

# Constraint Programmierung für Scheduling

Ekaterina Tikhoncheva

10. Juli 2014

# Agenda

- 1 Constraint Programmierung
  - Constraint Satisfaction Problem
  - Lösung des Constraint Satisfaction Problems
  - Constraint Optimization Problem
  - Vorteile der Constraint Programmierung
- 2 Anwendung von CP zur Lösung der Scheduling Probleme
  - Minimieren der gewichteten Gesamtverspätung
  - Job Shop Scheduling
  - Timetabling

# Überblick

- 1 Constraint Programmierung
  - Constraint Satisfaction Problem
  - Lösung des Constraint Satisfaction Problems
  - Constraint Optimization Problem
  - Vorteile der Constraint Programmierung
- 2 Anwendung von CP zur Lösung der Scheduling Probleme
  - Minimieren der gewichteten Gesamtverspätung
  - Job Shop Scheduling
  - Timetabling

# Dame oder Tiger?

# Dame oder Tiger?

Ein Gefangener soll zwischen **drei Türen** wählen. Hinter einer von zwei Türen ist entweder **eine Dame** oder **ein Tiger** versteckt. Das dritte Zimmer ist **leer**. Auf jeder Tür hängt ein Schild. Das Schild auf der Tür zur Dame sagt Wahrheit. Das Schild auf der Tür zum Tiger lügt. Das Schild ins leere Zimmer kann sowohl lügen, als auch Wahrheit sagen.

I

Zimmer III ist leer.

II

Der Tiger ist im Zimmer I.

III

Dieses Zimmer ist leer.

# Dame oder Tiger?

I

Zimmer III ist leer.

II

Der Tiger ist im Zimmer I.

III

Dieses Zimmer ist leer.

Variablen:  $r_1, r_2, r_3 \in \{l, t, e\}$  für Zimmer ( $l = Dame, t = Tiger, e = leer$ )

$s_1, s_2, s_3 \in \{0, 1\}$  für Schilder.

Nebenbedingungen:  $r_i = t \rightarrow s_i = 0, r_i = l \rightarrow s_i = 1$

$s_1 = l \rightarrow r_3 \neq e, s_2 = l \rightarrow r_1 \neq t, s_3 = l \rightarrow r_3 \neq e$

Lösung:

# Dame oder Tiger?

I

Zimmer III ist leer.

II

Der Tiger ist im Zimmer I.

III

Dieses Zimmer ist leer.

Variablen:  $r_1, r_2, r_3 \in \{l, t, e\}$  für Zimmer ( $l = Dame, t = Tiger, e = leer$ )

$s_1, s_2, s_3 \in \{0, 1\}$  für Schilder.

Nebenbedingungen:  $r_i = t \rightarrow s_i = 0, r_i = l \rightarrow s_i = 1$

$s_1 = l \rightarrow r_3 \neq e, s_2 = l \rightarrow r_1 \neq t, s_3 = l \rightarrow r_3 \neq e$

Lösung:

Aus  $r_i = l \rightarrow s_i = 1$  folgt  $r_3 \neq l. \Rightarrow r_3 \in \{t, e\}$

# Dame oder Tiger?

I

Zimmer III ist leer.

II

Der Tiger ist im Zimmer I.

III

Dieses Zimmer ist leer.

Variablen:  $r_1, r_2, r_3 \in \{l, t, e\}$  für Zimmer ( $l = Dame, t = Tiger, e = leer$ )

$s_1, s_2, s_3 \in \{0, 1\}$  für Schilder.

Nebenbedingungen:  $r_i = t \rightarrow s_i = 0, r_i = l \rightarrow s_i = 1$

$s_1 = l \rightarrow r_3 \neq e, s_2 = l \rightarrow r_1 \neq t, s_3 = l \rightarrow r_3 \neq e$

Lösung:

Aus  $r_i = l \rightarrow s_i = 1$  folgt  $r_3 \neq l. \Rightarrow r_3 \in \{t, e\}$

Angenommen  $r_2 = l$ . Daraus folgt  $s_2 = 1, r_1 = t, s_1 = 0$

$r_3 = e$ . Aber dann ist  $s_1 = 0$ . D.h. Zimmer III kann nicht leer sein, wir sind zu einem Widerspruch gekommen.



# Dame oder Tiger?

I

Zimmer III ist leer.

II

Der Tiger ist im Zimmer I.

III

Dieses Zimmer ist leer.

Variablen:  $r_1, r_2, r_3 \in \{l, t, e\}$  für Zimmer ( $l = Dame, t = Tiger, e = leer$ )  
 $s_1, s_2, s_3 \in \{0, 1\}$  für Schilder.

Nebenbedingungen:  $r_i = t \rightarrow s_i = 0, r_i = l \rightarrow s_i = 1$

$s_1 = l \rightarrow r_3 \neq e, s_2 = l \rightarrow r_1 \neq t, s_3 = l \rightarrow r_3 \neq e$

Lösung:

Aus  $r_i = l \rightarrow s_i = 1$  folgt  $r_3 \neq l. \Rightarrow r_3 \in \{t, e\}$

Angenommen  $r_2 = l$ . Daraus folgt  $s_2 = 1, r_1 = t, s_1 = 0$

$r_3 = e$ . Aber dann ist  $s_1 = 0$ . D.h. Zimmer III kann nicht leer sein, wir sind zu einem Widerspruch gekommen.

Angenommen  $r_1 = l$ . Dann gilt:  $s_1 = 1, r_2 = t, s_2 = 0, r_3 = e, s_3 = 1$ . Die Lösung ist gefunden!

# Was ist Constraint Programmierung?

Die **Constraint Logische Programmierung** (engl. Constraint Logic Programming) oder einfach **Constraint Programmierung** (kurz **CP**) ist ein Ansatz zur Lösung eines Bedingungserfüllungs- oder Optimierungsproblems. Die zwei Hauptprinzipien von CP sind:

- Deduktion der zusätzlichen Nebenbedingungen aus den vorhandenen Nebenbedingungen durch logische Folgerungen
- Anwendung der Suchalgorithmen zum Untersuchen des Lösungsraums

Ursprünglich stammt es aus dem Bereich der künstlichen Intelligenz und gilt als eine Erweiterung der Ideen der logischen Programmierung.

# Constraint Satisfaction Problem

## Definition

Ein **Constraint Satisfaction Problem (CSP)** ist definiert durch

- die Menge  $X = \{x_1, x_2, \dots, x_n\}$  diskrete Variablen zusammen mit ihren endlichen Definitionsbereichen  $\{D_1, D_2, \dots, D_n\}$
- die Menge der Bedingungen  $C_{ijk\dots}$  zwischen den Variablen  $x_i, x_j, x_k, \dots$ , die die möglichen Werte der Variablen zusätzlich einschränken.

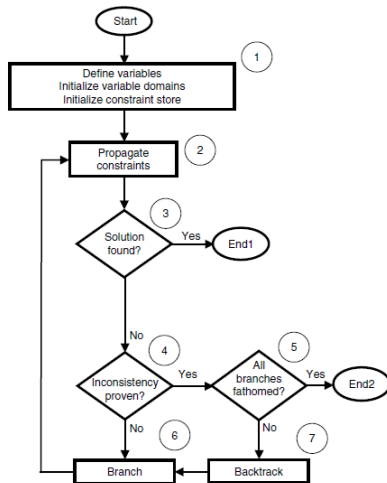
Ein CSP besitzt eine **(zulässige) Lösung**, wenn eine Zuweisung der Werte aus den Definitionsbereichen zu jeder Variable existiert, so dass alle Bedingungen erfüllt sind.

Die Variablen können von verschiedenen Typen sein:  
ganzzahlig, boolean, symbolisch, Mengen.

Dafür gibt es verschiedene Typen der Bedingungen:

- mathematische (Fertigstellungszeit = Startzeit + Bearbeitungszeit),
- disjunktive (Jobs  $J_1$  und  $J_2$  müssen an verschiedenen Maschinen abgearbeitet werden),
- relational (die Maschine  $I$  kann höchstens vier Jobs abarbeiten),
- explizite (die Jobs  $J_1$ ,  $J_2$  und  $J_5$  können nur auf der Maschine 1 abgearbeitet werden).

# Lösung des Constraint Satisfaction Problems



1: Initialisiere das Modell

Bild aus [3]

# Lösung des Constraint Satisfaction Problems

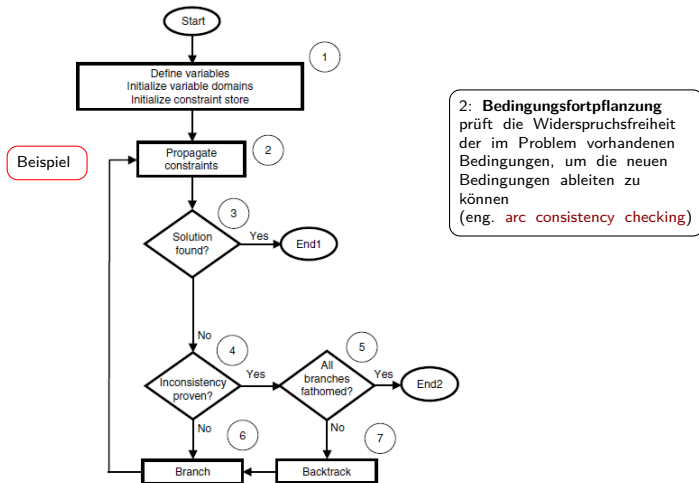
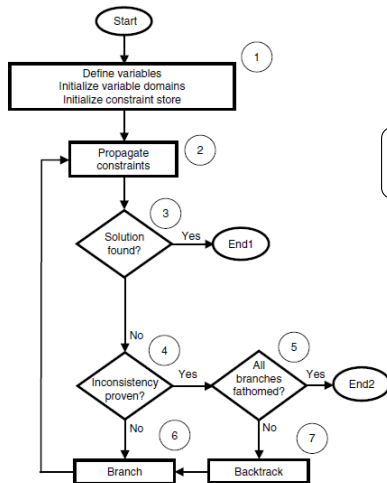


Bild aus [3]

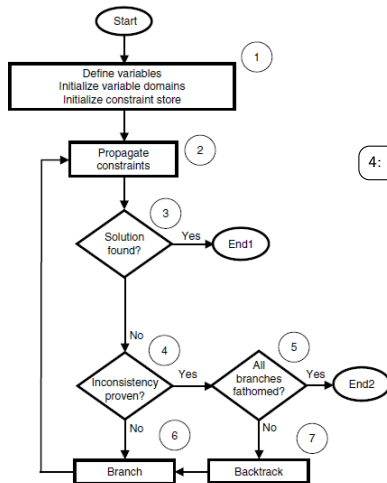
# Lösung des Constraint Satisfaction Problems



3: Falls eine Lösung des Problems gefunden ist, terminiert der Algorithmus

Bild aus [3]

# Lösung des Constraint Satisfaction Problems

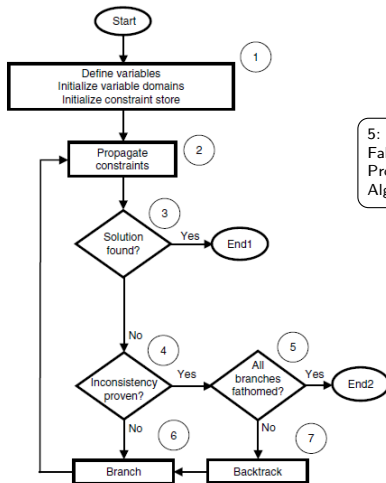


4: Ist das Problem widersprüchlich?

Bild aus [3]



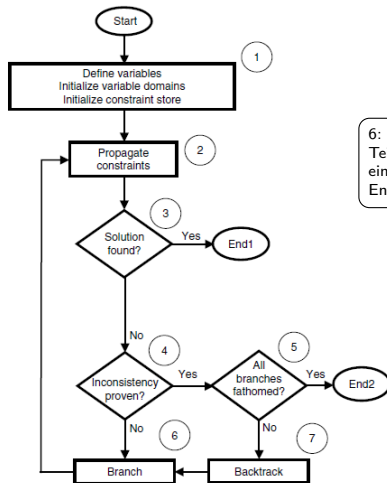
# Lösung des Constraint Satisfaction Problems



5: Sind alle Teilprobleme untersucht?  
Falls ja, dann ist das ursprüngliche  
Problem widersprüchlich und der  
Algorithmus terminiert.

Bild aus [3]

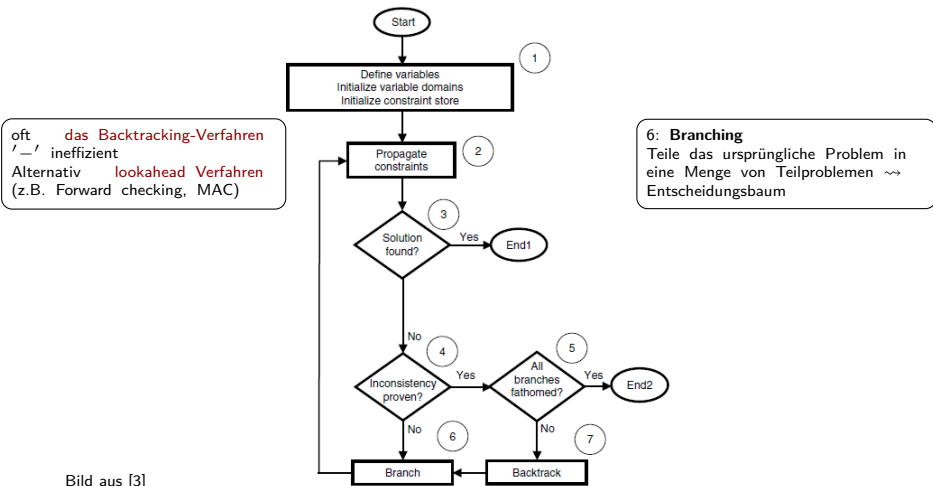
# Lösung des Constraint Satisfaction Problems



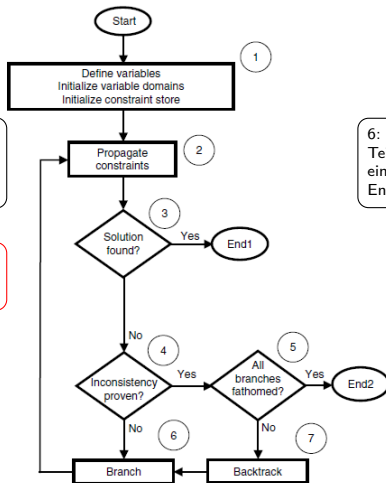
## 6: Branching

Teile das ursprüngliche Problem in eine Menge von Teilproblemen  $\rightsquigarrow$  Entscheidungsbaum

# Lösung des Constraint Satisfaction Problems



# Lösung des Constraint Satisfaction Problems



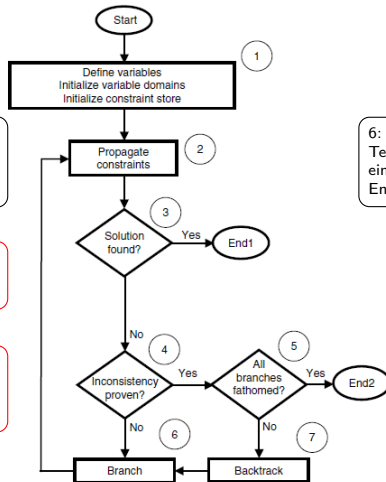
## 6: Branching

Teile das ursprüngliche Problem in eine Menge von Teilproblemen  $\rightsquigarrow$  Entscheidungsbaum

oft **das Backtracking-Verfahren**  
'-' ineffizient  
Alternativ **lookahead Verfahren**  
(z.B. Forward checking, MAC)

?: wo wird als erstes verzweigt  
z.B in die Variable mit dem  
kleinsten Definitionsbereich

# Lösung des Constraint Satisfaction Problems



## 6: Branching

Teile das ursprüngliche Problem in eine Menge von Teilproblemen  $\rightsquigarrow$  Entscheidungsbaum

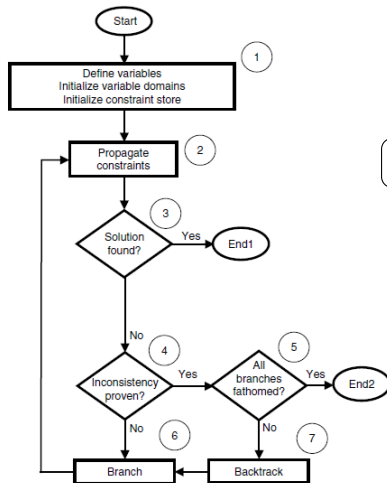
oft **das Backtracking-Verfahren**  
'-' ineffizient  
Alternativ **lookahead Verfahren**  
(z.B. Forward checking, MAC)

?: wo wird als erstes verzweigt  
z.B in die Variable mit dem  
kleinsten Definitionsbereich

?:welchen Wert soll die verzweigte  
Variable annehmen  
Z.B den kleinsten Wert aus dem  
Definitionsbereich

Bild aus [3]

# Lösung des Constraint Satisfaction Problems



7: Wähle nächstes Teilproblem und versuche es zu lösen

Bild aus [3]

# Constraint Optimization Problem

Constraint Optimization Problem (COP) kann einfach aus dem CSP erweitert werden:

- Angenommen unser COP hat eine Zielfunktion  $Z$ , die minimiert werden soll.
- Ohne  $Z$  haben wir ein CSP und seine Lösung können wir mit Hilfe des Algorithmus finden.

# Constraint Optimization Problem

**Constraint Optimization Problem (COP)** kann einfach aus dem CSP erweitert werden:

- Angenommen unser COP hat eine Zielfunktion  $Z$ , die minimiert werden soll.
- Ohne  $Z$  haben wir ein CSP und seine Lösung können wir mit Hilfe des Algorithmus finden.
- Sobald eine Lösung des CSP gefunden wurde, berechne den Wert der Zielfunktion  $Z_0$  und füge die Bedingung  $Z < Z_0$  zum Problem hinzu
- Löse neues CSP und so weiter, bis es zu einem Widerspruch kommt und alle Zweige im Suchbaum untersucht wurden.



# Constraint Optimization Problem

**Constraint Optimization Problem (COP)** kann einfach aus dem CSP erweitert werden:

- Angenommen unser COP hat eine Zielfunktion  $Z$ , die minimiert werden soll.
- Ohne  $Z$  haben wir ein CSP und seine Lösung können wir mit Hilfe des Algorithmus finden.
- Sobald eine Lösung des CSP gefunden wurde, berechne den Wert der Zielfunktion  $Z_0$  und füge die Bedingung  $Z < Z_0$  zum Problem hinzu
- Löse neues CSP und so weiter, bis es zu einem Widerspruch kommt und alle Zweige im Suchbaum untersucht wurden.
- Die zuletzt gefundene zulässige Lösung ist somit die Lösung des COP.

# Vorteile der Constraint Programmierung

- CP ist besonders für ganzzahlige Variablen geeignet.
- CP passt am besten zum Lösen von Problemen mit vielen Nebenbedingungen.
- CP ist mehr effektiv in Fällen vieler Nebenbedingungen mit wenigen Variablen in jedem einzelnen Constraint.
- Der Ansatz von CP lohnt sich besonders für Probleme, die sich als eine optimale Abbildung einer geordneten Menge in eine andere Menge darstellen lassen.

# Vorteile der Constraint Programmierung

- Analog zu den Programmiersprachen ist die Benutzung einer Variablen als Indizes von anderen Variablen erlaubt  
     $\leadsto$  Reduktion der Anzahl der Entscheidungsvariablen

Z.B. Berechnung der Kosten einer Jobreihenfolge

**IP:**  $n(n-1)$  Variablen  
 $y[i, j] \in \{0, 1\}$  1, falls Job  $i$  ein direkter Vorgänger des Jobs  $j$  ist  
Gesamter Preis der Jobreihenfolge :  
$$\sum_{i, j, i \neq j} \text{cost}[i, j] y[i, j]$$

**CP:**  $n$  Variablen  
 $\text{job}[k]$  = Index des  $k$ -ten Jobs in der Reihenfolge  
Gesamter Preis der Jobreihenfolge :  
$$\sum_{k > 1} \text{cost}[\text{job}[k-1], \text{job}[k]]$$

## • Einfachere Notation der Nebenbedingungen

Angenommen wir haben ein Problem mit zwei Maschinen. Die Variablen *MaschineA* und *MaschineB* können den Wert 0 haben, falls es keinen Job gibt, der auf der Maschine abgearbeitet werden muss, oder 1 ansonsten. Wir suchen nach einer solchen Zuordnung der Jobs zu diesen Maschinen, so dass nur eine Maschine in Betrieb sein kann.

IP:  $MaschineA + MaschineB = 1$

CP:  $MaschineA \neq MaschineB$

## • Einfachere Notation der Nebenbedingungen

Angenommen wir haben ein Problem mit zwei Maschinen. Die Variablen *MaschineA* und *MaschineB* können den Wert 0 haben, falls es keinen Job gibt, der auf der Maschine abgearbeitet werden muss, oder 1 ansonsten. Wir suchen nach einer solchen Zuordnung der Jobs zu diesen Maschinen, so dass nur eine Maschine in Betrieb sein kann.

$$\text{IP: } \textit{MaschineA} + \textit{MaschineB} = 1$$

$$\text{CP: } \textit{MaschineA} \neq \textit{MaschineB}$$

Seien die Variablen *MaschineA* und *MaschineB* ganzzahlig und ihre Werte sind die Nummern der Jobs, die auf diesen Maschinen abgearbeitet werden.

Die Bedingung lautet „ein Job kann nicht auf zwei Maschinen abgearbeitet werden“.

$$\begin{aligned} \text{IP:} \\ (\textit{MaschineA} - \textit{MaschineB} - \epsilon + \sigma \textit{BigM} \geq 0) \\ (\textit{MaschineB} - \textit{MaschineA} - \epsilon + (1 - \sigma) \textit{BigM} \geq 0) \\ \sigma \in 0, 1 \end{aligned}$$

$$\begin{aligned} \text{CP:} \\ \textit{MaschineA} \neq \textit{MaschineB} \end{aligned}$$

## • Einfachere Notation der Nebenbedingungen

Angenommen wir haben ein Problem mit zwei Maschinen. Die Variablen *MaschineA* und *MaschineB* können den Wert 0 haben, falls es keinen Job gibt, der auf der Maschine abgearbeitet werden muss, oder 1 ansonsten. Wir suchen nach einer solchen Zuordnung der Jobs zu diesen Maschinen, so dass nur eine Maschine in Betrieb sein kann.

$$\text{IP: } \text{MaschineA} + \text{MaschineB} = 1$$

$$\text{CP: } \text{MaschineA} \neq \text{MaschineB}$$

Seien die Variablen *MaschineA* und *MaschineB* ganzzahlig und ihre Werte sind die Nummern der Jobs, die auf diesen Maschinen abgearbeitet werden.

Die Bedingung lautet „ein Job kann nicht auf zwei Maschinen abgearbeitet werden“.

$$\begin{aligned} \text{IP:} \\ (\text{MaschineA} - \text{MaschineB} - \epsilon + \sigma \text{BigM} \geq 0) \\ (\text{MaschineB} - \text{MaschineA} - \epsilon + (1 - \sigma) \text{BigM} \geq 0) \\ \sigma \in 0, 1 \end{aligned}$$

$$\begin{aligned} \text{CP:} \\ \text{MaschineA} \neq \text{MaschineB} \end{aligned}$$

Die Fertigstellungszeit-Bedingung: falls Job *A* und Job *B* auf einer Maschine abgearbeitet werden, dann muss die Bearbeitung von Job *A* früher fertig werden, als die Bearbeitung vom Job *B* beginnt, oder umgekehrt.

$$\begin{aligned} \text{IP:} \\ (A.start - B.end - \epsilon + \sigma \text{BigM} \geq 0) \\ (B.start - A.end - \epsilon + (1 - \sigma) \text{BigM} \geq 0) \\ \sigma \in 0, 1 \end{aligned}$$

$$\begin{aligned} \text{CP:} \\ (A.start > B.end) \text{Xor} (B.start > A.end) \end{aligned}$$

Die beiden Methoden Constraint und ganzzahlige Programmierung sind ganz unterschiedlich und lassen sich nicht miteinander vergleichen.

Die Wahl, welche von beiden Paradigmen besser zu benutzen ist, hängt deswegen in der ersten Linie von dem Problem, den Problemgrößen, den Daten, dem Model, aber auch von den Forschern und der Software, die er benutzt, ab. Außerdem es ist auch möglich beide Methoden zu kombinieren.

# Überblick

- 1 Constraint Programmierung
  - Constraint Satisfaction Problem
  - Lösung des Constraint Satisfaction Problems
  - Constraint Optimization Problem
  - Vorteile der Constraint Programmierung
- 2 Anwendung von CP zur Lösung der Scheduling Probleme
  - Minimieren der gewichteten Gesamtverspätung
  - Job Shop Scheduling
  - Timetabling



$$1 || \sum w_j T_j |$$

Es ist ein bekanntes *NP*-hartes Problem.

Es gibt einen Lösungsansatz mittels *Dynamischer Programmierung*,  
der einen pseudopolynomialen Algorithmus liefert.

Die einfachste Formulierung des Problems in CP:

- $n$  Variablen  $s_j$  mit dem Definitionsbereich  $[0 \cdots \sum_{j=1..n} p_j]$  für Startzeiten
- $n$  Variablen  $C_j$  mit dem Definitionsbereich  $[0 \cdots \sum_{j=1..n} p_j]$  für die Fertigstellungszeiten von Jobs.

$$1 || \sum w_j T_j ||$$

### CP-Formulierung

$$\begin{aligned} \min \quad & \sum_j w_j T_j \\ \text{s.t.} \quad & C_j = p_j + s_j \quad \forall j = 1..n \\ & C_j \leq s_k \vee C_k \leq s_j \quad \forall j, k = 1..n, k > j \\ & T_j = \max\{0, C_j - d_j\} \quad \forall j = 1..n \end{aligned}$$

Insgesamt:  $2n$  Variablen und  $\frac{n(n+1)}{2}$  Nebenbedingungen.

## 1 || $\sum w_j T_j$ ||

Eine andere Formulierung ist möglich:

- Ersetze  $n$  Variablen  $s$  durch  $n$  Variablen  $pos$  mit den Definitionsbereichen  $[1..n]$ , wobei  $pos[j]$  die Position des Jobs  $j$  in der Bearbeitungsreihenfolge ist.

$$pos_j \neq pos_k \quad \forall j, k = 1..n, k > j$$

$$pos_j > pos_k \Leftrightarrow C_j \geq C_k + p_j \quad \forall j, k = 1..n, k \neq j$$

$$pos_j = 1 \Rightarrow C_j = p_j \quad \forall j = 1..n$$

Insgesamt:  $2n$  Variablen und  $\frac{3n^2-n}{2}$  Nebenbedingungen, aber die Optimierung ist viel langsamer

## 1 || $\sum w_j T_j$ IV

Weitere Bedingungen:

Wird ein Job auf der Position  $j$  platziert, ist die untere Schranke seiner Fertigstellungszeit gleich der Summe seiner Bearbeitungszeit und der Summe  $p_i$  von  $j - 1$  Jobs mit den kleinsten Bearbeitungszeiten.

$$pos_j = k \wedge j \leq k \quad \Rightarrow \quad C_j \geq \sum_{l \leq k} p_l$$

$$pos_j = k \wedge j > k \quad \Rightarrow \quad C_j \geq p_j + \sum_{l < k} p_l$$

Mögliche Heuristik für das Suchverfahren WMDD („*weighted modified due date*“)

## $Jm || C_{max}$ I

Die Aufgabe ist  $n$  Jobs zu  $m$  Maschinen zuzuordnen, wobei jeder Job  $j$  aus einer Menge von Operationen  $O_j$  besteht, die in einer bestimmten Reihenfolge abgearbeitet werden müssen.

Zur Modellierung dieses Problems kann man z.B. so genannte **disjunktive Graphen** und zur Lösung das **Branch & Bound** Verfahren verwendet.

## $Jm || C_{max} \parallel$

CP-Formulierung:

- definiere eine Variable  $s_o$  für die Startzeit der Operation  $o \in O_j$ ,  
 $s_o \in \{0, 1, \dots, C\}$ , wobei  $C_{max} < C$  ist.
- Oder besser:  $s_o \in [\sum_{o' \in pred(o)} p_{o'} \dots C - p_o - \sum_{o' \in succ(o)} p_{o'}]$ ,  
wobei  $pred(o)$  der Vorgänger und  $succ(o)$  ein Nachfolger von einer Operation  $o$  sind.

### Die Nebenbedingungen

$$s_o + p_o \leq s_{o'} \quad o, o' \in O_j, \quad o' \in succ(o), \quad j = 1 \dots n$$

$$s_o + p_o \leq C \quad o \in O$$

$$s_o + p_o \leq s_{o'} \vee s_{o'} + p_{o'} \leq s_o \quad o, o' \in O, \quad o \neq o', \quad m_o = m_{o'}$$

wobei  $O$  die Menge aller Operationen bezeichnet.

## $Jm||C_{max}$ III

Wie kann man die Bedingungsfortpflanzung für dieses Modell effektiv durchführen?

Definiere für jede  $o$

- die frühest mögliche Startzeit  $est(o)$
- die spät mögliche Startzeit  $lst(o)$

Aus  $s_o + p_o \leq s_{o'}$  folgt  $s_o + p_o \leq lst(o')$  und  $est(o) + p_o \leq p_{o'}$   
 $\Rightarrow$  lösche die Werte größer als  $lst(o') - p_o$  und kleiner als  $est(o) + p_o$  aus den Definitionsbereichen von  $s_o$  und  $s_{o'}$ .

Aus  $(s_o + p_o \leq s_{o'}) \vee (s_{o'} + p_{o'} \leq s_o)$  folgt  
 $(s_o \leq lst(o') - p_o) \vee (s_o \geq est(o') + p_{o'})$   
 $\Rightarrow$  falls  $lst(o') - p_o + 1 \leq est(o') + p_{o'} - 1$  gilt, lösche  
 $[lst(o') - p_o + 1 \dots est(o') + p_{o'} - 1]$  aus dem Definitionsbereich von  $s_o$ .

# Timetabling I

## Aufgabe

Einen Spielplan für 9 Basketball-Teams erstellen. Es sollen 18 Termine festgelegt werden, sodass an jedem Termin 8 Teams spielen und ein Team nicht spielt (d.h. hat „bye“). Alle Termine sollten in neun Wochen mit einem Spiel am Wochenende und einem Spiel am Arbeitstag pro Woche geplant werden und zusätzliche Wünsche von Teams erfüllen.



# Timetabling II

Das Problem wird in drei Phasen gelöst:

- Generierung von einem Muster (engl. pattern) für jedes Team, das für jeden Termin feststellt, ob das Team ein Heim-, Auswärtsspiel oder „bye“ spielen kann.
- Suche nach einer Menge von 9 Mustern, die die Erstellung eines Zeitplanes ermöglichen.
- Generierung eines Zeitplanes, indem man anhand des Zeitplans die Zuordnung von Spielen zu den Teams und Paare von Gegnern bestimmt.

# Timetabling III

## Phase 1: Pattern-Generierung (CSP)

Binäre Variablen  $h_j, a_j, b_j, j = 1 \dots 18$ , für Heimspiele, Auswärtsspiele und byes

- jedes Team hat nur ein Spiel an jedem Termin:  $h_j + a_j + b_j = 1$
- s.g. Mirroring: Termine sind paarweise in die vorgegebene Menge  $m$  gruppiert, jedes Paar bezeichnet Spieltermine der gleichen Teams:  
 $h_j = a_{j'}, a_j = h_{j'}, b_j = b_{j'} (j, j') \in m$
- die beiden letzten Spielen können nicht beide Auswärtsspiele sein:  
 $a_{17} + a_{18} = 2$
- Es sind nicht mehr als zwei Auswärtsspiele nacheinander erlaubt:  
 $a_j + a_{j+1} + a_{j+2} < 3$
- Es sind nicht mehr als zwei Heimspiele nacheinander erlaubt:  
 $h_j + h_{j+1} + h_{j+2} < 3$

## Timetabling IV

- Es sind nicht mehr als drei Auswärtsspiele oder byes nacheinander erlaubt:

$$a_j + b_j + a_{j+1} + b_{j+1} + a_{j+2} + b_{j+2} + a_{j+3} + b_{j+3} < 4$$

- Es sind nicht mehr als vier Heimspiele oder byes nacheinander erlaubt:

$$h_j + b_j + h_{j+1} + b_{j+1} + h_{j+2} + b_{j+2} + h_{j+3} + b_{j+3} + h_{j+4} + b_{j+4} < 5$$

- An allen Wochenenden spielt jedes Team vier Heim-, vier Auswärtsspiele und ein bye:  $\sum_{j \in \{2,4,\dots,18\}} h_j = 4$ ,

$$\sum_{j \in \{2,4,\dots,18\}} a_j = 4, \sum_{j \in \{2,4,\dots,18\}} b_j = 1$$

- Jedes Team spielt ein Heimspiel oder bye mindestens an zwei der ersten fünf Wochenenden  $\sum_{j \in \{2,4,6,8,10\}} (h_j + b_j) \geq 2$

# Timetabling V

## Phase 2: Menge von Mustern (CSP)

$$\begin{aligned} \sum_i x_i &= 9 \\ \sum_i h_{ij} x_i &= 4, \quad \sum_i s_{ij} x_i = 4, \quad \sum_i b_{ij} x_i = 1 \quad j = 1 \dots 18 \\ x_i + x_{i'} &< 1 \quad j = 1 \dots 18, \quad i, i' = 1 \dots 9, \quad i \neq i', \\ &\quad (h_{ij} = 0 \vee a_{i'j} = 0) \wedge (a_{ij} = 0 \vee h_{i'j} = 0) \end{aligned}$$

# Timetabling VI

## Phase 3: Erstellung von einem zulässigen Zeitplan

Ordne erst jedes Team zu einem von 9 vorhandenen Mustern zu und danach suche es für jedes Team nach den möglichen Gegnern an jedem Termin.

- formuliere die Aufgabe als ein CP Modell, das jeweils für jedes Team und jeden Termin einen Gegner und eine Spielart (Heim-/Auswärtsspiel oder bye) berechnet
- Die Nebenbedingungen werden von allgemeinen Bedingungen an ein Rundenturnier und Bedingungen aus Phase 1 bestimmt.

## Timetabling VII

Phase	IP SPARCstation 20	CP Pentium II, 233 MHz 64 MB RAM	Ergebnis der Phase
	CPLEX 4.0	Friar Tuck	
1	nicht gegeben	0.44 sec	38 Mustern
2	$< 1 \text{ min}$	„Sekunden“	17 Mengen von 9 Mustern
3	24 Stunden	53.7sec	179 Zeitpläne

## References I

- [1] S. C. Brailsford, C. N. Potts, and B. M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operation Research*, 119:557 – 581, 1999.
- [2] M. Henz. Scheduling a major college basketball conference. *Operations Research*, 49:163–168, 2002.
- [3] J. J. Kanet, S. L. Ahire, and M. F. Groman. Constraint programming for scheduling. In J. Y. T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Taylor & Francis, 2004.
- [4] R. Smullyan. *The Lady or the Tiger?: and Other Logic Puzzles, including a mathematical novel that features Gödel's great discovery*. Times Books, 1982.
- [5] Jannik Strötgen. Beamer theme, 2013.

The End

Vielen Dank für Ihre Aufmerksamkeit!



