

# Grundlagen der Künstlichen Intelligenz

## 23. Constraint-Satisfaction-Probleme: Constraint-Netze

Malte Helmert

Universität Basel

13. April 2015

# Grundlagen der Künstlichen Intelligenz

13. April 2015 — 23. Constraint-Satisfaction-Probleme: Constraint-Netze

## 23.1 Constraint-Netze

## 23.2 Beispiele

## 23.3 Belegungen und Konsistenz

## 23.4 Ausblick und Zusammenfassung

## Constraint-Satisfaction-Probleme: Überblick

### Kapitelüberblick Constraint-Satisfaction-Probleme:

- ▶ 22.–23. Einführung
  - ▶ 22. Einführung und Beispiele
  - ▶ 23. **Constraint-Netze**
- ▶ 24.–26. Kernalgorithmen
- ▶ 27.–28. Problemstruktur

## 23.1 Constraint-Netze

## Constraint-Netze: informell

### Constraint-Netze: informelle Definition

Ein **Constraint-Netz** ist definiert durch

- ▶ eine endliche Menge von **Variablen**
- ▶ einen endlichen **Wertebereich** für jede Variable
- ▶ eine Menge von **Constraints** (hier: **binäre Relationen**), denen die Variablen genügen müssen

Wir suchen eine **Lösung** für das Netz, d. h. eine Belegung der Variablen, die allen Constraints genügt.

Oft sagt man **Constraint-Satisfaction-Problem (CSP)** statt Constraint-Netz.

Im strengen Sprachgebrauch ist ein „CSP“ aber das algorithmische Problem der Lösungsfindung für Constraint-Netze.

## Constraint-Netze: formal

### Definition (binäres Constraint-Netz)

Ein (**binäres**) **Constraint-Netz** ist ein 3-Tupel

$\mathcal{C} = \langle V, \text{dom}, (R_{uv}) \rangle$  mit:

- ▶  $V$  ist eine nicht-leere, endliche Menge von **Variablen**,
- ▶  $\text{dom}$  ist eine Funktion, die jeder Variable  $v$  einen nicht-leeren endlichen **Wertebereich**  $\text{dom}(v)$  zuordnet
- ▶  $(R_{uv})_{u,v \in V, u \neq v}$  ist eine Familie von binären Relationen (**Constraints**) über den Variablen, wobei für alle  $u \neq v$  gilt:  $R_{uv} \subseteq \text{dom}(u) \times \text{dom}(v)$

### Häufige Generalisierungen:

- ▶ unendliche Wertebereiche (z.B.  $\text{dom}(v) = \mathbb{Z}$ )
- ▶ mehrstellige Constraints (z.B. Erfüllbarkeit in Aussagenlogik)

## Binäre Constraints

### Binäre Constraints:

- ▶ Wenn  $u, v$  zwei Variablen sind, drückt der Constraint  $R_{uv}$  aus, welche **gemeinsamen Belegungen** der Variablen in Lösungen erlaubt sind.
- ▶ Wenn  $R_{uv} = \text{dom}(u) \times \text{dom}(v)$ , ist der Constraint **trivial**: es besteht keine Einschränkung, und er wird bei der Beschreibung normalerweise weggelassen (ist aber formal gesehen immer vorhanden!)
- ▶ Constraints  $R_{uv}$  und  $R_{vu}$  sprechen über dieselben Variablen. Daher gibt man normalerweise nur einen der beiden an.

## Unäre Constraints

### Unäre Constraints:

- ▶ Oft ist es praktisch, auch zusätzliche Einschränkung an **einzelne** Variablen als Constraints zu notieren.
- ▶ Man spricht dann von **unären** Constraints.
- ▶ Ein unärer Constraint  $R_v$  für  $v \in V$  entspricht einfach einer Reduktion von  $\text{dom}(v)$  auf die durch  $R_v$  erlaubten Werte.
- ▶ Formal sind unäre Constraints unnötig, aber ihre Verwendung erlaubt es oft, Constraint-Netze klarer zu beschreiben.

## Kompakte Kodierung und allgemeine Constraint-Löser

Constraint-Netze bieten **kompakte Kodierungen** einer grossen Belegungsmenge:

- ▶ Betrachte ein Problem mit  $n$  Variablen, deren Wertebereiche alle jeweils  $k$  Elemente umfassen.
- ↪  $k^n$  Belegungen
- ▶ Für die **Beschreibung** als Constraint-Netz muss man maximal  $\binom{n}{2} = O(n^2)$  Constraints angeben, von denen jeder aus maximal  $k^2$  Paaren besteht
- ↪ Kodierungsgrösse  $O(n^2 k^2)$
- ▶ Die Zahl der Belegungen ist also **exponentiell grösser** als die Beschreibung des Constraint-Netzes
- ▶ Damit eignen sich solche Beschreibungen von Constraint-Netzen als Input für **allgemeine** Constraint-Löser.

## 23.2 Beispiele

### Beispiel: 4-Damen-Problem

#### 4-Damen-Problem als Constraint-Netz

- ▶ **Variablen:**  $V = \{v_1, v_2, v_3, v_4\}$   
 $v_i$  kodiert Zeile der Dame in der  $i$ -ten Spalte
- ▶ **Wertebereiche:**  
 $\text{dom}(v_1) = \text{dom}(v_2) = \text{dom}(v_3) = \text{dom}(v_4) = \{1, 2, 3, 4\}$
- ▶ **Constraints:** für alle  $1 \leq i < j \leq 4$  setzen wir:  $R_{v_i, v_j} = \{\langle k, l \rangle \in \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \mid k \neq l \wedge |k - l| \neq |i - j|\}$   
z. B.  $R_{v_1, v_3} = \{\langle 1, 2 \rangle, \langle 1, 4 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 2 \rangle, \langle 3, 4 \rangle, \langle 4, 1 \rangle, \langle 4, 3 \rangle\}$

|   | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|-------|-------|-------|-------|
| 1 |       |       |       |       |
| 2 |       |       |       |       |
| 3 |       |       |       |       |
| 4 |       |       |       |       |

### Beispiel: Sudoku

#### Sudoku als Constraint-Netz

- ▶ **Variablen:**  $V = \{v_{ij} \mid 1 \leq i, j \leq 9\}$ ;  $v_{ij}$ : Inhalt Zeile  $i$ , Spalte  $j$
- ▶ **Wertebereiche:**  $\text{dom}(v) = \{1, \dots, 9\}$  für alle  $v \in V$
- ▶ **unäre Constraints:**  $R_{v_{ij}} = \{k\}$ ,  
wenn  $\langle i, j \rangle$  vorgegebenes Feld, in dem  $k$  steht
- ▶ **binäre Constraints:** für alle  $v_{ij}, v_{i'j'} \in V$  setzen wir  
 $R_{v_{ij}, v_{i'j'}} = \{\langle a, b \rangle \in \{1, \dots, 9\} \times \{1, \dots, 9\} \mid a \neq b\}$ ,  
falls  $i = i'$  (dieselbe Zeile),  $j = j'$  (dieselbe Spalte)  
oder  $\langle \lceil \frac{i}{3} \rceil, \lceil \frac{j}{3} \rceil \rangle = \langle \lceil \frac{i'}{3} \rceil, \lceil \frac{j'}{3} \rceil \rangle$  (derselbe Block)

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 2 | 5 |   |   | 3 |   | 9 |   | 1 |
|   | 1 |   |   |   | 4 |   |   |   |
| 4 |   | 7 |   |   |   | 2 |   | 8 |
|   |   | 5 | 2 |   |   |   |   |   |
|   |   |   |   | 9 | 8 | 1 |   |   |
|   | 4 |   |   |   | 3 |   |   |   |
|   |   |   | 3 | 6 |   |   | 7 | 2 |
|   | 7 |   |   |   |   |   |   | 3 |
| 9 |   | 3 |   |   |   | 6 |   | 4 |

## 23.3 Belegungen und Konsistenz

### Belegungen

#### Definition (Belegung, partielle Belegung)

Sei  $\mathcal{C} = \langle V, \text{dom}, (R_{uv}) \rangle$  ein Constraint-Netz.

Eine **partielle Belegung** von  $\mathcal{C}$  (oder von  $V$ ) ist eine Funktion

$$\alpha : V' \rightarrow \bigcup_{v \in V} \text{dom}(v)$$

mit  $V' \subseteq V$  und  $\alpha(v) \in \text{dom}(v)$  für alle  $v \in V'$ ,

Wenn hierbei  $V' = V$  gilt, heisst  $\alpha$  auch **totale Belegung** oder einfach **Belegung**.

↪ **partielle Belegung** weist einigen oder allen Variablen Werte aus deren Wertebereich zu

↪ (totale) **Belegung** muss auf allen Variablen definiert sein

### Konsistenz

#### Definition (inkonsistent, konsistent, verletzter Constraint)

Eine partielle Belegung  $\alpha$  für ein Constraint-Netz  $\mathcal{C}$  heisst **inkonsistent**, wenn es zwei Variablen  $u, v$  gibt, für die  $\alpha$  definiert ist und für die gilt:  $\langle \alpha(u), \alpha(v) \rangle \notin R_{uv}$ .

Man sagt dann:  $\alpha$  **verletzt** den Constraint  $R_{uv}$ .

Eine partielle Belegung heisst **konsistent**, wenn sie nicht inkonsistent ist.

**triviales Beispiel:** leere Belegung ist immer konsistent

### Lösung

#### Definition (Lösung, lösbar)

Sei  $\mathcal{C}$  ein Constraint-Netz.

Eine konsistente totale Belegung für  $\mathcal{C}$  heisst **Lösung** von  $\mathcal{C}$ .

Wenn eine Lösung von  $\mathcal{C}$  existiert, heisst  $\mathcal{C}$  **lösbar**.

Wenn keine Lösung existiert, heisst  $\mathcal{C}$  **inkonsistent**.

## Konsistenz vs. Lösungseigenschaft

**Achtung:** dass eine partielle Belegung  $\alpha$  konsistent ist, heisst **nicht**, dass  $\alpha$  zu einer Lösung ergänzt werden kann.

Es heisst nur, das **bisher** (auf den Variablen, für die  $\alpha$  definiert ist) kein Constraint verletzt ist.

**Beispiel (4-Damen-Problem):**  $\alpha = \{v_1 \mapsto 1, v_2 \mapsto 4, v_3 \mapsto 2\}$

|   | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|-------|-------|-------|-------|
| 1 | q     |       |       |       |
| 2 |       |       | q     |       |
| 3 |       |       |       |       |
| 4 |       | q     |       |       |

## Komplexität von Constraint-Satisfaction-Problemen

**Satz (CSPs sind NP-vollständig)**

*Zu entscheiden, ob ein gegebenes Constraint-Netz lösbar ist, ist ein NP-vollständiges Problem.*

**Beweis.**

**Mitgliedschaft in NP:**

Eine geratene Lösung kann in polynomieller Zeit in der Eingabegrösse überprüft werden.

**NP-Härte:**

Der Spezialfall der Graphfärbbarkeit ist bereits als NP-vollständig bekannt.



## Schärfe von Constraint-Netzen

**Definition (schärfer, echt schärfer)**

Seien  $\mathcal{C} = \langle V, \text{dom}, R_{uv} \rangle$  und  $\mathcal{C}' = \langle V, \text{dom}', R'_{uv} \rangle$  Constraint-Netze mit denselben Variablen.

$\mathcal{C}$  heisst **schärfer** als  $\mathcal{C}'$ , in Symbolen  $\mathcal{C} \sqsubseteq \mathcal{C}'$ , wenn gilt:

- ▶  $\text{dom}(v) \subseteq \text{dom}'(v)$  für alle  $v \in V$
- ▶  $R_{uv} \subseteq R'_{uv}$  für alle  $u, v \in V$  (einschliesslich trivialer Constraints)

Ist mindestens eine dieser Teilmengenbeziehungen echt, dann heisst  $\mathcal{C}$  **echt schärfer** als  $\mathcal{C}'$ , im Symbolen  $\mathcal{C} \subset \mathcal{C}'$ .

englisch: tighter, strictly tighter

## Äquivalenz von Constraint-Netzen

**Definition (äquivalent)**

Seien  $\mathcal{C}$  und  $\mathcal{C}'$  Constraint-Netze mit denselben Variablen.  $\mathcal{C}$  und  $\mathcal{C}'$  heissen **äquivalent**, geschrieben  $\mathcal{C} \equiv \mathcal{C}'$ , wenn sie dieselben Lösungen besitzen.

## 23.4 Ausblick und Zusammenfassung

## CSP-Algorithmen

In den folgenden Kapiteln betrachten wir **Lösungsalgorithmen** für Constraint-Netze.

### Grundkonzepte:

- ▶ **Suche**: systematisches Ausprobieren von partiellen Belegungen
- ▶ **Backtracking**: Verwerfen inkonsistenter partieller Belegungen
- ▶ **Inferenz**: Herleiten schärferer äquivalenter Constraints, um Suchraum zu verkleinern (Backtracking früher möglich)

## Zusammenfassung

- ▶ formale Definition von **Constraint-Netzen**:  
**Variablen**, **Wertebereiche**, **Constraints**
- ▶ **kompakte Kodierungen** von exponentiell vielen Konfigurationen
- ▶ **unäre** und **binäre** Constraints
- ▶ **Belegungen**: partiell und total
- ▶ **Konsistenz** von Belegungen; **Lösungen**
- ▶ Entscheiden der Lösbarkeit ist **NP-vollständig**
- ▶ **Schärfe** von Constraints
- ▶ **Äquivalenz** von Constraints