

Grundlagen der Künstlichen Intelligenz

24. Constraint-Satisfaction-Probleme: Backtracking

Malte Helmert

Universität Basel

17. April 2015

Constraint-Satisfaction-Probleme: Überblick

Kapitelüberblick Constraint-Satisfaction-Probleme:

- 22.–23. Einführung
- 24.–26. Kernalgorithmen
 - 24. Backtracking
 - 25. Kantenkonsistenz
 - 26. Pfadkonsistenz
- 27.–28. Problemstruktur

CSP-Algorithmen

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)
↪ folgende Kapitel

Naives Backtracking

Naives Backtracking (= ohne Inferenz)

function NaiveBacktracking(\mathcal{C}, α):

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

if α is inconsistent with \mathcal{C} :

return inconsistent

if α is a total assignment:

return α

select **some variable** v for which α is not defined

for each $d \in \text{dom}(v)$ **in some order**:

$\alpha' := \alpha \cup \{v \mapsto d\}$

$\alpha'' := \text{NaiveBacktracking}(\mathcal{C}, \alpha')$

if $\alpha'' \neq \text{inconsistent}$:

return α''

return inconsistent

Eingabe: Constraint-Netz \mathcal{C} und partielle Belegung α von \mathcal{C}
(erster Aufruf: die leere Belegung $\alpha = \emptyset$)

Ergebnis: Lösung von \mathcal{C} oder **inconsistent**

Ist das ein neuer Algorithmus?

Wir haben diesen Algorithmus schon gesehen:

Backtracking entspricht Tiefensuche (vgl. Kapitel 12)

mit folgendem Zustandsraum:

- **Zustände:** konsistente partielle Belegungen
- **Anfangszustand:** leere Belegung \emptyset
- **Zielzustände:** konsistente totale Belegungen
- **Aktionen:** $assign_{v,d}$ weist Variable v Wert $d \in \text{dom}(v)$ zu
- **Aktionskosten:** alle 0 (alle Lösungen gleich gut)
- **Transitionen:**
 - für jede nicht-totale Belegung α wähle Variable $v = \text{selected}(\alpha)$, die in α unbelegt ist
 - Transition $\alpha \xrightarrow{assign_{v,d}} \alpha \cup \{v \mapsto d\}$ für alle $d \in \text{dom}(v)$

Warum Tiefensuche?

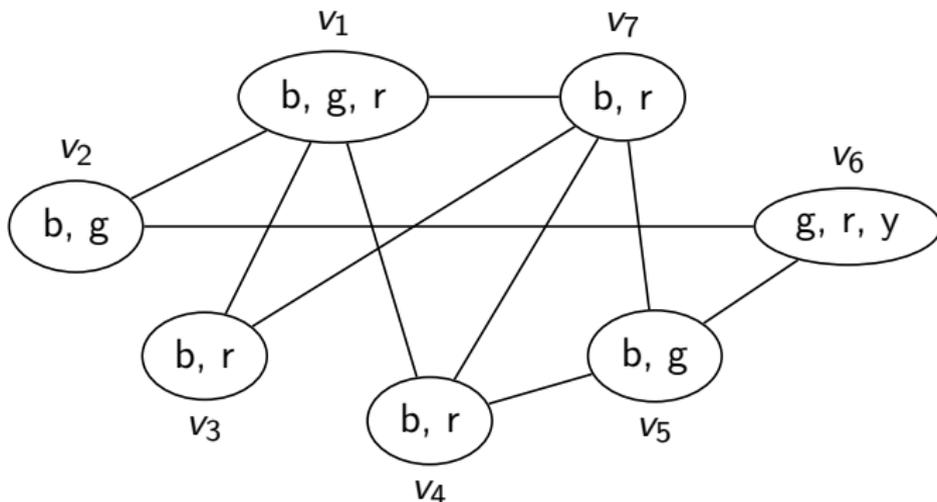
Tiefensuche ist für CSPs besonders geeignet:

- Pfadlänge **beschränkt** (durch Anzahl Variablen)
- Alle Lösungen in **derselben Tiefe** (in unterster Suchebene)
- Zustandsraum gerichteter **Baum**,
Anfangszustand ist Wurzel \rightsquigarrow **keine Duplikate** (**Warum?**)

Somit tritt keiner der für Tiefensuche problematischen Fälle auf.

Naives Backtracking: Beispiel

Betrachte das Constraint-Netz für folgendes Graphfärbungsproblem:

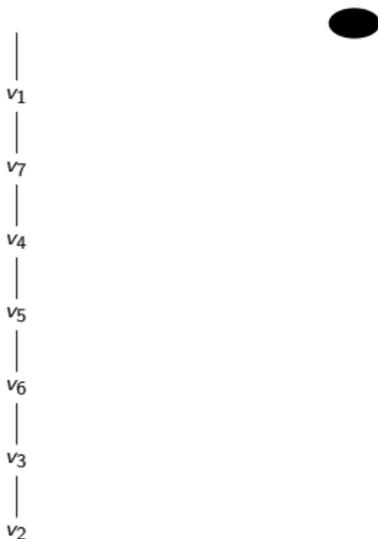


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

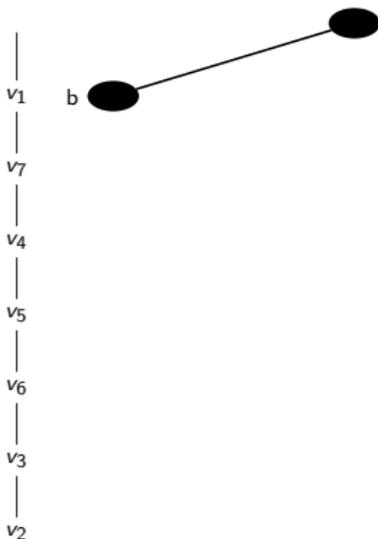


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

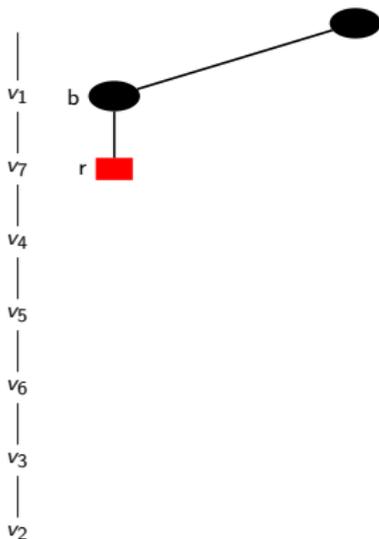


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

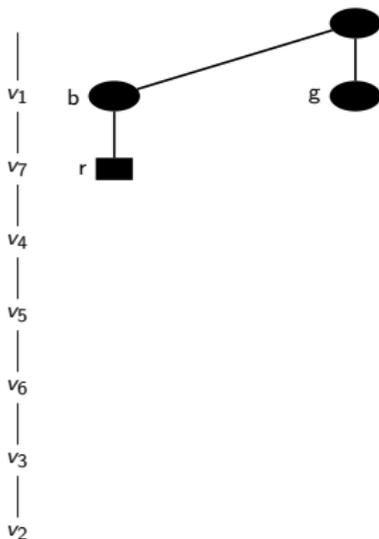


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

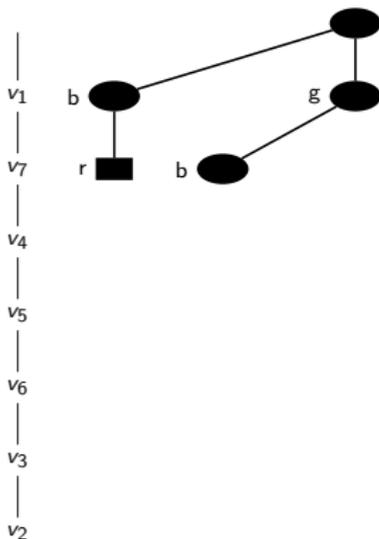


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

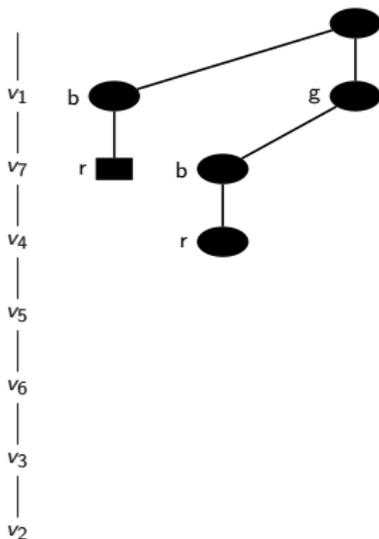


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

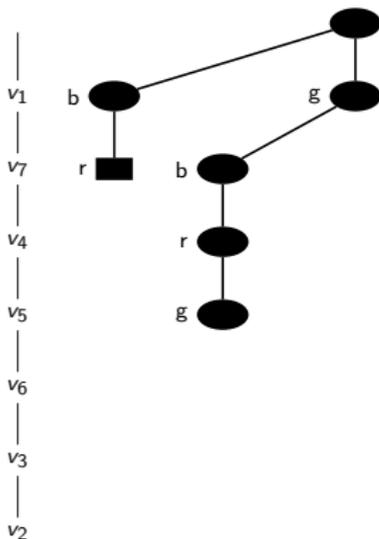


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

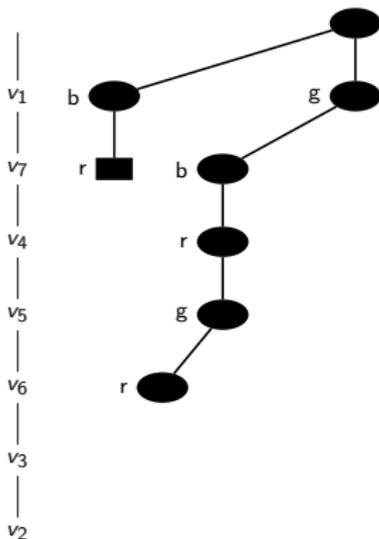


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

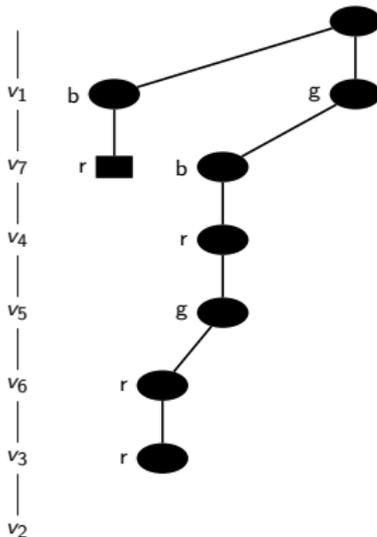


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

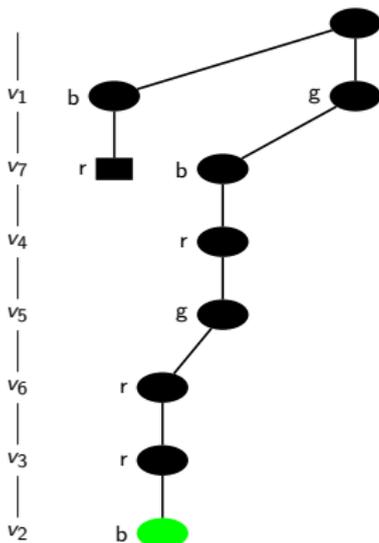


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

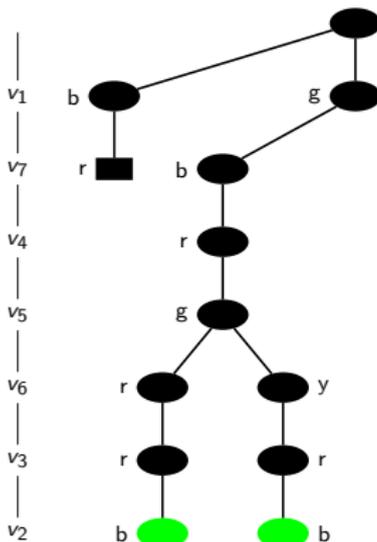


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

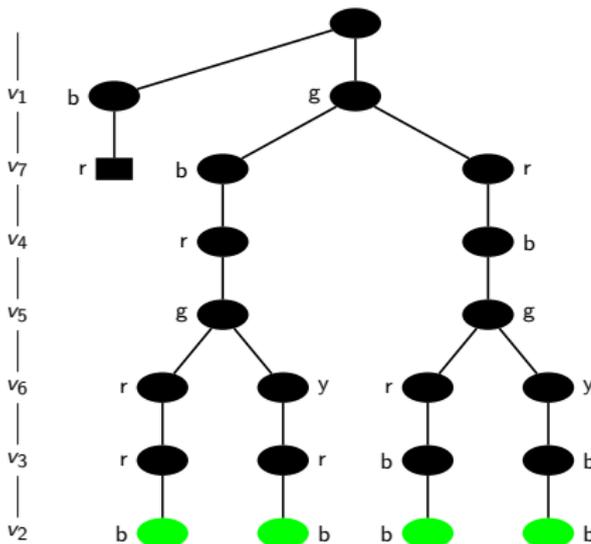


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)

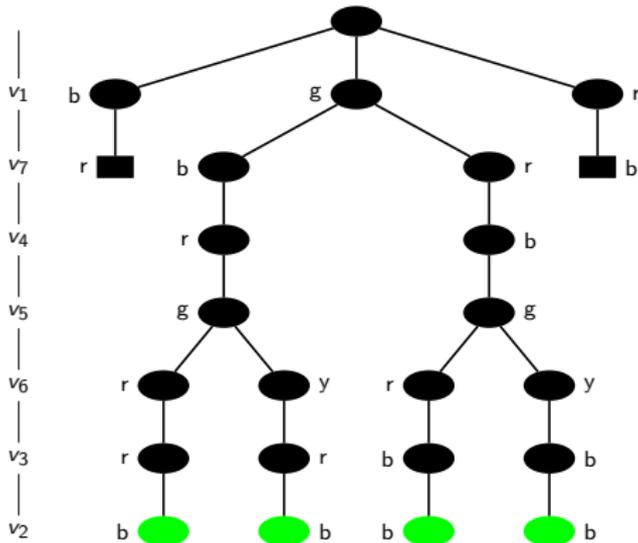


Naives Backtracking: Beispiel

Suchbaum für naives Backtracking mit

- **fester Variablenreihenfolge** $v_1, v_7, v_4, v_5, v_6, v_3, v_2$
- **alphabetischer** Reihenfolge der Werte

(ohne inkonsistente Knoten; bei Zielknoten fortgesetzt)



Naives Backtracking: Diskussion

- Naives Backtracking muss oft **ähnliche** Suchpfade (partielle Belegungen gleich bis auf wenige Variablen) erschöpfend durchsuchen.
- „Kritische“ Variablen nicht erkannt, daher (zu) spät belegt
- Entscheidungen, die später zwangsläufig zu Constraint-Verletzungen führen, werden erst erkannt, wenn alle beteiligten Variablen belegt wurden

↪ mehr Intelligenz durch **Fokus auf kritischen Entscheidungen** und **Inferenz** von Konsequenzen der bisherigen Entscheidungen

Variablen- und Wertordnungen

Naives Backtracking

```
function NaiveBacktracking( $\mathcal{C}, \alpha$ ):
```

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

```
  if  $\alpha$  is inconsistent with  $\mathcal{C}$ :
```

```
    return inconsistent
```

```
  if  $\alpha$  is a total assignment:
```

```
    return  $\alpha$ 
```

```
  select some variable  $v$  for which  $\alpha$  is not defined
```

```
  for each  $d \in \text{dom}(v)$  in some order:
```

```
     $\alpha' := \alpha \cup \{v \mapsto d\}$ 
```

```
     $\alpha'' := \text{NaiveBacktracking}(\mathcal{C}, \alpha')$ 
```

```
    if  $\alpha'' \neq \text{inconsistent}$ :
```

```
      return  $\alpha''$ 
```

```
  return inconsistent
```

Variablen- und Wertordnungen

Variablenordnung:

- Backtracking lässt offen, in welcher Reihenfolge **Variablen** belegt werden
- beeinflusst oft dramatisch die Grösse des Suchraums und damit die Performance der Suche
↪ Beispiel: Übungsaufgaben

Wertordnung:

- Backtracking lässt ebenfalls offen, in welcher Reihenfolge die **Werte** der ausgewählten Variable v betrachtet werden
- nicht ganz so wichtig, da in Teilbäumen ohne Lösung **nicht von Belang** (Warum nicht?)
- **wenn** Lösung im Teilbaum existiert, sollte nach Möglichkeit zunächst Wert ausgewählt werden, der zur Lösung führt (Warum?)

Statische vs. dynamische Ordnungen

Wir unterscheiden:

- **statische** Ordnungen (im Voraus festgelegt)
- **dynamische** Ordnungen (ausgewählte Variable/
ausgewählte Wertordnung hängt vom Suchzustand ab)

Vergleich:

- dynamische Ordnungen offensichtlich mächtiger
- statische Ordnungen verursachen dafür keinen Overhead während der Suche

Die folgenden Ordnungen können statisch vorgenommen werden, sind aber effektiver, wenn man sie mit Inferenz (\rightsquigarrow später) kombiniert und **dynamisch** auswertet.

Variablenordnungen

Zwei häufige Kriterien zur Variablenordnung:

- **Minimum Remaining Values:** wähle zuerst Variablen aus, deren **Wertebereich** möglichst klein ist
 - **Intuition:** wenige Teilbäume \rightsquigarrow kleiner Baum
 - **Extremfall:** nur **ein** Wert \rightsquigarrow erzwungene Belegung
- **Most Constraining Variable:** wähle zuerst Variablen aus, die an **möglichst vielen** nichttrivialen Constraints beteiligt sind
 - **Intuition:** Constraints möglichst früh testen
 \rightsquigarrow früh Inkonsistenzen erkennen \rightsquigarrow kleiner Baum

Kombination: verwende Minimum-Remaining-Values-Kriterium, dann Most-Constraining-Variable-Kriterium zum Tie-Breaking

Wertordnungen

Definition (Konflikt)

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

Für Variablen $v \neq v'$ und Werte $d \in \text{dom}(v)$, $d' \in \text{dom}(v')$ steht $v \mapsto d$ im **Konflikt** mit $v' \mapsto d'$, falls $\langle d, d' \rangle \notin R_{vv'}$.

Kriterium zur Wertordnung für partielle Belegung α und ausgewählte Variable v :

- **Minimum Conflicts:** Bevorzuge Werte $d \in \text{dom}(v)$, für die $v \mapsto d$ an möglichst wenigen Konflikten mit in α un belegten Variablen beteiligt ist.

Zusammenfassung

Zusammenfassung: Backtracking

grundlegender Suchalgorithmus für Constraint-Netze: **Backtracking**

- erweitere (anfänglich leere) partielle Belegung schrittweise, bis **Inkonsistenz** oder **Lösung** gefunden
- ist eine Form der **Tiefensuche**
- Tiefensuche hier besonders geeignet, da der Zustandsraum ein gerichteter Baum ist und alle Lösungen in derselben, a priori bekannten Tiefe liegen

Zusammenfassung: Variablen- und Wertordnungen

- **Variablenordnungen** beeinflussen die Performanz von Backtracking massgeblich
 - Ziel: **kritische** Entscheidungen möglichst früh
- **Wertordnungen** beeinflussen die Performanz von Backtracking auf **lösbaren** Constraint-Netzen massgeblich
 - Ziel: **viel versprechende** Belegungen zuerst