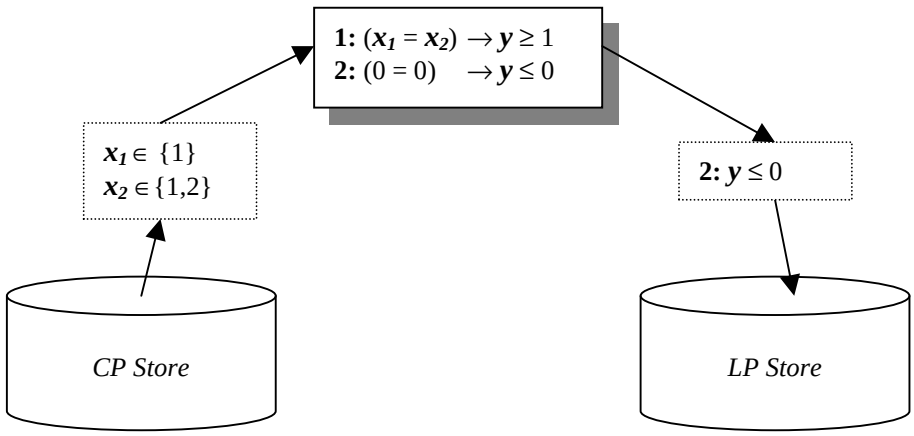
Ein CP-Algorithmus kann sich diese Domains aus dem Constraint Store holen und neue, kleinere Domains in den Constraint Store zurückschreiben, nachdem die Domains hinsichtlich der gegebenen Bedingungen geprüft wurden.



4.2 Constraint Stores in MLLP

Auch in MLLP können Constraint Stores eingerichtet werden. Dabei gibt es einen Constraint Store für die Domains der diskreten Variablen (den bereits bekannten CP-Store), und einen für die linearen Ungleichungen der kontinuierlichen Variablen (den LP-Store).

Ein MLLP-Algorithmus kann nun auf beide Stores zugreifen, und auch neue Bedingungen zu beiden Stores hinzufügen.



4.3 Zugriff auf den CP-Store

Ein wichtiger Unterschied zu der MLLP-Suche im Suchbaum ist, dass auch der Inhalt der CP-Stores verändert werden kann, d.h. die Domains der diskreten Variablen können verkleinert werden. Bisher war es nur möglich, die linearen Ungleichungen zu beeinflussen, die Domains blieben gleich.

Möglich ist dies durch folgende Überlegung:

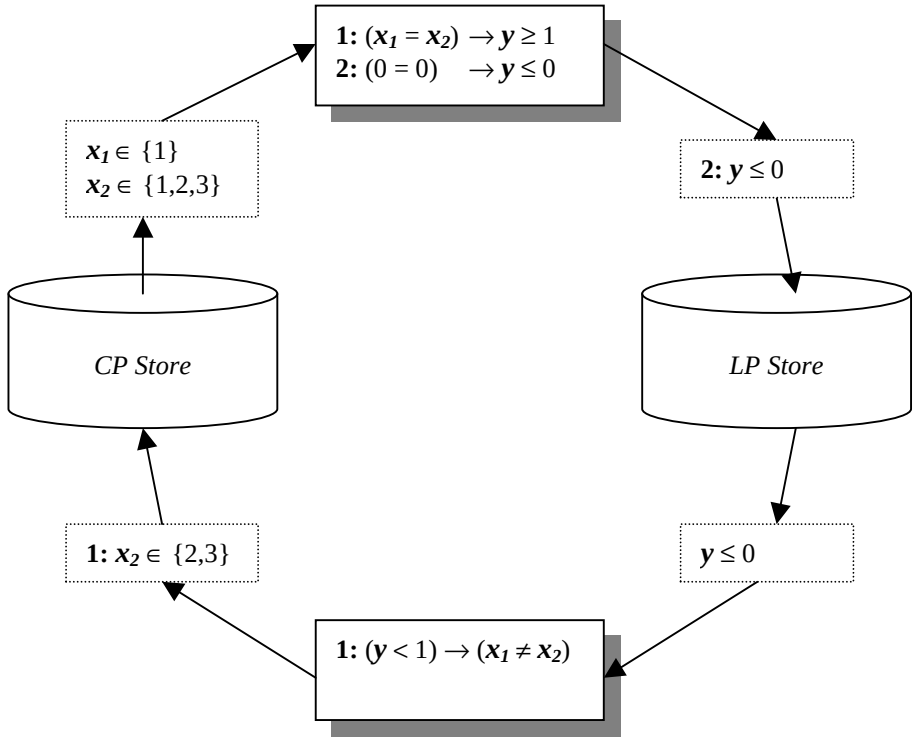
Gegeben sei die Bedingung $h(y) \rightarrow ax \geq b$. Wenn $h(y)$ wahr ist, dann tritt die Ungleichung auf der rechten Seite in Kraft. Wenn $h(y)$ aber falsch ist, kann keine Aussage über x gemacht werden. Dies ist das bekannte Verhalten des logischen Folgerungs-Operators.

Dieser Operator hat aber die Eigenschaft, dass die folgenden Ausdrücke äquivalent sind:

$$h(y) \rightarrow ax \geq b \qquad \Leftrightarrow \qquad \neg\,(ax \geq b) \rightarrow \neg\, h(y)$$

D.h. es ist durchaus möglich, dass der Inhalt des LP-Stores Auswirkungen auf die Domains des CP-Stores hat.

Im folgenden Beispiel wird zuerst aufgrund von Bedingung (**2**) eine lineare Ungleichung zum LP-Store hinzugefügt. Danach kann die äquivalente Form von Bedingung (**1**) die Domain der Variable x_2 verkleinern, denn der Wert 1 ist nicht mehr zulässig.



Durch die Anwendung der Constraint Stores im Suchbaum kann die Suche also weiter beschleunigt werden, da die Domains der diskreten Variablen im Lauf der Suche kleiner werden, und daher jeder Knoten im Lauf der Suche weniger Unterknoten hat.