

Intelligente Handlungsplanung

5. Scheduling

Prof. Dr. Susanne Biundo-Stephan

Institut für Künstliche Intelligenz, Universität Ulm

SS 2008



Inhalt

- 1 Planen versus Scheduling
- 2 Exkurs: Constraint Satisfaction
- 3 Scheduling Constraints
- 4 CSP-basiertes Scheduling

Planen vs. Scheduling 1

PLANEN

gegeben:

Planspezifikation, z.B. Start- und Zielzustandsbeschreibung

Aufgabe:

Synthese eines Planes, der die Spezifikation erfüllt

- ⇒ Finde geeignete Aktionen
- ⇒ Finde passende Anordnung dieser Aktionen

Ergebnis:

... ein *ausführbarer, korrekter* Plan

Planen vs. Scheduling 2

SCHEDULING

gegeben:

- Menge von Aktionen oder Aufgaben
- Ressourcen
- Constraints

Aufgabe:

Konstruktion eines *Ablaufplans*

- ⇒ weise jeder Aktion die erforderlichen Ressourcen zu
- ⇒ finde eine Anordnung der Aktionen, die alle Constraints erfüllt und die Ressourcen möglichst optimal nutzt

Ergebnis:

... ein *ausführbarer* Ablaufplan

Charakterisierung 1

Schedulingprobleme entstehen durch Veränderungen in der jeweiligen Ressourcensituation.

Scheduling:

... ein Entscheidungsprozess

⇒ Bestimmung der Ausführungsreihenfolge von Aktionen unter den dynamisch gegebenen Bedingungen

- Menge der Aktionen/Tasks/Jobs
- Art und verfügbare Menge der benötigten Ressourcen
- CONSTRAINTS
 - ▶ Liefertermine
 - ▶ Vorgangsdauern
 - ▶ Transfer- und Rüstzeiten
 - ▶ Verfügbarkeit

Charakterisierung 2

Die *Constraints* bestimmen den Lösungsraum.

Ansatz:

Scheduling als Wechselspiel von *Constraint-Fixierung* und
-Propagierung

⇒ Constraint Satisfaction

Exkurs: Constraint Satisfaction

Definition:

Ein *Constraint Satisfaction Problem* (CSP) umfasst

- eine endliche Menge von Variablen V
- zu jeder Variablen $v \in V$ einen endlichen Wertebereich $D(v)$ (Domain)
- eine endliche Menge von *Constraints*, C .
Constraints sind Formeln über V .
Ein Constraint heißt n -stellig gdw. es von n Variablen abhängt.
 $V(c)$ sind die Variablen, die in einem Constraint c vorkommen.

Binäre Constraints enthalten nur ein- und zweistellige Constraints.

Aufgabe:

Finde für jedes $v \in V$ ein $w \in D(v)$, so dass alle Constraints erfüllt sind.

Constraint Satisfaction 2

Wertebereiche sind *diskret* oder *kontinuierlich*, *endlich* oder *unendlich*.

Constraints sind *absolut* oder *preferentiell*.

- einstellige Constraints schränken den Wertebereich von Variablen ein
 - ⇒ bestimmen Teilmengen der Wertebereiche
- zweistellige Constraints beschreiben Beziehungen zwischen Variablen
- Constraints höherer Ordnung involvieren mehrere Variablen

Bei diskreten, endlichen Wertebereichen können die Constraints durch Aufzählung aller zulässigen Wertekombinationen dargestellt werden.

CSP sind i.a. NP-vollständig.

Constraint Satisfaction 3

Definitionen:

Sei $V = \{x_1 \dots x_n\} \subseteq V$. Eine *V-Markierung* ist eine Menge $\{x_1 \leftarrow w_1 \dots x_n \leftarrow w_n\}$ mit $w_i \in D(x_i)$ für $i \in \{1, \dots, n\}$.

Eine solche Markierung ist *konsistent* gdw. für jedes $c \in C$ die Ersetzung der Variablen x_i durch w_i für $i \in \{1, \dots, n\}$ zu einer wahren Bedingung führt.

Eine konsistente V-Markierung ist eine *Lösung* des entsprechenden Constraint Problems.

Repräsentation:

Für eine Variablenmenge V und eine Menge C von Constraints repräsentiert ein *Constraint-Netz* die wechselseitigen Beziehungen in Form eines Hypergraphen.

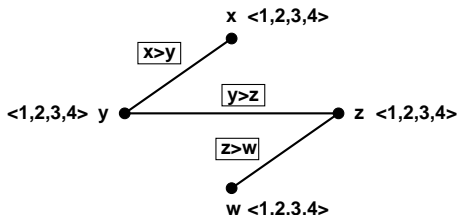
Constraint Netze

In einem Constraint-Netz entsprechen die Knoten den Variablen und ihren möglichen Belegungen. Die Kanten repräsentieren die Constraints.

Ein Constraint-Netz heißt *zusammenhängend* gdw. jedes Paar von Knoten durch einen Pfad verbunden ist.

Beispiel:

Für das Ungleichungssystem $x > y$, $y > z$, $z > w$ über der Variablenmenge $\{x, y, z, w\}$ und dem gemeinsamen Wertebereich $\{1, 2, 3, 4\}$ ist das Constraint-Netz:



Lokale Konsistenz 1

Knotenkonsistenz:

Ein Knoten x ist *knotenkonsistent* gdw. jedes einstellige Constraint $C(x)$ von jeder Belegung aus dem aktuellen Wertebereich für x erfüllt wird.

Kantenkonsistenz:

Ein Knoten x ist *kantenkonsistent* gdw. für jedes Element aus dem aktuellen Wertebereich für x gilt: Im aktuellen Wertebereich eines jeden Knoten y , zu dem x in Relation steht, gibt es ein Element, so dass das entsprechende Constraint erfüllt ist.

Lokale Konsistenz:

Ein Constraint-Netz ist *lokal konsistent*, wenn alle Knoten knoten- und kantenkonsistent sind.

Lokale Konsistenz wird erreicht, indem sukzessive alle Werte eliminiert werden, die ein Constraint verletzen.

Lokale Konsistenz 2

Beispiel:

Lokal konsistente Netze schließen von vornherein solche Variablenbelegungen aus, die einzelne Constraints verletzen.

⇒ Einschränkung des Suchraums

Die Herstellung lokaler Konsistenz führt in manchen Fällen bereits zur Lösung des Problems.

Im allgemeinen ist sie jedoch ein Vorverarbeitungsschritt und wird im Kontext systematischer Constraint Suche eingesetzt.

Beispiel:

Constraint Propagierung 1

Definition:

Constraint Propagierung bezeichnet den Prozess der sukzessiven Einschränkung des Suchraumes: Die Effekte lokaler Einschränkungen im Constraint-Netz manifestieren sich als neue lokale Constraints und breiten sich aus.

Herstellung von Knotenkonsistenz:

- prüfe für jede Variable ihren Wertebereich auf Verletzung unärer Constraints
- entferne entsprechende Werte aus dem Wertebereich

Laufzeit: $O(n \times m)$

n: # Variablen

m: # Werte

Knotenkonsistenz wird in *einem* Durchlauf über alle Knoten erreicht. Anschließend können alle unären Constraints aus dem Netz entfernt werden.

Constraint Propagierung 2

Herstellung von Kantenkonsistenz:

- prüfe für jedes Variablenpaar (x, y) , ob die Werte in den entsprechenden Wertebereichen kompatibel sind bzgl. der betroffenen Constraints
- entferne inkompatible Werte aus dem Wertebereich
- wende den Algorithmus auf das Variablenpaar (y, x) an
- wende den Algorithmus solange an, bis sich keine Änderung der Wertebereiche mehr ergibt

Laufzeit: $O(n^2)$

Constraint Propagierung 3

In manchen Fällen löst die Herstellung der Kantenkonsistenz das CSP bereits.

- Alle Wertebereiche werden auf genau ein Element reduziert
⇒ eine passende Belegung für jede der Variablen
- Der Wertebereich einer Variablen wird auf die leere Menge reduziert
⇒ es gibt keine Lösung.

Ein Constraint Solving Algorithmus

Systematische Suche durch chronologisches Backtracking

CS-CBT(CN)

Eingabe: CN: Constraint-Netz mit Variablen $x_1 \dots x_n$
und Wertebereichen $D_1 \dots D_n$

Initialisierung: $A := \emptyset$, $M := \{x_1 \dots x_n\}$, $S_i := D_i \quad i \in \{1 \dots n\}$

while $M \neq \emptyset$ do

choose $x_i \in M \quad M := M - \{x_i\}$

choose $d_i \in S_i \quad S_i := S_i - \{d_i\}$

push $((x_i, d_i), A)$

if *inconsistent*(A) then repeat until *cons*

$cons := false$

if $A = \emptyset$ then return(fail)

$(x_j, d_j) := pop(A)$

$M := M \cup \{x_j\}$

if $S_j \neq \emptyset$ then $cons := true$

else $S_j := D_j$

return(A)

Heuristiken 1

*In welcher Reihenfolge werden die Variablen instanziiert?
Welche Werte werden gewählt?*

Lösung:

Ordnungsheuristiken für Variablen und Werte
statische vs. dynamische Ordnungsheuristiken

Variablen-Ordnungsheuristiken:

- *Maximum-Degree*
größtmögliche Einschränkung des Suchraumes
wähle die Variablen mit *maximalem Verbindungsgrad* im Netz
⇒ ordnet nach Verbindungsgrad

Variablen-Ordnungsheuristiken (Forts.):

- *Maximum-Cardinality*
wähle die erste zu instanziiierende Variable zufällig aus
wähle danach die Variablen mit maximalem Verbindungsgrad
bzgl. der bereits instanziierten Variablen
⇒ ähnlich wie Maximum-Degree
- *Dynamic-search-Rearrangement*
wähle jeweils die Variable, die in ihrem aktuellen Wertebereich die
wenigsten Werte hat, die mit der aktuellen Teillösung kompatibel
sind

Heuristiken 3

Werte-Ordnungsheuristiken:

Ziel: möglichst viele Optionen offen halten

- wähle einen Wert, der mit möglichst vielen Werten noch nicht instanzierter Variablen kompatibel ist
- *Min-Conflict*
beginne mit einer zufällig gewählten, vollständigen Instanziierung
ändere den Wert einer Variablen mit inkompatibler Belegung so, dass er mit möglichst vielen Werten anderer Variablen kompatibel ist
⇒ lokale Suche

Verfahren, durch die inkompatible Instanziierungen ausgeschlossen werden können

⇒ lokale Konsistenz

Suchraumreduktion

Möglichkeit:

Herstellung lokaler Konsistenz nach jedem Instanziierungsschritt

Häufig:

Aufwand höher als Gewinn

Daher:

Reduktion des Aufwandes durch vereinfachte Form des Kantenkonsistenzalgorithmus

⇒ **Forward Checking:**

Für alle noch nicht instanziierten Variablen werden aus den Wertebereichen diejenigen Werte eliminiert, die mit dem letzten Instanziierungsschritt inkompatibel sind.

Charakterisierung 3

Zusätzlich:

Relaxierbare Präferenzconstraints bestimmen die Qualität der Scheduling-Entscheidungen

- Fertigstellungszeitpunkt
- Produktivität
- Häufigkeit von Umrüstungen
- Lagerbestände
- Betriebsstabilität

⇒ Prioritätsrelationen betreffen

- Herstellungsreihenfolge
- Produktionsverteilung
- Herstellungsprozeduren
- alternative Ressourcen

i.d.R. konfliktieren diese Präferenzconstraints

Charakterisierung 4

- ⇒ Das Schedulingproblem umfasst zusätzlich die Entscheidung, welche der Präferenzconstraints erfüllt werden und welche (wie weit) relaxiert werden können.

Präferenzconstraints 1

Behandlung von Präferenzconstraints

- 1 Kombination von Präferenzen in einer einheitlichen Evaluations- oder Kostenfunktion
 - ⇒ Das System erzeugt einem optimalen Schedule im Sinne dieser Evaluationsfunktion.
- 2 Übersetzung (Kompilierung) der Präferenzen in Evaluations*heuristiken*, die solche möglichen Lösungen bevorzugen, die (mit hoher Wahrscheinlichkeit) die entsprechenden Präferenzen erfüllen.
 - ⇒ Das System versucht, einen Schedule zu erzeugen, der die entsprechenden Präferenzen erfüllt, läßt jedoch den (globalen) Optimalitätsaspekt außer Acht.

Beide Möglichkeiten sind nicht wirklich befriedigend.

Präferenzconstraints 2

- Finden einer Evaluationsfunktion ist schwierig, insbesondere im Hinblick auf die sich dynamisch ändernden Gegebenheiten in der Produktionsumgebung
- Ohne einheitliche Evaluierungsfunktion fehlt andererseits die Möglichkeit, die Qualität eines erzeugten Schedules tatsächlich abzuschätzen.

Daher:

Die Komplexität von Schedulingproblemen legt nahe, Optimierung durch iterative Verbesserung immer besserer Lösungen zu implementieren:

geg: Lösung S_i zu einem Schedulingproblem P_i

erzeuge: Schedulingproblem P_{i+1}

ges: Lösung S_{i+1} mit geringeren Kosten als S_i

repeat until P_{n+1} ist unlösbar, $1 \leq i \leq n$

Dann ist S_n optimaler Schedule.

Scheduling als CSP 1

Unter der Annahme, dass absolute Optimierung nicht erforderlich ist, bzw. iterativ angenähert wird, kann Scheduling als ein CSP aufgefasst werden.

Lösung:

Eine Menge kompatibler *Fixierungen*, die die Erfüllung der Constraints garantieren.

- ⇒ Ziel einer jeden der sukzessiv erfolgenden Fixierungen ist die Erfüllung einiger Constraints so, dass keine Widersprüche zu anderen Constraints oder Fixierungen entstehen.
- ⇒ Jede Fixierung kann als ein neues Constraint aufgefasst werden.

$$P_0 \rightarrow P_0 \cup \{d\}$$

Schedulingproblem P_n : Menge kompatibler Constraints, so dass jede "Ausführung" gemäß der Fixierungen in P_n jedes Constraint in P_n erfüllt.

Scheduling als CSP 2

Voraussetzungen:

- Constraintsprache L
- Teilsprache $S \subseteq L$ zur Darstellung der Fixierungen. Constraints über S sind ausführbar.

Ziel des Fixierungsprozesses ist die schrittweise Verfeinerung des Schedulingproblems durch das Hinzufügen neuer Constraints so, dass schließlich alle (ursprünglich vorhandenen Constraints aus denjenigen in S folgen.

Vorteil:

verschiedene Arten von Fixierungen
verschiedene Arten der Intervention

Scheduling als CSP 3

Frage 1: Kompatibilität der Fixierungen

⇒ *Constraintpropagierung*

- Erkennung aller möglichen Inkompatibilitäten zwischen Fixierungen, um sicherzustellen, dass nur zulässige Schedules erzeugt werden
- Berücksichtigung der Auswirkungen von Fixierungen und anderen Constraints bei der Entscheidung

Frage 2: Reihenfolge der Fixierungen

- *mikro-opportunistisch*: Jeder ressourcenspezifische Modul macht eine Fixierung pro Aufruf

Steuerung:

- ▶ Welche Art Fixierung in welcher Phase
- ▶ Ausmaß der Propagierung

Frage 2: Reihenfolge der Fixierungen (Forts.)

- *makro-opportunistisch*: Jeder Modul macht eine *Anzahl* von Fixierungen

Steuerung:

- ▶ Anordnung der Aufrufe der Module
- ▶ Internes Verhalten der Module

Scheduling als CSP 5

Constraintpropagierung:

Aussagen werden als Variablenrestriktionen aufgefasst.

Angabe von Wertebereichen für Variablen.

z.B. $x \in \{2, 4, 8, 16\}$ $y \in \{2, 4, 8, 16, 32, 64\}$

Angabe von Constraints:

Spezifikation von Teilmengen des kartesischen Produkts der Wertebereiche

z.B. $(x + 1) \geq 2y^2$

Darstellung von Variablen und Constraints als Hypergraphen

- Knoten \equiv Variablen mit assoziierten Wertebereichen
- Kanten \equiv Constraints

Scheduling als CSP 6

Beispiel:

Propagierung von Constraints unterschiedlicher Art

- Intervallarithmetik
- Boole'sche Ausdrücke
- Euklidische Geometrie

Propagierung

- ⇒ Berechnung von Variablenwerten
- ⇒ Berechnung von Variablenrelationen

Allgemein:

Constraint Propagierung als Komponente im Problemlösungsprozess

Scheduling als CSP 7

Dadurch:

- Dekomposition des Problems
- Aufrechterhaltung von Abhängigkeiten zwischen Teilproblemen
- Bestimmung der restiktiertesten Teilprobleme
- Steuerung der Problemlösung

Constraint Propagierung:

- Ableitung neuer Constraints
- Aufdeckung von Inkonsistenzen

⇒ Scheduling als Kombination von Constraint-Fixierung und -Propagierung

Scheduling als CSP 8

Blackboard - Architektur

- *Wissensquellen*: Problemlösung, Constraint Propagierung
- *Blackboard-Datenstruktur*: Globale Datenbasis, die den aktuellen Zustand des Problemlöseprozesses hält.

Wissensquellen verändern diesen Zustand schrittweise.

Wissensquellen kommunizieren (ausschließlich) über das Blackboard.

- *Kontrolle*: Das eigentliche Modell besitzt keine Kontrollstruktur.

Wissensquellen sind selbstaktivierend; sie reagieren opportunistisch auf Veränderungen im Blackboard.

Jedoch:

I.d.R. entscheidet eine Kontrollkomponente, welche Wissensquellen in welcher Situation des Problemlösungsprozesses ausgeführt werden.

siehe Mitschrift

Eigenschaften 1

Ein CSP-basiertes Schedulingssystem ist *korrekt* gdw.

- das System nur dann in einem *Konfliktzustand* terminiert, wenn die initiale Constraintmenge inkonsistent ist;
- das System nur dann in einem *konfliktfreien* Zustand terminiert, wenn das initiale Constraintsystem konsistent ist.

Es ist *vollständig* für eine Menge S von Constraints gdw. es in einem Zustand terminiert, der entweder einen Konflikt enthält oder für jedes initiale Constraint c_i ein Constraint $s_j \in S$ auf dem Blackboard enthält, so dass die Erfüllung von s_j diejenige von c_i garantiert.

Eigenschaften 2

Theorem:

Ein CSP-basiertes Schedulingssystem ist korrekt und vollständig für eine Constraintmenge S , falls gilt:

- 1 Das Constraint-Propagierungsverfahren ist korrekt und widerspruchsvollständig für S .
- 2 Die Wissensquellen eliminieren keine initialen Constraints.
- 3 Wenn ein Constraint c auf dem Blackboard erscheint, das nicht aus einem Constraint s , das sowohl im Blackboard als auch in S vorkommt, folgt, so ist wenigstens eine Wissensquelle in der Lage, s zu erzeugen.
- 4 Wenn ein Konflikt über einer Constraintmenge $\{s_1, \dots, s_n\}$, die $\{c_1, \dots, c_n\}$ lösen soll, vorkommt, kann entweder eine Wissensquelle ein s_i durch t_i ersetzen (wobei c_i aus t_i folgt) oder die initiale Constraintmenge ist inkonsistent.
- 5 Die Kontrollstrategie garantiert die Terminierung.