Constraintbasierte Stundenplanung für Universitäten

Slim Abdennadher und Michael Marte

Institut für Informatik
Ludwig-Maximilians-Universität München

Aufgabenstellung

- Aufgabe: Erstellung eines Stundenplans für das kommende Semester auf Basis des Vorjahresplans unter Berücksichtigung von
  - Informationen über Veränderungen im Lehrangebot,
  - Lehrerwünschen und
  - allgemeinen Anforderungen, z.B. Überschneidungsfreiheit im Grund- und Hauptstudium.
- Maxime: *So wenig wie möglich ändern!*
  Konservatives Vorgehen vereinfacht Planung und sichert Akzeptanz des neuen Plans.
- Probleme:
  - Aufgabe ist komplex:
    Planung von Hand dauert mindestens zwei Tage.
    → Ziel: Reduktion auf wenige Minuten
  - Nicht alle Anforderungen können erfüllt werden.
    → Ansatz: Gewichtung der Anforderungen

Constraintprogrammierung

- Zentrale Ideen:
  - Deklarative Problemmodellierung durch Angabe
    - der Variablen und
    - Relationen zwischen Variablen (*Constraints*), z.B. $x \in \{2, 3, 4\}$ oder $x < y$.
  - *Constraintlöser* (spezielle, effiziente Verfahren) vereinfachen Problem, z.B. $x \in \{2, 3, 4\} \wedge x \in \{3, 4, 5\} \rightarrow x \in \{3, 4\}$.
- Constraintlösen liefert im allgemeinen keine Lösung: Kombination mit Suche ist notwendig.
- Ansatz ist flexibel: Constraintlöser können gut kombiniert werden mit weiteren Constraintlösern und Such- und Optimierungsmethoden.

Modellierung als Partial Constraint Satisfaction Problem

- Eine Variable pro Veranstaltung.
- Wertebereich ist die ganze Woche.
- Constraints:
  - Überschneidungsconstraints
  - Zeitconstraints
  - Verteilungsconstraints
  - Anschlußconstraints
- Unterscheidung von *harten* und *weichen* Constraints durch Gewichte.
- Eine Lösung
  - erfüllt alle harten Constraints und
  - minimiert die Gewichtssumme der verletzten weichen Constraints.

Lösung von PCSP

- Ansätze in der Literatur:
  - Lösung für alle harten Constraints suchen,
    Optimierung mit Branch & Bound
  - Propagierung weicher Constraints berechnet
    Einschätzungen für die möglichen Werte der
    Variablen,
    Suche orientiert sich bei der Wertauswahl an
    diesen Einschätzungen
    → weiche Constraints steuern Suche
    → auch erste Lösung erfüllt schon viele weiche
    Constraints,
    Optimierung mit Branch & Bound
- Problem bei zweitem Ansatz: Verbreitete
  Constraintsprachen unterstützen keine weichen
  Constraints.
  → Ziel: Implementierung des zweiten Ansatzes mit
  der Sprache *Constraint Handling Rules*.

Constraint Handling Rules

Deklarative, regelbasierte Sprache zum Schreiben von Constraintlösern.

- Simplifikationsregeln
- Propagationsregeln

Beispiel

```
reflexivitaet  @ X ≤ X          <=> true.
antisymmetrie  @ X ≤ Y ∧ Y ≤ X  <=> X = Y.
transitivitaet @ X ≤ Y ∧ Y ≤ Z  ==> X ≤ Z.
```

$\underline{\texttt{A} \leq \texttt{B}} \wedge \underline{\texttt{C} \leq \texttt{A}} \wedge \texttt{B} \leq \texttt{C}$	(Transitivität)
$\texttt{A} \leq \texttt{B} \wedge \texttt{C} \leq \texttt{A} \wedge \underline{\texttt{B} \leq \texttt{C}} \wedge \underline{\texttt{C} \leq \texttt{B}}$	(Antisymmetrie)
$\texttt{A} \leq \texttt{B} \wedge \underline{\texttt{C} \leq \texttt{A}} \wedge \underline{\texttt{B = C}}$	(eingebaute Behandlung)
$\underline{\texttt{A} \leq \texttt{B}} \wedge \underline{\texttt{B} \leq \texttt{A}} \wedge \texttt{B = C}$	(Antisymmetrie)
$\texttt{A = B} \wedge \texttt{B = C}$	

Grundlegende Constraints

- `domain(C, S, D)`: Veranstaltung `C` beginnt zu einem Zeitpunkt `S` in `D`.
  `D` ist Liste von Paaren (Zeitpunkt, Einschätzung).
- `notin(C, L, W)`: Veranstaltung `C` darf/ soll zu keinem Zeitpunkt in `L` beginnen.
- `in(C, L, W)`: Veranstaltung `C` darf/ soll zu einem Zeitpunkt in `L` beginnen.

Alle applikationsspezifischen Constraints können auf `in` und `notin` zurückgeführt werden.

Der Constraintlöser

- Propagierung von `notin`-Constraints:

```
domain(C, S, D), notin(C, L, W) <=>
  (
     W = hard
  -> domain_subtraction(D, L, D1),
  ;  decrease_assessment(W, L, D, D1)
  ),
  domain(C, S, D1).
```

- Propagierung von `in`-Constraints: analog
- Kein Platz mehr für eine Veranstaltung:

```
domain(_, _, []) <=> fail.
```

Zusammenfassung und Ausblick

- Eigenschaften des Constraintlösers:
  - kurz und deklarativ
  - propagiert harte und weiche Constraints
  - leistungsfähig genug zur Stundenplanung für unser Institut
- Stundenplan für das Sommersemester 98 wurde mit unserem Prototypen erstellt.
  Dauer der Planung: zweieinhalb Minuten
- Weitere Projekte:
  - Integration von Optimierungsmethoden
  - Raumplanung
  - Stundenplanung für Schulen