

Seminar Methoden der KI (SS 2010)

Constraint Processing mit Hilfe von  
Baumzerlegung

Sebastian Wenzel

# Inhaltsverzeichnis

<b>INHALTSVERZEICHNIS</b>	<b>2</b>
<b>EINLEITUNG</b>	<b>4</b>
<b>Begriffe</b>	<b>4</b>
Constraints	4
Hypergraphen	4
Primal Graph	5
Dual Graph	5
Join-Graph	5
Connectedness	5
Join-tree	6
Hypertree	6
Acyclic Network	6
<b>LÖSEN VON ACYCLIC NETWORKS</b>	<b>6</b>
<b>ACYCLIC-SOLVING</b>	<b>6</b>
<b>Beispiel</b>	<b>7</b>
<b>ERKENNEN VON ACYCLIC NETWORKS</b>	<b>8</b>
<b>EINLEITUNG</b>	<b>8</b>
<b>Erkennen über Dual Graph:</b>	<b>9</b>
Algorithmus	9
Komplexität	9
Beispiel	9
<b>Erkennen über Primal Graph</b>	<b>10</b>
Begrifflichkeiten	10
Chordal Graph	10
Ordered Graph	10
Maximal cardinality ordering	10
Cliques	10
Conformality	10
Algorithmus	11

Beispiel	11
<b>BILDEN VON ACYCLIC NETWORKS</b>	<b>12</b>
Join Tree Clustering	12
Algorithmus	13
Komplexität	13
<b>AUSSICHT</b>	<b>13</b>
<b>QUELLENVERZEICHNIS</b>	<b>14</b>

## Einleitung

Beim Constraint Processing mit Baumzerlegung geht es darum, wie kann man Constraint Probleme mit Hilfe von Bäumen darstellen und lösen. Im Folgenden werde ich zeigen, welche Strukturen hierfür verwendet werden und welche Eigenschaften sie haben müssen, um ein Constraint Problem erfolgreich zu lösen. Die Baumzerlegung spielt hierbei eine große Rolle, da nur mit ihrer Hilfe allgemeine Probleme so umgeformt werden, dass sie mit diesen Algorithmen lösbar sind.

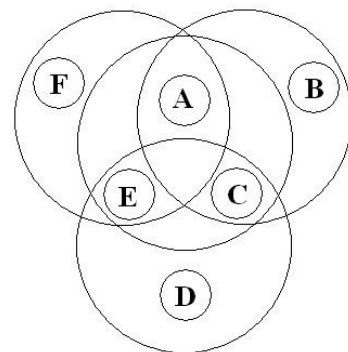
## Begriffe

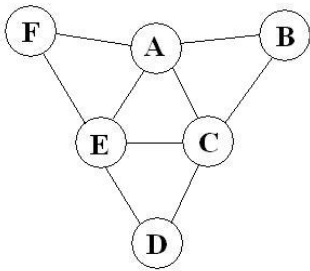
### Constraints

Im Folgenden wollen wir Constraints als gültige Lösungsbereiche betrachten. Z.B.: Wenn man sich alle Zahlen von 1-10 ansieht und sagt  $2 \cdot A = B$ , dann gilt für unser Constraint AB folgende Menge an gültigen Lösungen  $\{(1,2),(2,4),(3,6),(4,8),(5,10)\}$ . Diese Menge zu finden, kann je nach Aufgabenstellung schwierig sein. Außerdem können hier nicht endliche Mengen nicht abgebildet werden. Wir benötigen daher für diese Algorithmen immer einschränkende Domänen für jede Variable, welche wir aber nur indirekt durch absorbieren in die Constraints betrachten.

### Hypergraphen

Bei Hypergraphen verbindet eine Kante mehrere Knoten miteinander. Diese Kante wird als Hyperkante bezeichnet. Constraint Probleme mit mehr als 2 Variablen pro Constraint können mit Hilfe der Hypergraphen dargestellt werden. Hierbei werden die Variablen zu den Knoten und die Constraints sind die Hyperkanten.



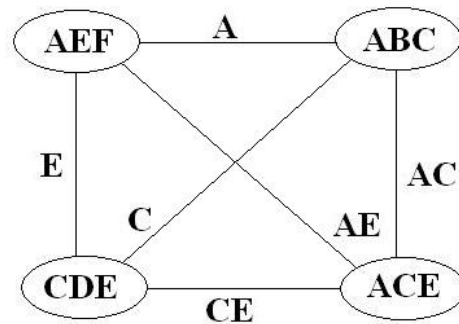


### Primal Graph

Der Primal Graph wird aus dem Hypergraphen gebildet. Hierbei werden die Hyperkanten durch mehrere einfache ungerichtete Kanten ersetzt. In der Abbildung erkennt man z.B., dass die Hyperkante AEF durch die drei Kanten AE, EF und AF ersetzt wurde. Hierbei ist zu beachten, dass keine Kante doppelt hinzugefügt wird. Die Kante AE stammt sowohl von der Hyperkante AEF, als auch von ACE ab. Sie ist trotzdem nur einmal vorhanden.

### Dual Graph

Der Dual Graph wird ebenfalls aus dem Hypergraphen gebildet. Beim Dual Graph dienen allerdings die Hyperkanten als Knoten. Zwei Knoten sind miteinander verbunden, wenn beide Hyperkanten im Hypergraphen gemeinsame Knoten hatten. So entsteht z.B. die Kante zwischen AEF und ACE, da beide die Knoten A und E im Hypergraphen besaßen.



### Join-Graph

Der Join-Graph ist eine Spezialisierung des Dual Graphen. Er fordert eine weitere Eigenschaft, die erfüllt sein muss. Bei dieser Eigenschaft handelt es sich um „Connectedness“

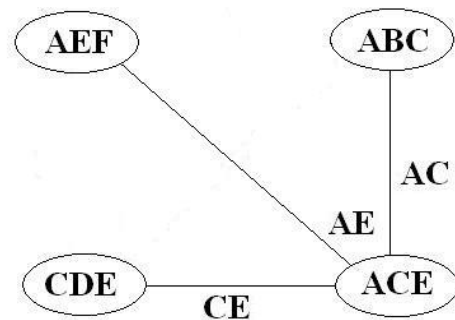
### Connectedness

Diese Eigenschaft fordert für alle Knotenpaare, die gemeinsame Variablen haben, dass die Paare über mindestens einen Weg verbunden sind, der diese Variablen enthält. Ein Weg kann hierbei aus einer oder mehreren Kanten bestehen.

## Join-tree

Der Join-tree ist eine Spezialisierung des Join-Graphs. Hierbei muss die Eigenschaft des Baumes erfüllt sein. Das heißt, es dürfen keine Kreise mehr vorhanden sein.

Die Herausforderung liegt nun darin, sowohl die Eigenschaft connectedness zu erhalten und gleichzeitig nur einen Baum zur Verfügung zu haben.



## Hypertree

Man bezeichnet einen Hypergraphen als Hypertree, wenn sich aus dem Hypergraphen ein Join-tree bilden lässt.

## Acyclic Network

Von einem Acyclic Network spricht man, wenn es sich um ein Constraint Problem handelt, dessen Hypergraphen ein Hypertree ist.

## Lösen von Acyclic Networks

Wir können nun Acyclic Networks lösen, indem wir den Baum traversieren und durch vergleichen der jeweiligen gültigen Teillösungen alle korrekten Gesamtlösungen erhalten.

## ACYCLIC-SOLVING

1. Sortieren der Constraints anhand der Position im Join Tree. Hierbei ist zu beachten, dass ein Constraint erst auftreten darf, wenn sein Vaterknoten bereits aufgetreten ist.
2. Man beginne mit dem letzten Constraint in der Liste und suche seinen Vaterknoten im Join-tree.

Vergleiche die gültigen Teillösungen der Constraints anhand ihrer gemeinsamen Variablen.

Entferne nicht gültige Teillösungen des Vaterknotens.

Sollte die Leere Menge entstehen, so gibt es keine Lösung.

Wiederhole bis zur Wurzel

3. Wenn niemals eine Leere Menge erreicht wurde gibt es eine Lösung.

Um diese nun zu erhalten, gehe den Baum von der Wurzel bis zu den Blättern und verwende die übrig gebliebenen Teillösungen, um eine gültige Gesamtlösung herzustellen

### Beispiel

Wir nehmen das am Anfang definierte Acyclic Network mit den Constraints ABC, AEF, CDE und ACE. Als gültige Teillösung legen wir fest:

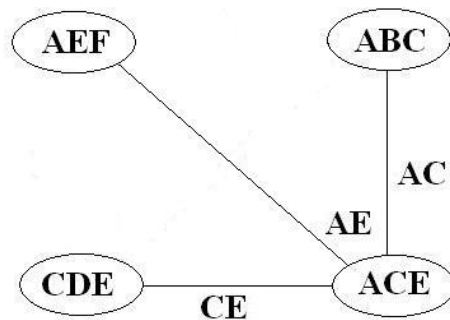
$$ABC = AEF = \{(0,0,1),(0,1,0),(1,0,0)\}$$

$$ACE = \{(1,1,0),(0,1,1),(1,0,1)\}$$

$$CDE = \{(1,0,1)\}$$

Wir verwenden den Join-tree in der Abbildung. Hierfür sortieren wir die Constraints in folgende Reihenfolge: [ACE,CDE,AEF,ABC]

ACE steht hierbei an erster Stelle, da es sich um die Wurzel handelt. CDE, AEF und ABC sind von der Reihenfolge her zufällig gewählt.



Bei Schritt 2 beginnen wir also nun von hinten mit ABC. Der Vaterknoten von ABC im Join-tree ist ACE. Also vergleichen wir die Teillösungen von ACE mit den Teillösungen von ABC, um zu entscheiden wie wir die gültigen Teillösungen von ACE einschränken können.

Schauen wir uns die Teillösungen von ACE mal an:

$$\{(A=1,C=1,E=0), (A=0,C=1,E=1), (A=1,C=0,E=1)\}$$

$(A=1,C=1,E=0)$  kann nicht durch eine Lösung von ABC abgebildet werden. Wir finden keine Teillösung in der gilt  $A = C = 1$ . Daher muss diese entfernt werden.

Bei  $(A=0,C=1,E=1)$  finden wir in ABC  $(A=0,B=0,C=1)$ , daher kann diese Lösung bleiben.

Genauso finden wir für  $(A=1,C=0,E=1)$  in ABC die gültige Teillösung  $(A=1,B=0,C=0)$ .

Damit hat ACE nun nur noch 2 gültige Teillösungen  $\Rightarrow \{(0,1,1),(1,0,1)\}$

Nun schauen wir uns das nächste Element der Liste an. Für AEF haben wir wieder ACE als Vaterknoten.

Hierbei finden wir nun für  $(A=1, C=0, E=1)$  keine gültige Teillösung.

Damit bleibt für ACE nur noch eine Teillösung übrig  $\Rightarrow \{(0, 1, 1)\}$

Für CDE sehen wir nun, dass für beide Constraints gilt:  $C = E = 1$

Daher wird keine Teillösung aus ACE entfernt.

Wir sind nun an der Wurzel des Baums und damit beim ersten Element der Liste angelangt. Da zu keiner Zeit im Algorithmus eine Leere Menge erreicht wurde, gibt es eine oder mehrere gültige Lösungen.

Um diese zu finden, beginnen wir mit Schritt 3.

Hierbei nehmen wir unsere gültige Teillösung  $A=0, C=1$  und  $E=1$  und gehen die Liste wieder vorwärts durch. Wir erweitern die Lösung nun folgendermaßen.

- Durch CDE erweitern wir die Lösung um D mit  $D=0$  und erhalten  $A=0, C=1, D=0$  und  $E=1$ .
- Bei AEF finden wir nur eine Teillösung, die auf unsere bisherige Lösung passt. Daher erweitern wir unsere Lösung um  $F = 0$ .
- In ABC gibt es ebenfalls nur eine Teillösung mit  $B=0$ , mit der wir unsere Lösung erweitern können.

Damit haben wir eine vollständige gültige Lösung  $\Rightarrow A=0, B=0, C=1, D=0, E=1, F=0$

## **Erkennen von Acyclic Networks**

### **Einleitung**

Da wir nun wissen, wie wir mit Hilfe eines Join-trees Constraint Probleme lösen können, müssen wir nun nur noch erarbeiten, wie man von einem Hypergraphen zu einem Join-tree kommt.

Hierfür bieten sich mehrere Alternativen an. Im Folgenden werde ich 2 davon vorstellen. Die erste Alternative startet mit dem Dual Graph, während die zweite mit dem Primal Graphen beginnt.

## Erkennen über Dual Graph

Wenn ein Hypergraph einen Join-Tree besitzt, dann ist dieser der maximal spannende Baum des Dualen Graphen, mit der Gewichtung anhand der Variablen pro Kante.

Allerdings ist zu beachten, dass die connectedness Eigenschaft weiterhin erfüllt sein muss.

Sollte diese Eigenschaft nach bilden des maximal spannenden Baums nicht mehr erfüllt sein, so ist der Hypergraph kein Acyclic Network.

## Algorithmus

Aus dieser Aussage ergibt sich folgender Algorithmus.

1. Gewichte die Kanten des Dual Graph mit der Anzahl seiner Variablen pro Kante
2. Erstelle den maximal spannenden Baum
3. Überprüfe die Eigenschaft connectedness, indem jeder Weg zwischen 2 Knoten auf die Variablen der Constraints überprüft wird.

## Komplexität

Die Komplexität des Algorithmus liegt bei  $O(n^3)$  mit  $n$  = Anzahl der Constraints (Knoten).

Beweis:

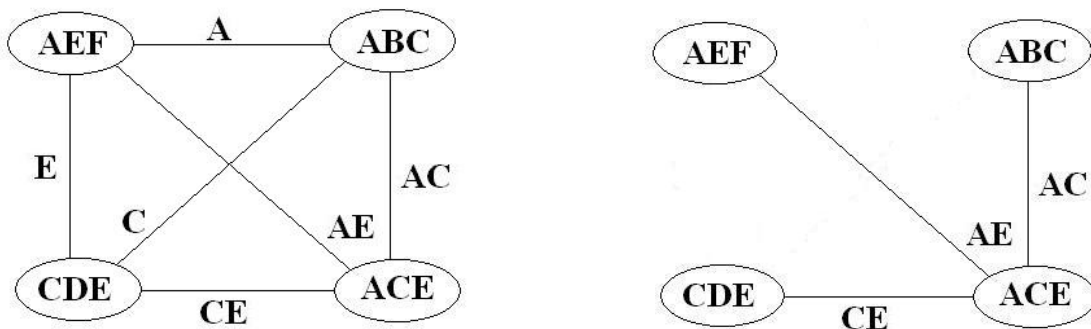
Schritt 1 kostet  $O(m)$ , da jede Kante einmal gewichtet werden muss.  $m$  liegt im schlimmsten Fall bei  $O(n^2)$ .

Das Erstellen des maximal spannenden Baums kann in  $O(n^2)$  ausgeführt werden.

Allerdings kostet Schritt 3  $O(n^3)$ , da wir  $O(n^2)$  Knotenpaare haben und das Überprüfen eines Knotenpaars bei  $O(n)$  liegt.

## Beispiel

Wenn wir den maximal spannenden Baum auf die Dual Graphen, aus der linken Abbildung anwenden, dann sehen wir, dass die drei Kanten mit einer einzigen Variable verschwinden. Hieraus ergibt sich der Join-tree, aus der rechten Abbildung, den wir im letzten Abschnitt erfolgreich verwendet haben.



# ***Erkennen über Primal Graph***

## **Begrifflichkeiten**

### **Chordal Graph**

Ein Graph erfüllt die Eigenschaft der chordality, wenn jeder Kreis in diesem Graphen, der die Länge 4 oder größer besitzt, durch eine Kante zwischen 2 Knoten des Kreises verbunden wird. Hierdurch ergeben sich nur minimale Kreise der Länge 3.

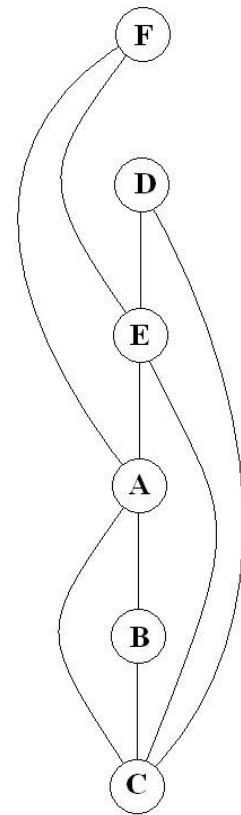
### **Ordered Graph**

Von einem Ordered Graph spricht man, wenn die Knoten in einer Ordnung vorliegen. Diese Ordnung dient zur besseren Abarbeitung des Graphen. Alle vorangehenden Knoten, die mit einem Knoten verbunden sind, werden als Elternknoten bezeichnet. Man beachte, dass der Graph hier von oben nach unten verarbeitet wird. Das heißt, z.B. E und A sind die Elternknoten von F usw.

### **Maximal cardinality ordering**

Die maximal cardinality ordering wird durch die Erfüllung weiterer Bedingungen erreicht. So muss für diese order folgendes gelten:

- Der erste Knoten der ausgewählt wird ist zufällig.
- Alle weiteren Knoten werden so gewählt, dass sie mit maximal vielen Knoten, die bereits ausgewählt wurden, verbunden sind.



### **Cliquen**

Bei einer Clique handelt es sich um einen vollständigen Teilgraphen innerhalb eines Graphen

### **Conformality**

Mit der Conformality wird der Zusammenhang zwischen dem Primal Graphen und dem ursprünglichen Constraint Problem wieder hergestellt. Hierbei werden die maximalen Cliques des Primal Graphen mit den Constraints verglichen. Sind beide gleich, so ist die Conformality erfüllt

## Algorithmus

Der zweite Ansatz einen Join-tree zu finden, basiert auf der Tatsache, dass ein Primal Graph von einem Hypertree chordal ist. Allerdings kann ein Primal Graph durch unterschiedliche Hypergraphen erstellt werden. Daher müssen wir die Conformality überprüfen.

Diese zwei Eigenschaften lassen sich nun durch die Hilfe der Ordered Graphs erreichen. So können wir mit der maximal cardinality ordering die chordality überprüfen. Denn in der maximal cardinality ordering ist die chordality erfüllt, wenn für jeden Knoten gilt, dass seine Elternknoten untereinander verbunden sind. Die Conformality können wir überprüfen, indem wir uns die Cliques des Graphen auflisten lassen. Dies lässt sich einfach erledigen, indem wir alle Knoten mit ihren Eltern auflisten.

Diese Erkenntnisse führen zu dem folgenden Algorithmus:

1. Test auf chordality:

Bilden der maximal cardinality ordering und überprüfen aller Knoten, ob ihre Eltern verbunden sind. Ist dies nicht der Fall, handelt es sich nicht um ein Acyclic Network.

2. Test auf conformality:

Liste für jeden Knoten die cliques mit seinen Eltern auf und überprüfe, ob es einen Constraint derselben Form gibt. Ist dies nicht der Fall, handelt es sich nicht um ein Acyclic Network.

3. Erstellen des Join-trees:

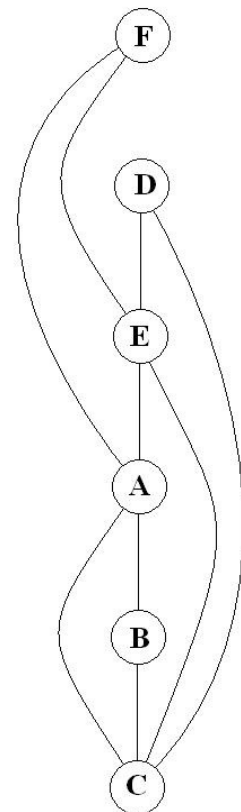
Sortiere die cliques nach ihrer höchsten Variable.

Starte von hinten und verbinde die clique mit einer anderen, mit der es die meisten Variablen gemeinsam hat.

Die Komplexität ist  $O(n^2)$ .

## Beispiel

Wir nehmen wieder unser Beispiel vom Anfang. Als erstes bauen wir die maximal cardinality ordering. Wir wählen den Knoten C zufällig als Startknoten aus. Dann bauen wir die order nach dem vorgegebenen Algorithmus auf. Im fertigen Graph, der in der Abbildung gezeigt wird, kann man nun erkennen, dass die Eltern von F – C alle untereinander verbunden sind.



Bei den maximalen cliques, aufgezählt von oben nach unten, handelt es sich um FAE (clique von F), DEC (clique von D), EAC (clique von E) und ABC (clique von A). Die cliques von B und C sind nicht maximal. Wir haben nun 4 maximale cliques, welche identisch zu unseren Constraints sind.

Als letztes müssen wir nun nur noch den Join-tree aufbauen. Daher verbinden wir clique FAE mit der clique EAC und DEC mit EAC, welche anschließend mit ABC verbunden wird. Hierdurch erhalten wir wieder den Join-tree, den wir am Anfang verwendet haben.

## **Bilden von Acyclic Networks**

Da wir Acyclic Networks effizient lösen können, zielen wir darauf ab, ein allgemeines Constraint Problem in ein Acyclic Network umzuwandeln. Dieses Ziel lässt sich dadurch erreichen, dass wir unsere vorhandenen Constraints nehmen und so gruppieren oder clustern, dass neue Constraints entstehen, welche dann ein Acyclic Network sind.

### ***Join Tree Clustering***

Es gibt mehrere Wege, um einen Hypergraph in einen Hypertree umzuwandeln. Die Herausforderung in so einer Umwandlung liegt darin, neue Constraints zu finden, die von der Größe so klein wie möglich sind.

Am einfachsten dient hierbei die Manipulation des Primal Graphen. Hierfür können wir den Algorithmus zum erkennen von Acyclic Networks verwenden. Wir verändern ihn so, dass er nicht mehr abbricht sobald eine Regel nicht erfüllt ist, sondern dass er den Graphen so manipuliert bis die Regel gültig ist.

So kann man im ersten Schritt beim chordality Test einfach die Kanten hinzufügen, die fehlen, damit alle Elternknoten untereinander verbunden sind. Hierbei lässt sich jede Ordnung verwenden. Allerdings empfiehlt sich die maximal cardinality ordering, da hier dann keine Kanten hinzugefügt werden müssen, wenn die chordality erfüllt ist.

Das hinzufügen von Kanten wird allerdings sehr wahrscheinlich dazu führen, dass die Conformality verletzt wird. Um diese Eigenschaft wieder für korrekt zu erklären, werden die gefundenen maximalen cliques als Constraints angenommen. Nun müssen nur noch aus den alten Constraints die neuen berechnet werden.

## **Algorithmus**

1. Wähle eine Ordnung der Variablen für den Ordered Graph  
(z.B. maxima cardinality ordering)
2. Wenn eine Verbindung zwischen den Knoten  $(i,j)$  und  $(k,j)$  besteht dann verbinde  $k$  mit  $i$
3. Erkenne die neuen maximalen cliquen und bilde den Join-tree
4. Füge jeden alten Constraint zu dem neuen Constraint hinzu, wenn Variablen gemeinsam sind.
5. Löse die neuen Constraints

## **Komplexität**

Die Komplexität ist von Schritt 5 abhängig, da hier die größte Rechenzeit verbraucht wird. Angegeben ist die Komplexität mit  $O(r_i \cdot k^{w^*(d)+1})$ . Hierbei ist  $r_i$  die Anzahl der Constraints für das Problem  $i$ .  $k$  ist die maximale Domaingröße und bei  $w^*(d)$  handelt es sich um die induced width der order  $d$ . Bei der induced width handelt es sich im groben um die maximale Anzahl der Variablen pro neuen Constraint. Daher zeigt sich hier wieder, wie wichtig es ist diese Größe klein zu halten.

## **Aussicht**

Die vorgestellten Algorithmen dienen nun dazu um Constraint Probleme zu lösen, in denen wir für die Constraints endliche gültige Teillösungen kennen. Mit diesem Algorithmus werden alle gültigen Lösungen gefunden. Es wird allerdings keine Aussage über die Güte der Lösungen gemacht. Das hat zur Folge, wenn wir viele gültige Lösungen haben, dass wir auch eine entsprechend höhere Laufzeit haben. Selbst wenn uns nur die besten 3 Lösungen genügen oder uns vielleicht sogar irgendeine reicht.

Was die Kosten des clusters angeht, so ist es sowohl Zeit als auch vom Speicherplatz her exponentiell. Allerdings wurde in meiner Quelle nicht darauf eingegangen, ob schon bewiesen wurde, dass es sich um ein NP-Vollständiges Problem handelt.

Da diese Ansätze nicht nur bei Constraints Anwendung finden, kann man davon ausgehen, dass hier sicherlich noch nicht die unterste Schranke der Komplexität erreicht ist.

## **Quellenverzeichnis**

Rina, Dechter 2003: Constraint Processing. Irvine