Ausarbeitung im Rahmen des Seminars „Optimierung industrieller Anwendungsprobleme“ zum Vortrag mit dem Thema :

Integration of CP and OR methods for Matching Problems

Version 1.0

Autor: Thomas Elias

Inhaltsverzeichnis:

1. EINFÜHRUNG 3

1.1 INHALT DES ARTIKELS 3
1.2 ÜBERBLICK 3

2. VOM N-DAMENPROBLEM ZU CONSTRAINT PROPAGATION 4

2.1 DAS N-DAMENPROBLEM 4
2.2 SCHRITTWEISE LÖSUNGSFINDUNG 4
2.3 FORMULIERUNG ALS CONSTRAINT SATISFACTION PROBLEM 5
2.4 LÖSUNG DURCH CONSTRAINT PROPAGATION 6

3. DAS ASSIGNMENT PROBLEM 8

3.1 FORMULIERUNG DES ASSIGNMENT PROBLEMS 8
3.2 BEZUG ZUM N-DAMENPROBLEM 8
3.3 FORMULIERUNG ALS INTEGER LINEAR PROGRAM 8

4. DER UNGARISCHE ALGORITHMUS 10

4.1 KORRESPONDENZ 10
4.2 DAS PERFECT MATCHING PROBLEM 11
4.3 ZUNEHMENDE WEGE 11
4.4 EIN ALGORITHMUS ZUR BESTIMMUNG VON KORRESPONDENZEN 12
4.5 DER UNGARISCHE ALGORITHMUS 16

5. RELAXATION 22

6. ANHANG 24

6.1 PERSÖNLICHE KRITIK 24
6.2 LITERATURHINWEISE 24

1. Einführung

Dieses Dokument ist die Ausarbeitung zum Vortrag mit dem Thema „Integration of CP and OR methods for Matching Problems“. Der Vortrag wurde im Rahmen des Seminars „Optimierung industrieller Anwendungsprobleme“ gehalten.
Vorlage war dabei der gleichnamige Artikel von F. Focacci, A. Lodi und M. Milano.

1.1 Inhalt des Artikels

Der Artikel beschreibt die Integration des Assignment Problems, das durch den Ungarischen Algorithmus gelöst worden ist, in ein Constraint Satisfaction Problem, das durch Constraint Propagation gelöst worden ist.
Dabei ist hauptsächlich auf Details von diesem Lösungskonzept eingegangen worden, indem versucht worden ist, durch kleine Anpassungen und Heuristiken noch mehr Effizienz zu erhalten.
Das Verfahren wurde am n-Damenproblem evaluiert. Am Ende des Artikels werden dann als Ergebnis die Daten von durchgeführten Messungen verschiedener Varianten vorgestellt.

1.2 Überblick

Im Gegensatz zum Artikel, in dem das Verständnis des wesentlichen Konzepts vorausgesetzt worden war und man sich lediglich auf die genaue Ausführung von Details beschränkt hat, wurden in dem Vortrag und in dieser Ausarbeitung alle Voraussetzungen geschaffen, um den Grundgedanken des Artikels verständlich machen zu können.
Dabei lässt sich das behandelte Gebiet in drei Teile gliedern, die in den Kapiteln zwei bis vier zu finden sind. Als erstes wird das n-Damenproblem vorgestellt und als ein Constraint Satisfaction Problem formuliert. Anschliessend wird der Suchbaum aufgezeigt, der sich ergibt, wenn zur Lösung des Problems Constraint Propagation verwendet wird.
Im zweiten Teil wird das Assignment Problem beschrieben und als Integer Linear Program formuliert. Darüber hinaus wird der Zusammenhang zum n-Damenproblem verdeutlicht.
Der dritte Teil schildert den Ungarischen Algorithmus, der verwendet worden ist, um das Assignment Problem zu lösen. Dazu werden vorweg erforderliche Methoden aus der Graphentheorie aufgeführt, wobei insbesondere das Perfect Matching und die zunehmenden Wege betrachtet werden. Anschliessend wird ein Algorithmus genau untersucht, der in dem Ungarischen Algorithmus enthalten ist.
Nach diesen Vorüberlegungen wird dann der eigentliche Ungarische Algorithmus vorgestellt und seine Arbeitsweise durch ein Beispiel verdeutlicht.

Abschließend wird im fünften Kapitel auf den Zusammenhang der oben beschriebenen drei Gebiete eingegangen, wobei die Relaxation und die sich dadurch ergebenen Vorteile beschrieben werden.

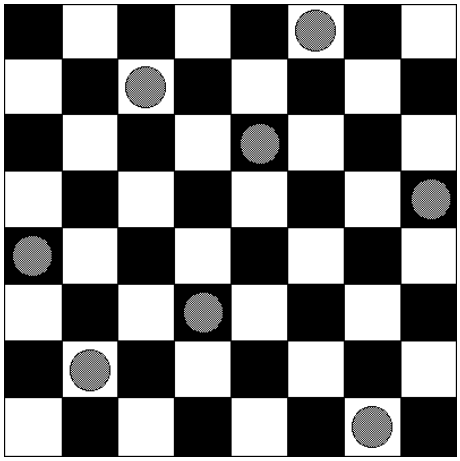
Am Ende der Ausarbeitung findet sich eine Kritik des zugrundeliegenden Artikels.

2. Vom n-Damenproblem zu Constraint Propagation

Dieses Kapitel beginnt mit einer Beschreibung des n-Damenproblems. Anschliessend wird gezeigt, wie man dieses Problem von Hand schrittweise lösen kann. Aus dieser Vorgehensweise wird dann ein Constraint Satisfaction Problem hergeleitet, das abschließend durch Constraint Propagation gelöst wird.

2.1 Das n-Damenproblem

Beim n-Damenproblem müssen auf einem n * n großen Schachbrett n Damen so angeordnet werden, dass sie sich nicht gegenseitig schlagen können. Das folgende Bild zeigt als Beispiel eine Lösung für n = 8:



2.2 Schrittweise Lösungsfindung

Aus der Beschreibung des n-Damensproblems im Abschnitt 2.1 kann man folgern, dass je zwei Damen nicht auf der gleichen horizontalen, vertikalen oder diagonalen Linie stehen dürfen, wobei sich eine Linie über das gesamte Spielbrett erstrecken muss.
Um eine eindeutige Position für eine Dame zu beschreiben, kann das Schachbrett in n Spalten und n Zeilen aufgeteilt werden. Die Spalten gehen dabei von links nach rechts und werden von 1 bis n durchnumeriert. Die Zeilen gehen von oben nach unten und werden ebenfalls von 1 bis n durchnumeriert.
Um alle Damen richtig zu platzieren kann man nach dem im Folgenden beschriebenen Schema vorgehen:

Der Reihe nach wird in jeder Spalte von 1 bis n eine Dame gesetzt. Das n-Damenproblem beschränkt sich nun also auf das finden einer geeigneten Zeile in jeder Spalte.

Bei der erste Spalte ist die Plazierung der ersten Dame auf jeder Zeile möglich. Wählt man beispielsweise die erste Zeile, so weiß man für alle noch folgenden Damen, dass sie nicht in der ersten Zeile platziert werden dürfen. Die von der ersten Dame ausgehende diagonale Linie verbietet die Plazierung auf die zweite Zeile in der zweiten Spalte und auf die dritte der dritten Spalte u.s.w.
Je mehr Spalten man auf dieser Weise abgearbeitet hat, desto weniger Möglichkeiten bleiben für die Auswahl von Zeilen. Wenn man in der letzten Spalte mindestens ein Feld in einer gültigen Zeile findet, so hat man das n-Damenproblem gelöst. Falls bei einer Spalte keine Zeile mehr zur Auswahl steht, so muss man so viele Schritte, beziehungsweise Spalten, zurückgehen, bis man bei einer Spalte die Dame auch auf eine andere Zeile platzieren kann.
Durch diese Änderung kann es dann passieren, dass man so eine Lösung findet. Es ist bei dem Problem bekannt, dass es ab n=5 immer Lösungen gibt.

2.3 Formulierung als Constraint Satisfaction Problem

Die im Abschnitt 2.2 aufgeführten Überlegungen lassen sich einfach in Constraints ausdrücken, um so das n-Damenproblem als Constraint Satisfaction Problem zu formulieren.
Dazu wird die Variable x_i eingeführt. i bezeichnet dabei die Nummer der Spalte und der Wert von x_i entspricht der Nummer der Zeile. Man benötigt somit n Variablen um die Positionen der Damen genau festzulegen.

Das folgende Beispiel zeigt ein Constraint Satisfaction Problem für n = 5:

Variablen: $x_1,..,x_5$

Domains: $D_1,..,D_5 = \{1,..,5\}$

Constraints:

1. $x_i \neq x_j$ für alle i,j mit: $1 \leq i < j \leq 5$
2. $x_i \neq x_j + (j - i)$ für alle i,j mit $1 \leq i < j \leq 5$
3. $x_i \neq x_j - (j - i)$ für alle i,j mit $1 \leq i < j \leq 5$

Zielfunktion:

$$\min. \sum_{i=1}^{n} \sum_{i=1}^{n} \chi \{x_i = j\}$$

1. Der erste Constraint bestimmt, dass in jeder Zeile nur eine Dame stehen darf. Die Einschränkung, dass auch in jeder Spalte nur eine Damen stehen darf muss nicht durch einen Constraint formuliert werden, da die Spalten sequentiell durchgegangen werden und in jeder Spalte nur eine Zeile ausgesucht wird.

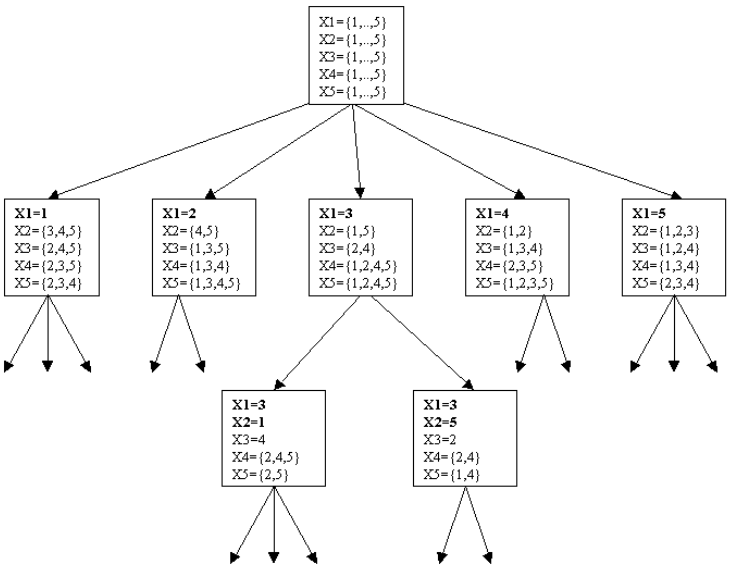
2. Der zweite Constraint sorgt dafür, dass entlang aller Hauptdiagonalen des Schachbretts nur jeweils eine Dame platziert werden darf. Die Hauptdiagonalen sind alle Linien, die von rechts oben nach links unten verlaufen.

3. Der letzte Constraint verhindert das Setzen von mehr als einer Dame auf jeder Nebendiagonalen des Schachbretts, also jeder Linie, die von links oben nach rechts unten verläuft.

4. Die zu minimierende Zielfunktion ist nur der Vollständigkeit halber hier aufgeführt. Beim n-Damenproblem spielen die Kosten keine Rolle, denn jede Lösung ist gleichwertig, unabhängig davon, auf welchen Zeilen die Damen platziert worden sind.
   Da zu einem Constraint Satisfaction Problem eine Zielfunktion gehört, wurde sie hier folgendermaßen gewählt: Das χ sorgt dafür, dass wenn dem x_i ein j zugeordnet worden ist, der Wert des Ausdrucks 1 ist und ansonsten 0. Alle sich so ergebenden Werte für alle zulässigen Paare i und j werden aufsummiert.
   Da den einzelnen Zuordnungen von Zeilen zu Spalten keine Kostenmatrix zugrunde liegt, ist das Ergebnis der Zielfunktion für jede gefundene Lösung des n-Damenproblems n.

2.4 Lösung durch Constraint Propagation

Das prinzipielle Vorgehen bei der Lösung des Constraint Satisfaction Problems durch Constraint Propagation ist der Aufbau eines Suchbaums. Jeder Knoten besteht dabei aus den n Variablen x_i und den noch zulässigen Werten, also Domains, die diesen Variablen zugewiesen werden können. Der Aufbau entspricht dem im Abschnitt 2.2 beschriebenen schrittweise Vorgehen.
Den Variablen der Wurzel des Suchbaums lassen sich noch alle Werte aus den entsprechenden Domains zuweisen. Bei der zweite Schicht des Suchbaums wird eine Variable aus dem Wurzelknoten gewählt und ihr alle möglichen Werte zugewiesen, wobei man n vom Wurzelknoten ausgehende Zweige bildet.
Bei jedem Knoten bestimmt man nun, wie auch beim Wurzelknoten, einen möglichen Wert für eine weitere Variable und vergrößert den Suchbaum danach auf gleiche Weise.
Falls bei einem Knoten allen Variablen ein Wert zugewiesen werden konnte, wurde eine Lösung gefunden. Entsteht entlang eines Zweiges innerhalb eines Knotens eine leere Domain für mindestens eine Variable, so muss die Suche entlang dieses Zweiges abgebrochen werden, da man keine gültige Lösung finden kann.
Das nächste Bild zeigt den Beginn eines möglichen Suchbaums für n = 5. Jeder Kasten steht für einen Knoten und enthält die Variablen mit den jeweils noch gültigen Domains.


X1={1,..,5}
X2={1,..,5}
X3={1,..,5}
X4={1,..,5}
X5={1,..,5}
X1=1
X2={3,4,5}
X3={2,4,5}
X4={2,3,5}
X5={2,3,4}
X1=2
X2={4,5}
X3={1,3,5}
X4={1,3,4}
X5={1,3,4,5}
X1=3
X2={1,5}
X3={2,4}
X4={1,2,4,5}
X5={1,2,4,5}
X1=4
X2={1,2}
X3={1,3,4}
X4={2,3,5}
X5={1,2,3,5}
X1=5
X2={1,2,3}
X3={1,2,4}
X4={1,3,4}
X5={2,3,4}
X1=3
X2=1
X3=4
X4={2,4,5}
X5={2,5}
X1=3
X2=5
X3=2
X4={2,4}
X5={1,4}

3. Das Assignment Problem

Dieses Kapitel beschreibt das Assignment Problem und zeigt eine mögliche Lösung in Form eines Integer Linear Programs.
Darüber hinaus wird der Zusammenhang zum n-Damenproblem gezeigt, das im ersten Kapitel behandelt worden und mit Constraint Propagation gelöst worden ist.

3.1 Formulierung des Assignment Problems

Beim Assignment Problem müssen die n Elementen einer Menge S n anderen Elementen einer Menge T zugeordnet werden, wobei mit jeder Zuweisung bestimmte Kosten entstehen, die zu minimieren sind.
Folgendes ist also beim Assignment Problem gegeben:

- Die Menge S = {1,..,n} und die Menge T = {1,..,n}
- Die Kosten c_{ij} aus der Menge der reellen Zahlen mit $i \in S$ und $j \in T$

Gesucht ist eine Zuweisung für die folgendes gilt:

- Jedem $i \in S$ muss genau ein $j \in T$ zuordnet werden
- Jedes $j \in T$ muss genau einem $i \in S$ zugewiesen werden
- Die Summe aller dabei entstehender Kosten c_{ij} muss minimal sein

3.2 Bezug zum n-Damenproblem

Im ersten Kapitel wurde gezeigt, dass man das n-Damenproblem darauf beschränken kann, dass man jeder Spalte genau eine Zeile zuordnet.
Falls man die Spalten als Elemente der Menge S ansieht und die Zeilen als Elemente der Menge Z, sieht man, dass das Assignment Problem ein Teilproblem des n-Damenproblem ist. Innerhalb des Assignment Problems wird nur abgedeckt, dass in jeder Spalte und in jeder Zeile nur eine Dame stehen darf. Es wird nicht beachtet, dass auf den diagonalen Linien keine zwei Damen stehen dürfen.

3.3 Formulierung als Integer Linear Program

Damit das Assignment Problem als Integer Linear Program (ILP) formuliert werden kann, muss die Variable $x_{ij} \in \{0,1\}$ mit der folgenden Bedeutung eingeführt werden:

- $x_{ij} = 1$, wenn $i \in S$ diesem $j \in T$ zugeordnet ist
- $x_{ij} = 0$, sonst

Das ILP kann nun wie folgt formuliert werden:

$$\min. \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad 1 \le j \le n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad 1 \le i \le n$$

$0 \le x_{ij} \le 1$ and integral, $1 \le i, j \le n$

- Die Zielfunktion summiert alle Kosten auf, die bei einer Zuweisung entstehen.

- Die erst Summenbildung sorgt dafür, dass jedem Element aus der Menge S nur ein Element aus der Menge T zugeordnet wird.
  Jede Zuordnung setzt die Variable x_{ij} auf den Wert 1. Falls mehr als eine Zuordnung durchgeführt worden wäre, wäre das Ergebnis der Summe größer als 1. Falls gar keine Zuordnung durchgeführt worden wäre, wäre die Summe 0.

- Analog zur ersten Summenbildung sorgt die zweite Summenbildung dafür, dass jedem Element aus der Menge T nur ein Element aus der Menge S zugeordnet wird.

Anmerkung:

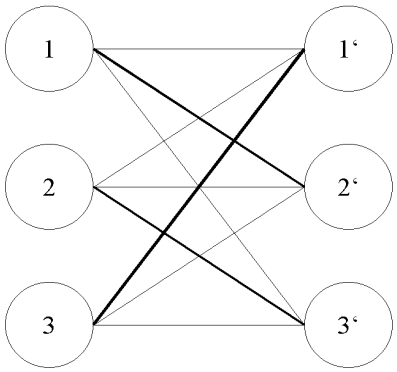
Die Formulierung des Assignment Problems als Integer Linear Program wurde in dem hier zugrunde liegenden Artikel aufgeführt. Zur Lösung des Assignment Problems wurde aber der Ungarische Algorithmus verwendet, auf den das nächste Kapitel näher eingeht. Aus diesem Grund wird an dieser Stelle nicht weiter auf das Gebiet des Integer Linear Programmings eingegangen.

4. Der Ungarische Algorithmus

Dieses Kapitel beschäftigt sich mit dem Ungarischen Algorithmus, der in dem Artikel verwendet worden ist, um das Assignment Problem zu lösen.
Um den Algorithmus besser erklären zu können, werden vorweg erforderliche Kenntnisse der Graphentheorie aufgeführt, wobei insbesondere das Perfect Matching und die zunehmenden Wege betrachtet werden. Anschliessend wird ein Algorithmus genau untersucht, der in dem Ungarischen Algorithmus enthalten ist.
Nach diesen Vorüberlegungen wird dann der eigentliche Ungarische Algorithmus vorgestellt und seine Arbeitsweise durch ein Beispiel verdeutlicht.

4.1 Korrespondenz

Eine Korrespondenz in einem Graphen G ist eine Menge K von Kanten, von denen keine zwei einen Endpunkt gemeinsam haben.
Eine Korrespondenz maximaler Mächtigkeit ist eine Korrespondenz, die nicht zu einer größeren Korrespondenz erweitert werden kann.
Das folgende Bild zeigt einen vollständigen bipartiten Graphen $K_{3,3}$. Die Korrespondenz maximaler Mächtigkeit wird dabei durch die dicken Kanten verdeutlicht.



Das Gewicht w(K) einer Korrespondenz K ist die Summe der Gewichte aller Kanten der Korrespondenz, wie die folgende Formel verdeutlicht (wobei E die Menge aller Kanten der Korrespondenz K ist):

$$w(K) = \sum_{e \in E} w(e)$$

4.2 Das Perfect Matching Problem

Das Perfect Matching Problem, wie es uns interessiert, kann formuliert werden als die Bestimmung einer Korrespondenz K maximaler Mächtigkeit mit maximalem Gewicht w(K) in einem vollständig bipartiten, gewichteten Graphen $K_{n,n}$.
Da eine Lösung für das im dritten Kapitel geschilderte Assignment Problem gesucht wird und dort die Zuweisungskosten minimiert werden, müssen bei Algorithmen, die das Perfect Matching Problem lösen, die Kosten umgerechnet werden, bevor man sie zur Lösung des Assignment Problem einsetzen kann.
Dazu benötigt man eine Variable C, die das maximale Gewicht einer Kante e des Graphen enthält. Dann ergeben sich die neuen Kosten $\gamma(e)$ jeder Kante e des Graphen dadurch, dass man von C die ursprünglich zu maximierenden Kosten w(e) abzieht.
Für jede Kante e ergibt sich also die folgende Rechnung: $\gamma(e) = C - w(e)$

Nach dieser Umrechnung lassen sich die Kosten einer Korrespondenz K maximaler Mächtigkeit mit maximalem Gewicht wie folgt berechnen, wobei E wieder die Menge aller Kanten der Korrespondenz K ist :

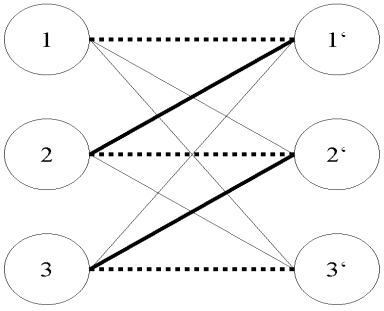
$$\gamma(K) = \sum_{e \in E} \gamma(e)$$

4.3 Zunehmende Wege

Zunehmende Wege werden oftmals in Algorithmen, wie beispielsweise im Ungarischen Algorithmus, verwendet, um eine Korrespondenz K durch eine Korrespondenz K‘ größerer Mächtigkeit zu ersetzen.

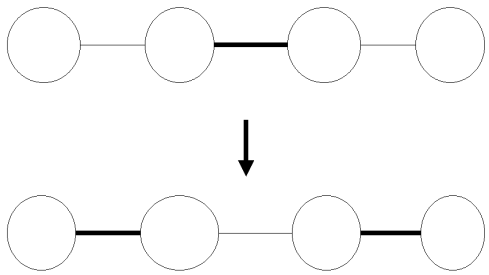
Vor weiteren Überlegungen müssen an dieser Stelle die Begriffe exponierte Punkte, gesättigte Punkte, alternierende Wege und letztlich zunehmende Wege beschrieben werden, was anschießend durch einen Beispielgraphen nochmals illustriert wird.

Jeder Punkt, beziehungsweise Knoten, der nicht auf einer Kante, die zu einer Korrespondenz K gehört, liegt, heißt exponiert bezüglich K.
Im Gegensatz dazu nennt man Punkte, die auf Kanten von K liegen, gesättigte Punkte.
Ein alternierender Weg (engl.: alternating path) bezüglich einer Korrespondenz K ist ein Weg, dessen Kanten abwechselnd in K sind und nicht in K sind.
Ein zunehmender Weg (engl.: augumenting path) ist ein alternierender Weg w, wobei die Endpunkte von w verschiedene exponierte Punkte sein müssen.



In dem oberen Bild stellen die dick eingezeichneten Kanten eine Korrespondenz dar. Zusammen mit den gestrichelten Kanten ergibt sich ein zunehmender Weg, der bei dem exponierten Punkt 1 beginnt und bei dem exponierten Punkt 3‘ endet.

Wenn auf einem zunehmenden Weg w die Rollen von Kanten und Nicht-Kanten der zugrundeliegenden Korrespondenz K vertauscht werden, so ergibt sich, wie das folgende Bild zeigt, eine Korrespondenz K‘ mit Mächtigkeit |K|+1.



Diese Vorüberlegungen führen zu dem „augmenting path theorem“, das die Grundlage fast aller bekannten Algorithmen für die Bestimmung von Korrespondenzen maximaler Mächtigkeit ist:

„Eine Korrespondenz K in einem Graphen G = (V,E) ist genau dann maximal, wenn es in G keinen zunehmenden Weg bezüglich K gibt.“

Dies ist leicht zu verstehen, wenn man zuerst annimmt, dass eine Korrespondenz K maximal ist. Falls es einen zunehmenden Weg w gibt, dann kann K wie oben gezeigt durch K‘ ersetzt werden. Dann ist K‘ eine Korrespondenz der Mächtigkeit |K| + 1, was ein Widerspruch zur obigen Annahme ist.

4.4 Ein Algorithmus zur Bestimmung von Korrespondenzen

Der in diesem Abschnitt erläuterte Algorithmus dient dazu, Korrespondenzen maximaler Mächtigkeit in einem ungewichteten bipartiten Graphen zu finden.
Er wird an dieser Stelle ausführlich erklärt, da er in dem Ungarischen Algorithmus enthalten ist.
Zu Beginn werden die vom Algorithmus benötigten Variablen aufgeführt.
Anschliessend wird der Algorithmus abgebildet und als erstes recht abstrakt erklärt.
Erst im zweiten Schritt erfolgt dann eine Erklärung, die sich näher am Algorithmus orientiert.

Gegeben sei also ein ungerichteter bipartiter Graph G = (V,E) mit V = S ∪ T, wobei S = {1,..,n} ist und T = {1‘,..,m‘}. Ohne Beschränkung der Allgemeinheit soll n ≤ m‘ gelten.

Anmerkung: Alle Knoten, die aus der Menge T sind, werden in diesem Abschnitt mit einem Hochstrich versehen, damit man sie innerhalb des Algorithmus sofort erkennen kann.

Das Ergebnis des Algorithmus ist eine Korrespondenz maximaler Mächtigkeit, die als ein Array „mate“ zurückgeliefert wird.

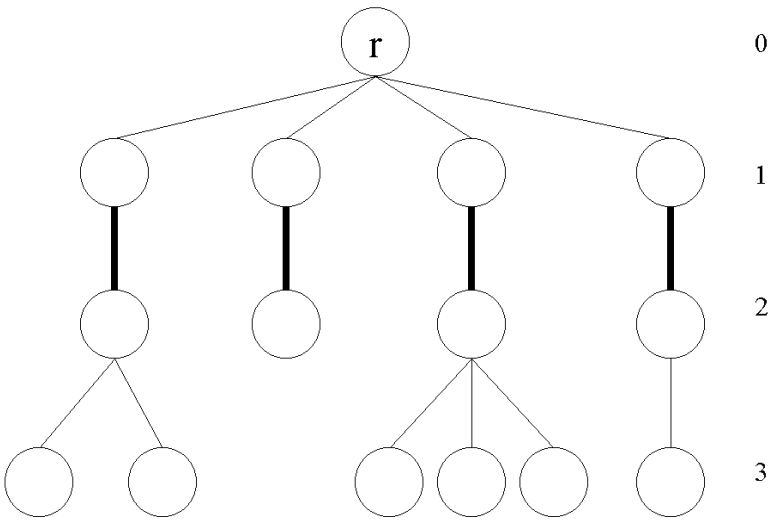
Die folgenden Variablen werden von dem Algorithmus und teilweise auch vom später beschriebenen Ungarischen Algorithmus verwendet:

mate: falls die Kante ij in einer Korrespondenz K enthalten ist, dann wird mate(i) = j und mate(j) = i gesetzt. Falls für einen Punkt k mate(k) = 0 ist, so ist dieser Punkt exponiert
p(y): gibt für $y \in T$ denjenigen Punkt in S an, der von y aus erreicht wurde
A_i: enthält die Punkte j, für die es eine Kante ij in G gibt
nrex: die Anzahl exponierter Punkte bezüglich K
N: die Anzahl der Knoten V
Q: enthält die Randmenge der Breitensuche
aug: wird auf true gesetzt, falls ein zunehmender Weg gefunden wurde

Hier nun der eigentliche Algorithmus:

```
(1) for v ∈ V do mate(v) = 0 od;
(2) r = 0; nrex = N;
(3)         while (nrex ≥ 2) AND (r < N) do
(4)         r = r + 1;
(5)         if mate(r) = 0
(6)        then for i=l to m do p(i‘)=0 od;
(7)                 Q = { }; füge r an Q an; aug = false;
(8)                 while aug = false und Q ≠ { } do
(9)                         entferne den ersten Punkt x von Q;
(10)                        if es gibt y ∈ Ax mit mate(y)=0
(11)                        then wähle ein derartiges y;
(12)                                while x ≠ r do
(13)                                        mate(y) = x; next = mate(x); mate(x) = y;
(14)                                        y = next; x = p(y)
(15)                                od;
(16)                                mate(y) = x; mate(x) = y; nrex = nrex - 2; aug = true
(17)                        else for y ∈ Ax do
(18)                                if  p(y) = 0 then p(y) = x; füge mate(y) an Q an  fi
(19)                                od
(20)                        fi
(21)                od
(22)        fi
(23) od.
```

Der Algorithmus findet die maximale Korrespondenz, indem versucht wird, einen zunehmenden Weg zu finden, wie es im Abschnitt 4.3 beschrieben worden ist.
Falls ein zunehmender Weg gefunden worden ist, wird aus diesem eine größere Korrespondenz erstellt und man versucht erneut einen zunehmenden Weg in der neuen Korrespondenz zu finden, bis dies nicht mehr möglich ist.
Die an dieser Stelle aufgeführten sieben Schritte, die später noch genauer beschrieben werden, sollen einen ersten Überblick über die Arbeitsweise des Algorithmus geben.
Das folgenden Bild soll dabei die verschiedenen Schichten aufzeigen, die durch die Breitensuche entstehen:



(1.) Als erstes muss ein bezüglich K exponierten Punkt r von G gewählt werden.
(2.) Falls es einen zu r adjazenten, exponierten Punkt s gibt, kann K um die Kante rs erweitert werden. Anderenfalls sind alle zu r adjazenten Punkte gesättigt und man beginnt eine Breitensuche bei r.
(3.) In Schicht 1 werden dann alle zu r adjazenten Punkte $a_1,..,a_p$ zusammengefasst.
(4.) Dann wird die Breitensuche von allen Punkten a_i fortgeführt.
(5.) In Schicht 2 werden nur die Punkte $b_i = mate(a_i)$ zusammengefasst, also wird nur den Kanten gefolgt, die zu K gehören, da man zunehmende Wege sucht.
(6.) In der Schicht 3 sucht man Punkte $c_1,..,c_q$, die Endpunkte einer mit b_i inzidenten Kanten sind und die Endpunkte von Kanten sind, die nicht zu K gehören.
(7.) Die Suche wird nun solange fortgesetzt, bis man in den Schichten 3,5,7,.. einen exponierten Punkt erreicht und somit einen zunehmenden Weg findet.

Nach diesem Überblick soll im nächsten Schritt der Algorithmus genauer erklärt werden. Dazu ist die folgende Definition hilfreich:

“Ein Teilbaum T von G mit Wurzel r heisst ein alternierender Baum, wenn r ein exponierter Punkt ist und wenn jeder in r beginnende Weg alternierend ist.”

Wie man sieht ist das obere Bild ein Beispiel für einen alternierenden Baum. Dieser Baum ist durch die Breitensuche innerhalb des Algorithmus entstanden, so dass man als Aufgabe des Algorithmus auch die Suche nach einem alternierenden Baum sehen kann.

Der oben aufgeführte Algorithmus wird in dem folgenden zweiten Schritt genauer erklärt, wobei man sich auf den Aufbau eines alternierenden Baumes T bezieht.

Man nehme an, dass die Schicht 2i-1 bereits konstruiert worden wäre. Falls kein Punkt in dieser Schicht exponiert ist, wird die nächste Schicht 2i konstruiert, in dem für jeden Punkt v aus Schicht 2i-1 w = mate(v) und die Kante vw zu T hinzugefügt wird.
Beim Aufbau der nächsten Schicht 2i+1 sei x ein Punkt der Schicht 2i und der Punkt y, für den $y \neq mate(x)$ gilt, ein mit x adjazenter Punkt. Dann sind bei einem zugrundeliegenden bipartiten Graphen die folgenden drei Fälle möglich:

Fall 1: Der Punkt y ist exponiert und liegt noch nicht in T. Dann ist ein zunehmender Weg gefunden.

Fall 2: Der Punkt y ist nicht exponiert und weder y noch mate(y) liegen in T. Dann lässt sich y in die Schicht 2i+1 hinzufügen und später mate(y) in die Schicht 2i+2.

Fall 3: Der Punkt y liegt bereits in T. Dann würde durch Hinzufügen der Kante xy in T ein Kreis entstehen. Diese Kante ist überflüssig, da man bereits einen in r beginnenden alternierenden Weg gefunden haben muss, der im Punkt y aufhört.

Falls man sich nun den oben aufgeführten Algorithmus nochmals ansieht, erkennt man, dass innerhalb der while-Schleife (Zeile 3) als erstes ein exponierter Punkt r gesucht wird. Diese Eigenschaft wird in der Zeile 5 abgeprüft. Der Punkt r wird dann zur Randmenge mit aufgenommen. In der Zeile 9 wird dann der Punkt wieder aus der Randmenge entfernt und mit x bezeichnet. In der nächsten Zeile findet die Überprüfung statt, ob es von x aus mindestens eine Kante zu einem y gibt, so dass y exponiert ist. Falls das zutrifft, wird innerhalb der while-Schleife von Zeile 12 bis 15 die nächste Schicht des alternierenden Baumes aufgebaut.
Falls das nicht zutrifft, werden innerhalb der for-Schleife von Zeile 17 bis 19 alle von x aus direkt erreichbaren Knoten in die Randmenge hinzugefügt, die alle nicht exponiert sind.

4.5 Der Ungarische Algorithmus

Der Ungarische Algorithmus arbeitet auf einem vollständigen und gewichteten bipartiten Graphen $K_{n,n}$ mit der Punktemenge $V = S \cup T$, mit $S = \{1,...,n\}$ und $T = \{1',..,n'\}$.
Darüberhinaus muss für jede Kante ij' des Graphen ein nicht-negatives Gewicht w_{ij} angegeben werden.

Das Ergebnis ist eine Korrespondenz mit maximaler Mächtigkeit und mit maximalem Gewicht , die durch ein Array "mate" beschrieben wird. Diese Korrespondenz wird im Folgenden als „perfekte Korrespondenz“ bezeichnet.

Der zum Finden der perfekten Korrespondenz nötige Aufwand beträgt beim Ungarischen Algorithmus $O(n^3)$.

Der im Abschnitt 4.4 beschriebene Algorithmus ist in dem Ungarischen Algorithmus enthalten. Zusätzlich werden Berechnungen durchgeführt, die sicherstellen, das die Korrespondenz mit maximalem Gewicht gefunden wird.
Bevor der Algorithmus abgebildet und anhand von einem Beispiel erklärt wird, werden weitere Variablen aufgeführt, die zur Berechnung der Gewichte verwendet werden.

Die Vektoren $u=(u_1,..,u_n)$ und $v=(v_1,..,v_n)$ heissen zulässig, wenn folgende Bedingung erfüllt wird:

(1) $u_i + v_j \geq w_{ij}$ für alle $i,j = 1,..,n$

Für jedes zulässige Paar (u,v) und für jede perfekte Korrespondenz gilt dann die folgende Ungleichung:

(2) $w(K) \leq M \leq (u_i + v_j)$ (M ist das Gewicht der perfekten Korrespondenz. Beim $(u_i + v_j)$ werden genau die u_i und v_j aufsummiert, die auch in der Korrespondenz K enthalten sind.)

Wenn es nun gelingt ein Paar (u,v) zulässiger Vektoren und eine Korrespondenz K zu finden, für die bei (2) Gleichheit gilt, so folgt, dass K die gesuchte perfekte Korrespondenz ist.

Die Variable δ_j gibt für jedes $j' \in T$ den jeweils erreichten minimalen Wert von $u_i + v_j - w_{ij}$ an. Hat die Variable den Wert 0, so zeigt das, dass es sich bei dem Paar (u,v) um dasjenige handelt, für das bei (2) Gleichheit gilt.

Auf der nächsten Seite wird nun der eigentliche Ungarische Algorithmus aufgeführt:

```
(1)  for v ∈ V do mate(v) = 0 od;
(2)  for i=l to n do u_i = max {w_ij: j=1,...,n}; v_i = 0 od;
(3)  nrex = n;
(4)  while nrex ≠ 0 do
(5)      for i=l to n do m(i) = false; p(i) = 0; δ_i = ∞ od;
(6)      aug = false; Q = {i ∈ S: mate(i) = 0};
(7)          repeat
(8)              entferne einen beliebigen Punkt i aus Q; m(i) = true; j=1;
(9)              while aug = false and j ≤ n do
(10)                 if mate(i) ≠ j‘
(11)                 then if u_i+v_j-w_ij < δ_j
(12)                         then δ_j = u_i+v_j-w_ij ; p(j) = i;
(13)                                 if δ_j = 0
(14)                                 then if mate(j‘)=0
(15)                                   then AUGMENT (mate,p,j'; mate);
(16)                                           aug = true; nrex = nrex-1
(17)                                      else Q = Q ∪ {mate(j‘)}
(18)                                      fi
(19)                                 fi
(20)                         fi
(21)                 fi
(22)                 j = j+l
(23)             od;
(24)             if aug = false and Q = { }
(25)             then J = { i ∈ S: m(i) = true}; K = (j' ∈ T: δ_j = 0};
(26)                 δ = min {δ_j: j‘ ∈ T\K};
(27)                 for i ∈ J do u_i = u_i-δ od;
(28)                 for j' ∈ K do v_j = v_j+δ od;
(29)                 for j' ∈ T \ K do δ_j = δ_j - δ od
(30)                 X = {j' ∈ T\ K: δ_j = 0};
(31)                 if mate(j‘) ≠ 0 für alle j' ∈ X
(32)                 then for j' ∈ X do Q = Q ∪ {mate(j‘)} od
(33)                 else   wähle j' ∈ X mit mate(j‘)  = 0;
(34)                        AUGMENT (mate,p,j'; mate);
(35)                        aug = true; nrex = nrex-1
(36)                 fi
(37)             fi
(38)         until aug = true
(39) od.
```

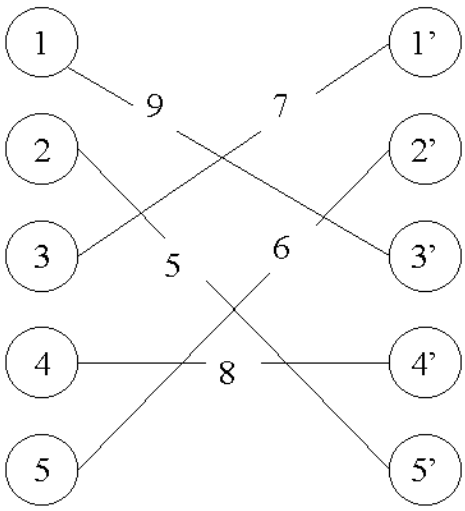
Prozedur AUGMENT (mate,p,j'; mate)

```
(1)      repeat
(2)              i = p(j); mate(j‘) = i; next = mate(i); mate(i) = j';
(3)              if next ≠ 0 then j' = next fi
(4)      until next = 0.
```

Die wichtigsten Schritte des Ungarischen Algorithmus sollen nun durch das folgende Beispiel gezeigt werden.

Gegeben sein ein $K_{5,5}$ und eine Matrix W, in der alle Gewichtungen w_{ij} enthalten sind. Das untenstehende Bild zeigt nicht den ganzen Graphen, sondern nur die Kanten der perfekten Korrespondenz und nimmt somit das Ergebnis vorweg.
Die Zahlen rechts neben beziehungsweise unter der Matrix sind die u_i und v_j, wie sie sich durch die Initialisierung in der Zeile 2 ergeben.

$K_{5,5}$:



$W=(w_{ij})$:

$$\begin{pmatrix} 3 & 8 & 9 & 1 & 6 & 9 \\ 1 & 4 & 1 & 5 & 5 & 5 \\ 7 & 2 & 7 & 9 & 2 & 9 \\ 3 & 1 & 6 & 8 & 8 & 8 \\ 2 & 6 & 3 & 6 & 2 & 6 \\ \\ 0 & 0 & 0 & 0 & 0 & v/u \end{pmatrix}$$

Die while-Schleife, die sich von der Zeile 4 bis zur Zeile 39 erstreckt wird als Phase bezeichnet. Da ein $K_{5,5}$ bearbeitet wird, sind 5 Phasen zur Berechnung notwendig.

Phase 1:

Nach einigen Initialisierungen werden in der Zeile 6 alle exponierten Punkte aus der Menge S in die Randmenge Q aufgenommen (Q={1,2,3,4,5}).
In der Zeile 8 wird ein beliebiger Punkt, in diesem Beispiel 1, aus Q entfernt und bearbeitet. Die while – Schleife von Zeile 9 bis Zeile 23 geht nun nacheinander alle Punkte 1‘ bis 5‘ aus T durch. Bei jeder Kante, also (1,1‘) bis (1,5‘) wird geprüft, ob $\delta_j = u_i+v_j-w_{ij} = 0$ ist (Zeile 11 bis 13).
Dies ergibt sich zum ersten Mal bei (1,3‘) so dass diese Kante durch einen Aufruf von der Hilfsprozedur „AUGMENT“ in die Korrespondenz mit aufgenommen wird. Anschliessend wird aug auf true gesetzt, wodurch die erste Phase abgeschlossen wird.

Phase 2:

In der zweiten Phase ist die Randmenge Q = {2,3,4,5}, da der Punkt 1 schon abgearbeitet worden ist und nicht exponiert ist.
Die Berechnungen verlaufen analog zu denen in der ersten Phase, wobei hier die Kanten (2,1‘) bis (2,5‘) geprüft werden.
Bei der Kante (2,4‘) hat das δ_j zum ersten Mal den Wert 0, so dass diese Kante in die Korrespondenz mit aufgenommen wird.

Phase 3:

Am Anfang der dritten Phase ist die zu untersuchende Randmenge Q = {3,4,5}. Wiederum werden alle Kanten (3,1‘) bis (3,5‘) untersucht. Bei der Kante (3,4‘) wurde für das δ_j wieder der Wert 0 ermittelt. Die Überprüfung in der Zeile 14 zeigt aber, dass der Punkt 4‘ bereits gesättigt ist, so dass mate(4‘) = 2 in die Rangmenge mit aufgenommen wird.
Innerhalb der while-Schleife von Zeile 9 bis 23 wurden nun alle Kanten durchprobiert, ohne dass eine weitere Kante zur Korrespondez hinzugefügt werden konnte. Aus diesem Grund muss die repeat-Schleife ab der Zeile 7 nochmals durchlaufen werden. Jetzt ist Q={2,4,5} und es werden die Kanten (2,1‘) bis (2,5‘) geprüft. Bei der Kante (2,5‘) ist der Wert von δ_j wieder 0 und die Korrespondenz kann vergrössert werden. Sie besteht nun aus den Kanten (1,3‘), (2,5‘) und (3,4‘).

Phase 4:

Die vierte Phase beginnt mit der Randmenge Q = {4,5}. Beim Abarbeiten der Kanten (4,1‘) bis (4,5‘) hat das δ_j bei der Kante(4,4‘) und (4,5‘) den Wert 0 angenommen. Bei beiden Kanten ist man aber auf bereits gesättigte Punkte gestossen, so dass die Randmenge erweitert werden musste. Bei der ersten Kante um den Punkt 3, der mate(4‘) ist und bei der zweiten Kante um den Punkt 2, der mate(5‘) ist.

Beim zweiten Durchlauf der repeat-Schleife enthält die Randmenge Q die Punkte 2, 3 und 5.

Beim Überprüfen der Kanten (2,1‘) bis (2,4‘) hat δ_j nie den Wert 0 angenommen. Die letzte Kante (2,5‘) ist bereits in der Korrespondenz enthalten, was die Abfrage in der Zeile 10 ermittelt.

Der dritte Durchlauf der repeat-Schleife, bei dem mit der Randmenge Q = {3,5} gearbeitet worden ist, brachte genau wie der zweite Durchlauf keine neue Kante für die Korrespondenz.

Der vierte Durchlauf der repeat-Schleife, in dem die Randmenge nur noch den Punkt 5 enthält, konnte bereits bei (5,2‘) eine Kante ausfindig machen, für die δ_j wieder den Wert 0 hat. Die Phase endete mit der Aufnahme dieser Kante in die Korrespondenz und dem Setzen von aug auf true.

Phase 5:

Die Randmenge Q enthält den einzigen noch exponierten Punkt 4. Der erste Durchlauf der repeat-Schleife verhält sich exakt wie der bei der Phase 4, so dass der Punkt 2 und der Punkt 3 wieder in die Randmenge mit aufgenommen worden sind.

Die nächsten zwei Durchläufe, in denen zuerst der Punkt 2 und dann der Punkt 3 aus der Randmenge entfernt wurden, konnten keine neue Kante für die Korrespondenz finden, da entweder δ_j nicht den Wert 0 angenommen hat oder man auf Kanten gestossen ist, die bereits in der Korrespondenz enthalten sind.

Jetzt tritt zum ersten Mal der Fall auf, dass alle Punkte der Randmenge abgearbeitet worden sind ohne dass eine neue Kante gefunden worden ist. Dadurch werden die Anweisungen zwischen der Zeile 24 und 38 ausgeführt, die das zulässige Paar (u,v) anpassen.
Dazu werden in der Zeile 25 zwei neue Mengen ermittelt. Die Menge J enthält alle Punkte aus S, die bei den Durchläufen in der aktuellen Phase aus der Randmenge entfernt worden sind. Der Inhalt von J ist also {2,3,4}.
Die Menge K enthält alle Punkte aus T, für die man bei den Durchläufen festgestellt hat, dass sie bereits gesättigt sind. K ist also {4‘,5‘}.
In der Zeile 26 wird dann der kleinste Wert von allen δ_j ermittelt, die zu Punkten der Menge T\K gehören, also {1‘,2‘,3‘}. Dieser Wert ist der Spielraum, den man nutzen kann, um das zulässige Paar(u,v) neu anzupassen. Dieser Wert ist hier 1. Er wird von allen Variablen u_i, die zu J gehören, abgezogen und zu allen Variablen v_j, die zu K gehören, addiert.
Nach diesen Korrekturen ist u = (9,4,8,7,6) und v = (0,0,0,1,1).
In der Zeile 30 werden alle $j‘ \notin K$ mit $\delta_j = 0$ in der Menge X zusammengefasst. Falls ein $j‘ \in X$ exponiert ist, konnte ein zunehmender Weg gefunden werden und man kann die Korrespondenz mit einem Aufruf von AUGMENT in den Zeilen 33 bis 35 vergrössern. Anderenfalls kann für jedes $j‘ \in X$ der Punkt mate(j‘) in Q eingefügt werden, weswegen Q nicht mehr leer ist.

In diesem Fall ergibt sich X = {2‘} und mate(2‘) = 5 wird in Q eingefügt.

Danach wird die repeat-Schleife wieder aufgerufen, wobei 5 aus Q entnommen wird. Auch in diesem Durchlauf konnte die Korrespondenz nicht vergrössert werden, so dass Q wieder leer ist und das zulässige Paar(u,v) neu angepasst werden muss.

Dieses Mal ist nach den Änderungen u = (9,3,7,6,5) und v = (0,1,0,2,2). Im Gegensatz zur ersten Änderung wird jetzt ein exponiertes j‘ ∈ X gefunden (1‘). Ein Aufruf von AUGMENT fügt erst die Kante (3,1‘) in die Korrespondenz mit ein. Da der Punkt 3 aber schon gesättigt war, tritt nun der Fall auf, dass entlang des zunehmenden Weges die Rollen von Kanten und Nicht-Kanten vertauscht werden, wie es im Abschnitt 4.3 gezeigt worden ist.
Dadurch wurde die Kante (3,4‘) zu einer „Nicht-Kante“ der Korrespondenz, die im nächsten Schritt um die Kante (4,4‘) erweitert wird.

Nach dem Ausführen von AUGMENT wird aug wieder auf true gesetzt. Jetzt ist die Anzahl der exponierten Punkte, die in nrex festgehalten wird, 0 und der Ungarische Algorithmus endet mit der Korrespondenz (1,3‘), (2,5‘), (3,1‘), (4,4‘) und (5,2‘) als Ergebnis.

5. Relaxation

Dieses Kapitel geht auf den Zusammenhang der zuvor beschriebenen Gebiete Constraint Propagation, Assignment Problem und Ungarischer Algorithmus ein.

Die Grundidee bei der Methode der Relaxation ist, dass es bei vielen Problemen genügt, eine "nahezu optimale" Lösung statt des optimalen Ergebnisses zu finden. Dazu werden untere Schranken benötigt, so dass man eine maximal erlaubte Abweichung von der tatsächlichen optimalen Lösung nachweisen kann.

Etwas formaler ausgedrückt versteht man unter Relaxation den Übergang von dem Problem P zu einem P umfassenden Problem P', das durch Abschwächung einiger Bedingungen von P konstruiert worden ist.
Man kann nun eine optimale Lösung von P' berechnen. Für den Fall von Minimierungsproblemen ist dann das Gewicht w(P') eine untere Schranke für das Gewicht w(P).
Nun kann man das eigentliche Problem P solange optimieren, bis $w(P) \leq w(P')$ ist und man so eine "nahezu optimale" Lösung gefunden hat.

Damit die Relaxation durchführbar ist, muss w(P') eine möglichst gute Approximation für w(P) liefern und das Problem P' muss effizient lösbar sein.

Im Bezug auf das Constraint Satisfaction Problem, das im Abschnitt 2.4 durch Constraint Propagation gelöst worden ist, lässt sich eine Assignment-Relaxation durchführen, mit dem Ziel, den entstehenden Suchbaum möglichst stark zu verkleinern, so dass die Lösung schneller gefunden werden kann.

Die Assignment-Relaxation liefert an jedem Knoten des Suchbaums eine Lösung mit bestimmten Kosten Z. Beim ersten Aufruf vom ersten Knoten aus, initialisiert diese Lösung die Variable Z_{min}, die die bisher niedrigsten Kosten speichert.

Falls bei jedem weiteren Knoten des Suchbaums die durch die Assignment-Relaxation berechneten Kosten größer sind als Z_{min}, kann die Suche entlang diesem Zweig des Suchbaums abgebrochen werden. Falls sie geringer sind, muss dieser Zweig weiter untersucht werden und die neuen Kosten werden Z_{min} zugewiesen.
Auf diese Weise lassen sich somit ganze Zweige des Suchbaums einsparen.

Eine andere Methode, die darauf abzielt den Suchbaum enger werden zu lassen, ist unter dem Begriff "Domain Reduction" bekannt. Dabei wird untersucht, welche Domains des aktuellen Knotens im Suchbaum zu keiner Verbesserung der Kosten führen können. Diese Domains werden dann entfernt, so dass von diesem Knoten aus weniger weitere Verzweigungen ausgehen müssen. Zur Veranschaulichung der "Domain Reduction" soll das folgende Beispiel dienen:

Man nehme an, dass man sich inmitten des Suchbaums befindet und man eine Lösung mit Kosten 100 bereits gefunden hat, was bedeutet, dass auch die Variable Z_{min} den Wert 100 hat.
Weiter nehme man an, dass die Kosten bei dem aktuellen Knoten des Suchbaums bereits 96 betragen. Die Kosten für die nächste Zuweisung $X_i = j$ sind dann $96 + c_{ij.}$

Dadurch ergibt sich, dass bei jedem Paar ij mit $c_{ij} > 3$ j von der Domain von X_i entfernt werden kann.

Innerhalb des Artikels wurden Resultate von Messungen beschrieben, die mit dem n-Damenproblem für n = 12, 15 und 18 durchgeführt worden sind. Dabei wurden jeder Zuordnung von einer Spalte zu einer Zeile zufällig Kosten zugeordnet.
Die folgende Tabelle enthält die Zeiten in Sekunden auf einem Pentium II 200 MHz Rechner und die Anzahl der ungültigen Zuordnungen, die während der Suche vorgekommen sind.
Dabei sind zwei Messreihen angegeben, die erste (Pure-CP) gibt die Werte für die jeweilige Berechnung an, die sich ergeben haben, wenn nur CP zur Lösung verwendet worden ist.
Die zweite Messreihe (CP-AP) gibt die Werte an, die sich ergaben, als zusätzlich noch die oben beschriebene Assignment-Relaxation durchgeführt worden ist:

n	12	12	15	15	18	18
	Time	*Fails*	*Time*	*Fails*	*Time*	*Fails*
Pure-CP	0.80	1300	27.20	37000	242.30	290000
CP-AP	0.20	295	3.20	3600	6.10	5000

6. Anhang

Zum Abschluss findet sich hier eine kurze persönliche Kritik zu dem zugrundeliegenden Artikel und eine Auflistung der Literatur, die verwendet worden ist um den Vortrag und die Ausarbeitung erstellen zu können.

6.1 Persönliche Kritik

Der Artikel von F. Focacci, A. Lodi und M. Milano „Integration of CP and OR methods for Matching Problems“ erstreckte sich über lediglich 5 Seiten und war dadurch nur eine kurze Zusammenfassung des Lösungsweges. Detaillierte Informationen wurden dem Leser daher meistens vorenthalten. Darüber hinaus wurden umfangreiche Vorkenntnisse verlangt, zu denen die Inhalte nahezu aller hier beschriebener Kapitel gehören.
Aus diesem Grund wurde der Artikel nur als grobe Vorlage gewählt. Alle hier aufgeführten Gebiete wurden aus der unter 6.2 aufgeführten Literatur entnommen und vorgestellt.

Ein anderer störender Aspekt war die Wahl des n-Damenproblems zur Evaluierung der Assignment-Relaxation beim Constraint Propagation. Am Anfang konnte man die wesentlichen Methoden damit recht anschaulich beschreiben, aber spätestens bei der Relaxation hat dieses einfache Problem nicht mehr ausgereicht, da dort künstlich Kosten eingeführt werden mussten. Dadurch entstand das Verständnis für den hohen Aufwand durch die Relaxation und den Ungarischen Algorithmus erst am Ende der Einarbeitung ins Thema, nachdem das Prinzip der Relaxation verstanden war und man die Zusammenhänge begriff.
Die Wahl eines klassischen Zuordnugsproblems, bei dem von vornherein Kosten eine Bedeutung haben, hätte weniger Verwirrung verursacht.

6.2 Literaturhinweise

- F. Focacci, A. Lodi und M. Milano „Integration of CP and OR methods for Matching Problems“
- Dieter Jungnickel „Graphen, Netzwerke und Algorithmen“
- S. Vajda „Problems in Linear and Nonlinear Programming“
- Rainer E. Burkard, Ulrich Derigs „Assignment Problems: Solution Methods with FORTRAN-Programs
- Paul R. Thie „An Introduction to Linear Programming and Game Theory“
- Vijay Saraswat, Pascal Van Hentenryck „Principles and Practice of Constraint Programming“