Künstliche Intelligenz

Problemlösen mit Constraints

Dr. Christian Meilicke
Research Group Data and Web Science
Universität Mannheim

Teile der Vorlesung basieren auf einem
Foliensatz von Prof. Dr. Heiner Stuckenschmidt

Restprogramm

- Siehe ILIAS Startseite:
  - 17.11. Vorlesung 07: Constraint Satisfaction Problems
  - 24.11. Vorlesung 08: Logik I
  - 01.12. Vorlesung 09: Logik II
  - 08.12. Vorlesung 10: Planen

- Noch drei Blätter (7,8,9) die korrigiert werden
- Blatt 10 wird nicht mehr bewertet, aber klausur relevant!

- Morgen Bohnen-WM
- Durchsicht der Abgaben Programmierprojekt evtl. erst Anfang nächster Woche

Programmierprojekt

Alle, die nach einigen erfolglosen Versuchen etwas abgegeben haben, was die Refernz-KI nicht geschlagen hat:

Finale Deadline: Freitag 20.11. 14:00 Uhr

Korrekturen erst am Wochenende!

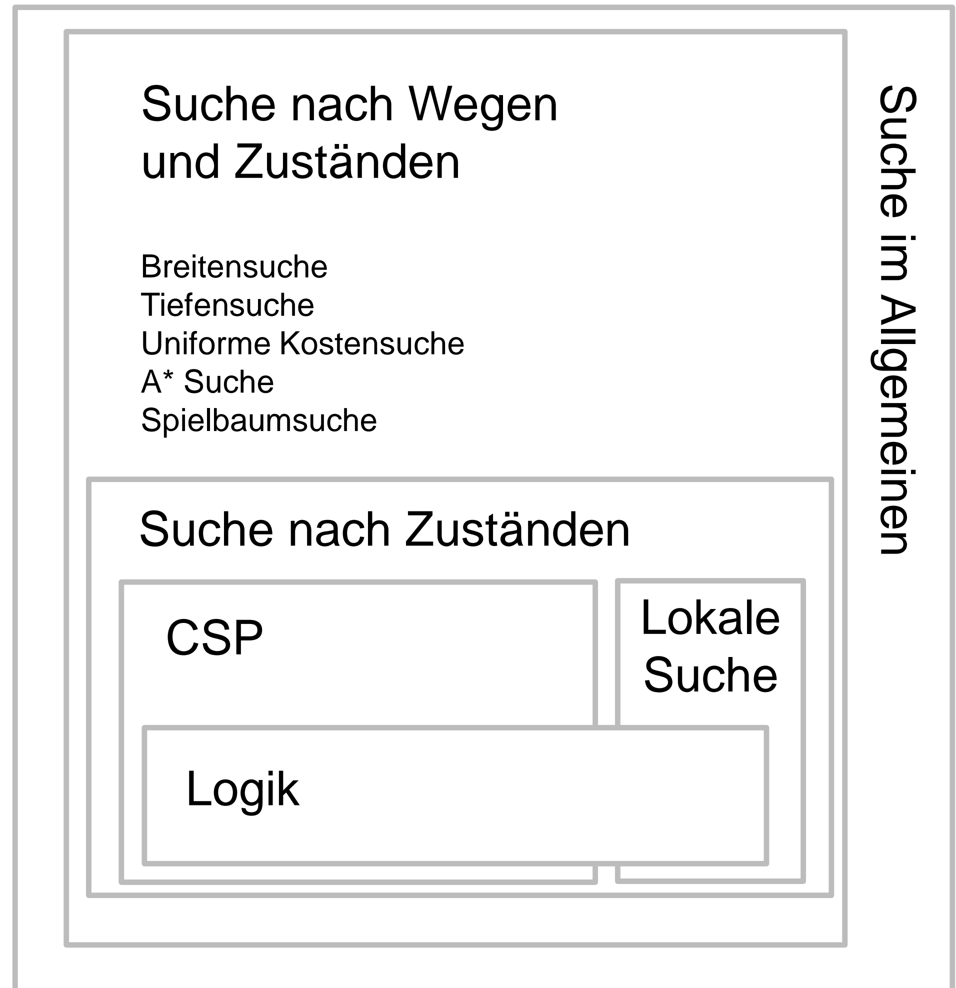
Klausur

- Probeklausur, was den reinen technischen Ablauf betrifft in ILIAS verlinkt:
  - Ablauf bei der Klausur leicht verschieden
  - Schritt 1 wird zu Schritt 1 und 2
    - Schritt 1: Einverständniserklärung ect.
    - Schritt 2: Download
  - Schritt 2 wird zu Schritt 3
    - Schritt 3 : Abgabe der Lösung

- „Echte“ Probeklausur
  - Auf dem Webserver unter „Sonstiges“
  - Aufgabe 6 anderes Thema, sonst typische Klausur

Rückblick und Ausblick

- Drei Themen
  - Suchverfahren
  - CSP
  - Logik
- Zum Teil sind Inhalte überlappend
- Methoden aus einem Teil anwendbar auf anderen Teil
  - Robinsonroulette kann zum Beispiel auch mit Logik gelöst werden



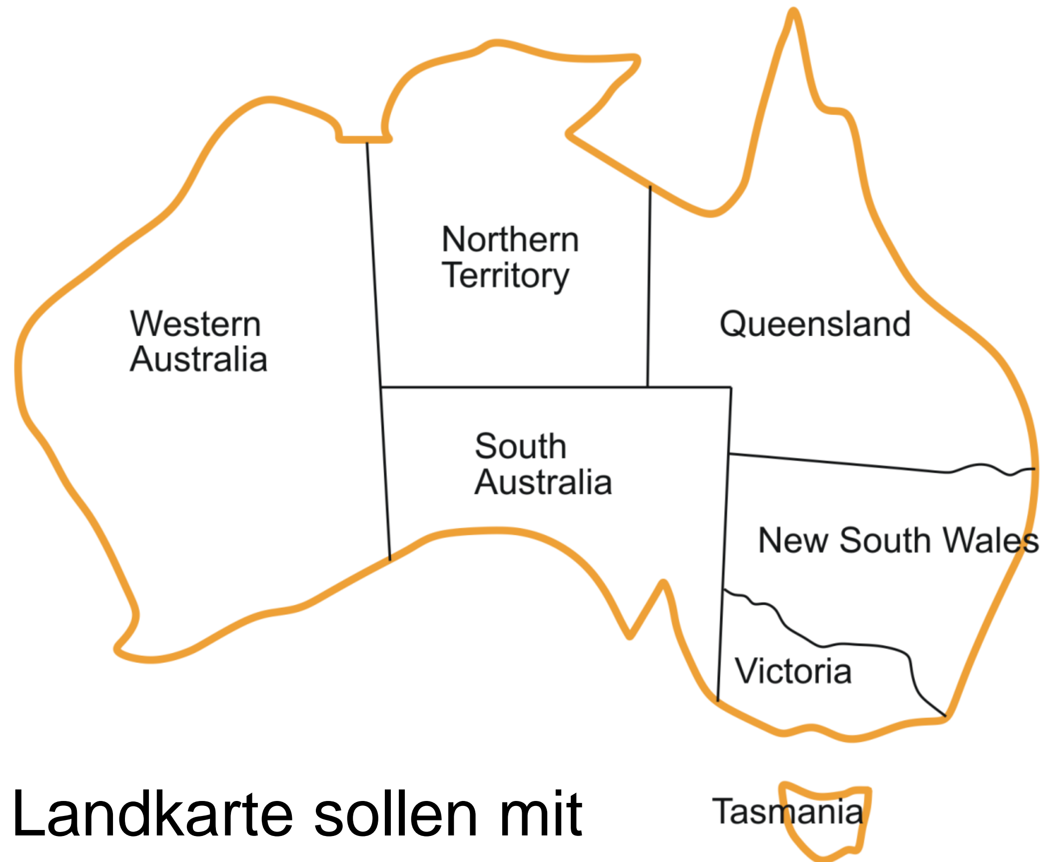
Suche vs. Constraints

- Suche ist ein allgemeines Paradigma
  - Das Problem wird als Zustandsraum dargestellt
  - Zustände haben keine interne Struktur
  - Problem-spezifische Heuristiken können die Suche erleichtern
- Constraint-Probleme sind ein Spezialfall von Suchproblemen
  - Zustände im Suchraum haben eine vorgegebene Struktur
  - Diese Struktur erlaubt es problem-unabhängige Heuristiken anzuwenden

Struktur eines Constraint Problems

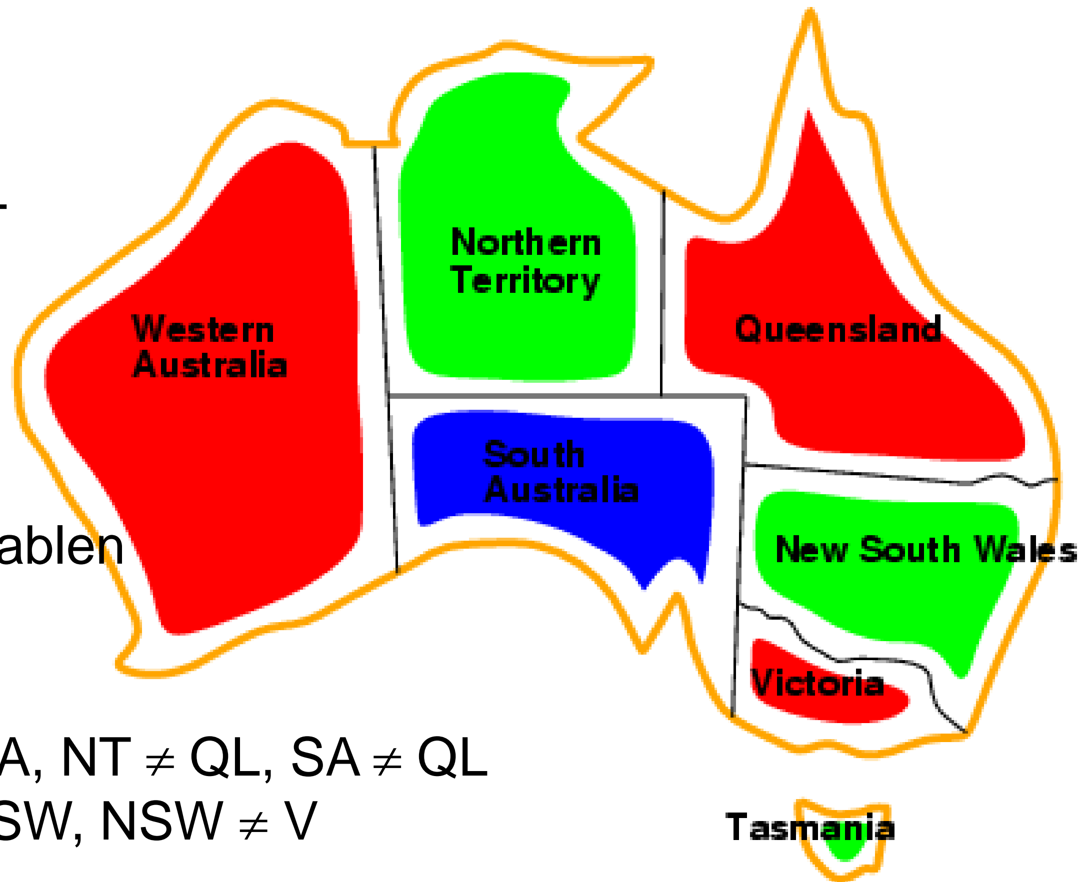
- Ein Constraint-Problem besteht aus:
  - Menge von Variablen $X_1, ..., X_n$ mit Wertebereichen $D_1, ..., D_n$
  - Menge von Constraints $C_1, C_2, ..., C_m$ (Einschränkungen der möglichen Werte von $X_1, X_2, ..., X_n$) in einer formalen Sprache
- Ein Zustand ist eine (teilweise) Belegung der Variablen mit Werten $X_i = v_i$ mit $v_i \in D_i$
- Eine Belegung, die alle Constraints, erfüllt heißt konsistent
- Eine Lösung ist eine vollständige konsistente Belegung der Variablen

Beispiel: Drei Farben Problem



Aufgabe: Länder einer Landkarte sollen mit drei Farben so eingefärbt werden, dass keine benachbarten Länder die gleiche Farbe haben

Repräsentation des Problems



Variablen:
WA, NT, SA, QL, NSW, V, T

Wertebereiche:
D_{WA} = {rot, grün, blau} und
analog für die anderen Variablen

Constraints:
WA ≠ NT, WA ≠ SA, NT ≠ SA, NT ≠ QL, SA ≠ QL
SA ≠ NSW, SA ≠ V, QL ≠ NSW, NSW ≠ V

Lösung:
WA=rot, NT=grün, QL=rot, SA=blau, NSW=grün, V=rot, T=grün

Historische Anmerkung

- Vier Farben Satz:
  - Vier Farben reichen immer aus, um eine beliebige Landkarte so einzufärben, dass keine zwei angrenzenden Länder die gleiche Farbe bekommen
- Der Satz wurde 1852 aufgestellt
  - Es gab mehrfache Beweisversuche
  - Falsche Gegenbeispiele
- Erst 2005 konnte mit Hilfe eines Computerprogramms die Richtigkeit der Behauptung bewiesen werden

Formulierung als Suche

- Formulierung als Suchproblem
  - Zustandsraum: konsistente Belegungen
  - Startzustand: die leere Belegung
  - Lösungsschritt: Zuweisung einer Variablen
  - Zielzustand: vollständige Belegung
  - Schrittkosten: konstant für jeden Schritt

- Eigenschaften
  - Suchtiefe von Lösungen ist bekannt (= Anzahl der zu belegenden Variablen)
  - Lösungsweg nicht von Bedeutung (Kommutativität)

Backtracking Suche für Constraint Probleme

- Tiefensuche auf partiellen Belegungen
- In jedem Schritt wird einer Variablen ein Wert zugeordnet
- Führt keine mögliche Zuordnung zu einer konsistenten Belegung wird zurückgegangen (Backtracking)
- **Wieso ist Tiefensuche hier als grundlegendes Verfahren sinnvoll?**

Eigenschaften: Tiefensuche

Da d=m, keine Kreise, und konstante Kosten, ist Vollständigkeit und Optimalität gewährleistet

- Vollständigkeit:

  NEIN: Der Suchraum kann unendlich sein. Beginnt die Suche in einem unendlichen Zweig, werden Lösungen in anderen nicht gefunden

- Optimalität:

  NEIN: Der zuerst expandierte Zweig kann eine Lösung enthalten, die in einer größere Tiefe liegt als die optimale Lösung

- Zeitkomplexität:

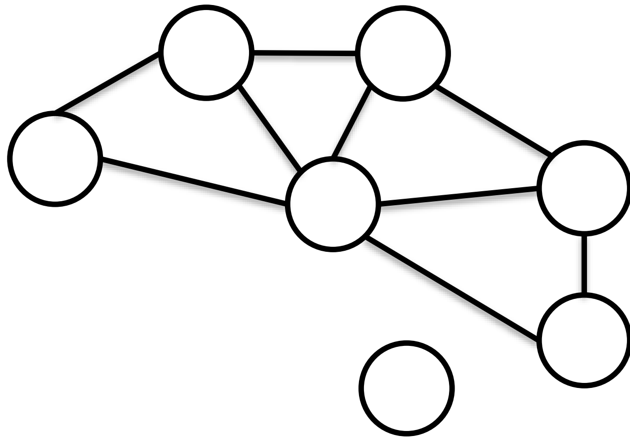
  $O(b^m)$: Es müssen (fast) alle möglichen Knoten eines Zweigs expandiert werden, auch wenn keine Lösung enthalten ist.

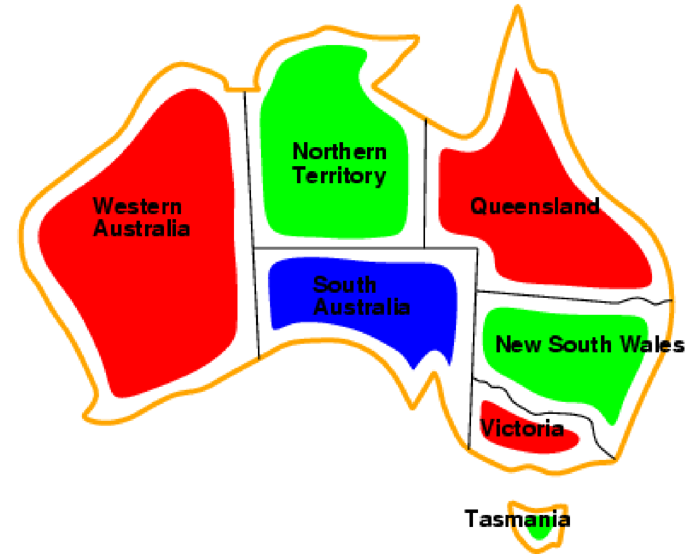
- Speicherkomplexität:

  $O(b*m)$: Es muss immer nur ein Zweig gespeichert werden. Ist er vollständig durchsucht, kann er gelöscht werden.

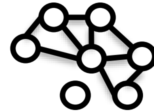
d = Tiefe der Lösung
m = Max.Tiefe des Suchbaums
b = Branching Faktor
c = Kosten der Lösung
e = Minimale Kosten pro Schritt

Backtracking Beispiel

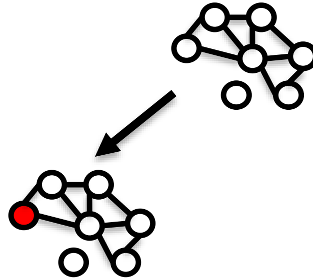




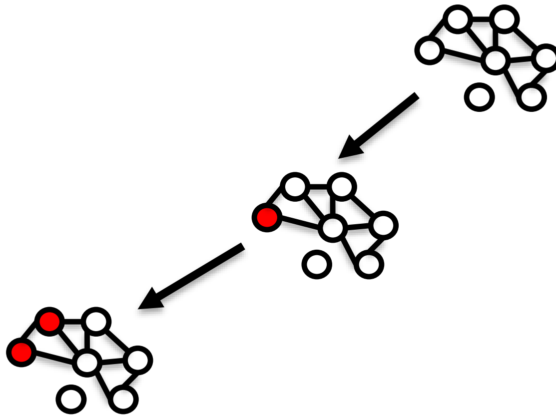
Backtracking Beispiel



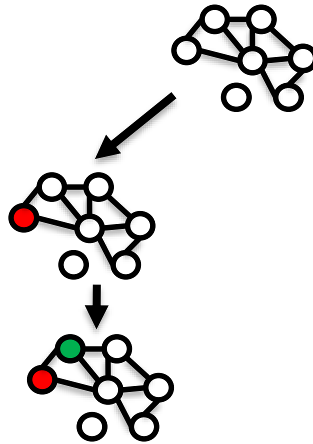
Backtracking Beispiel



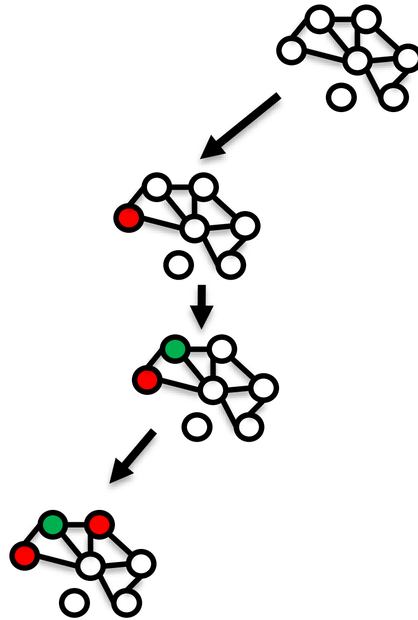
Backtracking Beispiel



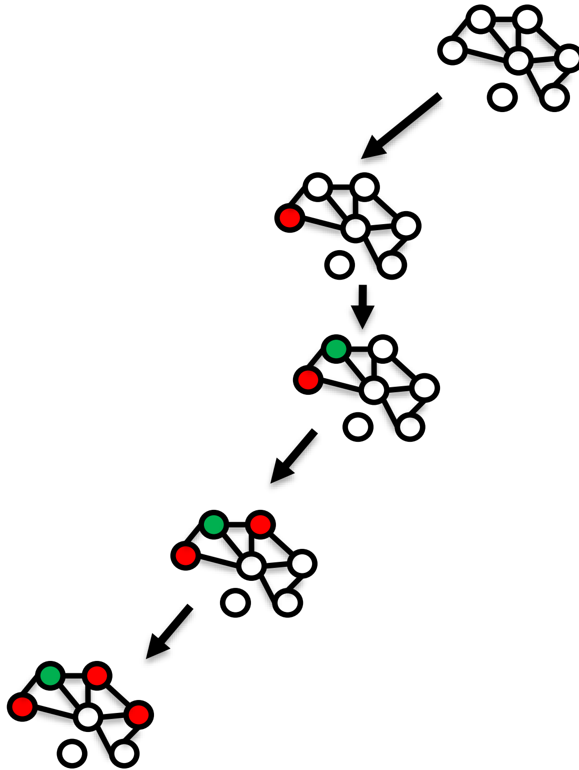
Backtracking Beispiel



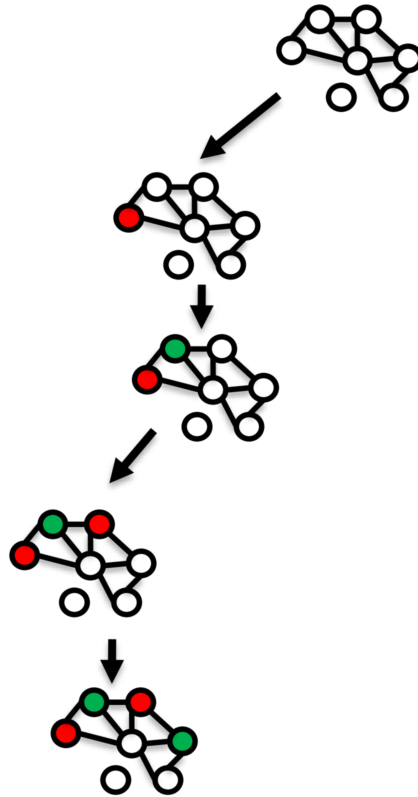
Backtracking Beispiel



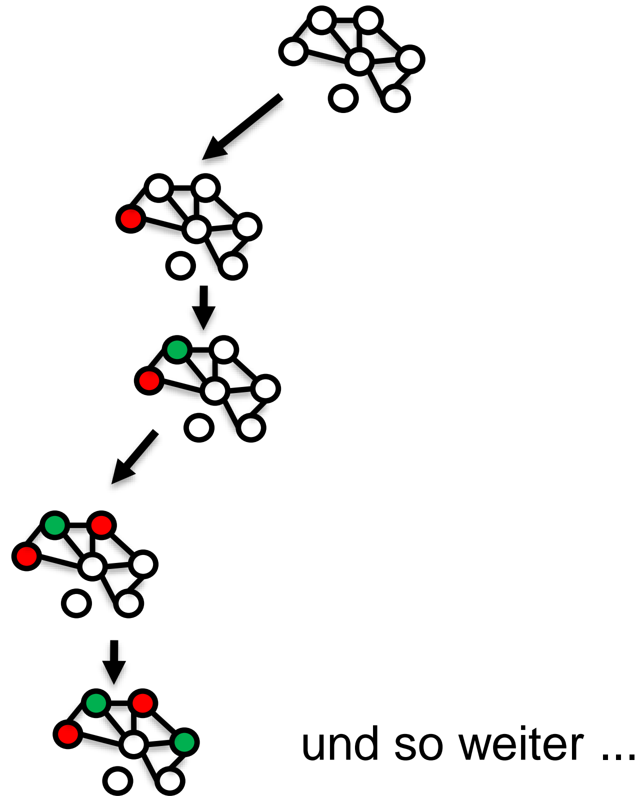
Backtracking Beispiel



Backtracking Beispiel



Backtracking Beispiel



und so weiter ...

Backtracking Algorithmus

```
Solve(1)
// V ist Feldvariable mit der Lösung
// V[i] ist der Wert der i-ten Variable


Solve(int i)
   IF i > n:
      solution = true
      RETURN
   FOR v from Domain(i):
      V[i] = v
      IF getConflict(i) == 0 Solve(i+1)
      IF solution == true RETURN
```

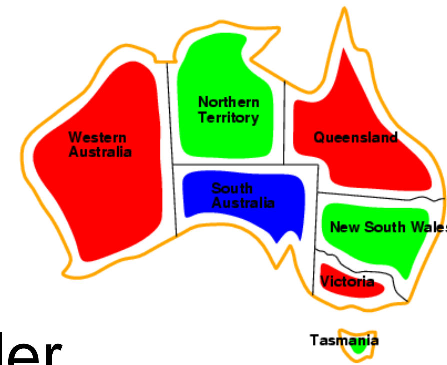
Effizienz von Backtracking

- Backtracking ist im allgemeinen komplex

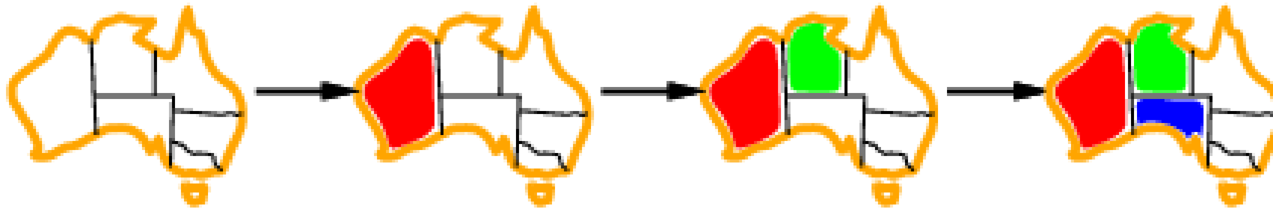
- Problem:
  - dead-ends, die Backtracking erzwingen
  - Kein Backtracking = lineare Laufzeit

- Ziel: Vermeidung von dead ends durch:
  - Intelligente Auswahl von Variablen und Werten
  - Einschränkung des Suchraums

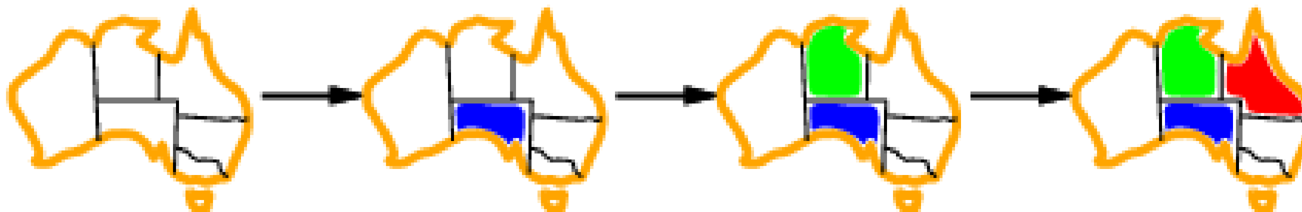
Heuristiken für Variablenwahl



- **Minimum remaining Values:** Variable mit der geringsten Anzahl möglicher Werte wird zuerst belegt (SA wird vor QL belegt)

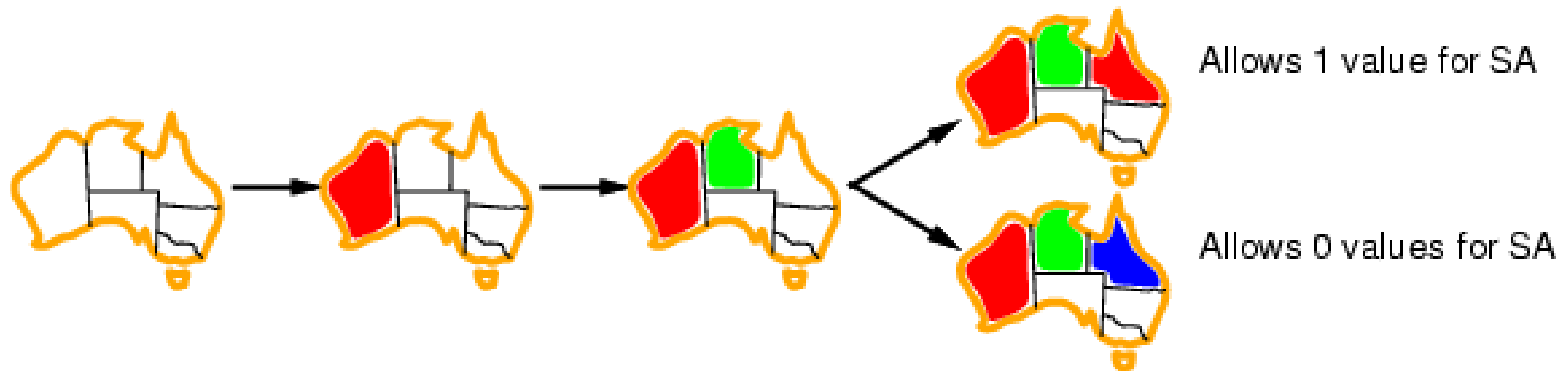


- **Degree Heuristic:** Variable, die in den meisten Constraints anderer, nicht zugewiesener Variablen enthalten ist (SA wird als erstes belegt)

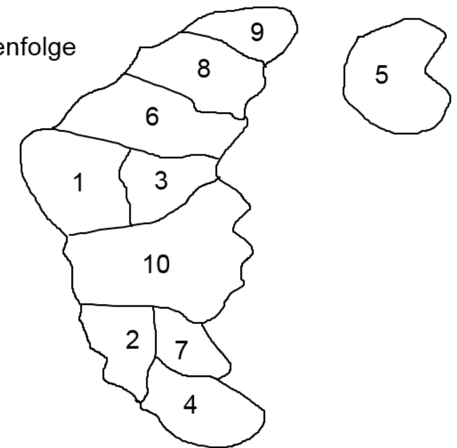


Heuristiken für Wertewahl

- **Least Constraining Value:** Wert, der andere Variablen am wenigsten einschraenkt wird gewählt (NT wird mit rot und nicht blau belegt)



Beispiel



- Variablenwahl
  - (1) Degree Heuristic
  - (2) Minimum Remaining Values

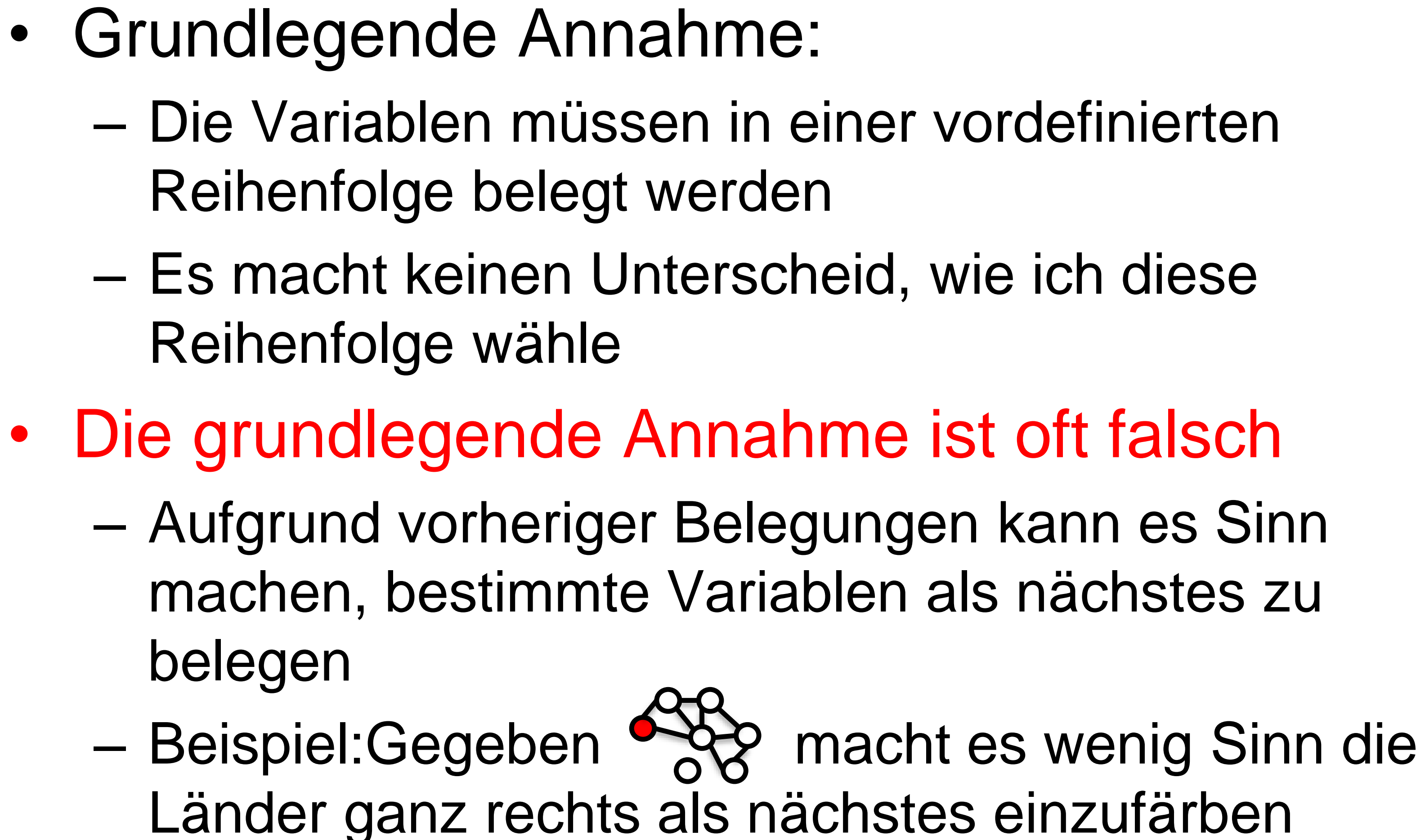
- Wertewahl
  - Least Constraing Value

- In welcher Reihenfolge werden die Variablen belegt und welche Werte werden gewählt, wenn (1) vor (2) angewendet wird?

Intelligentes Backtracking I

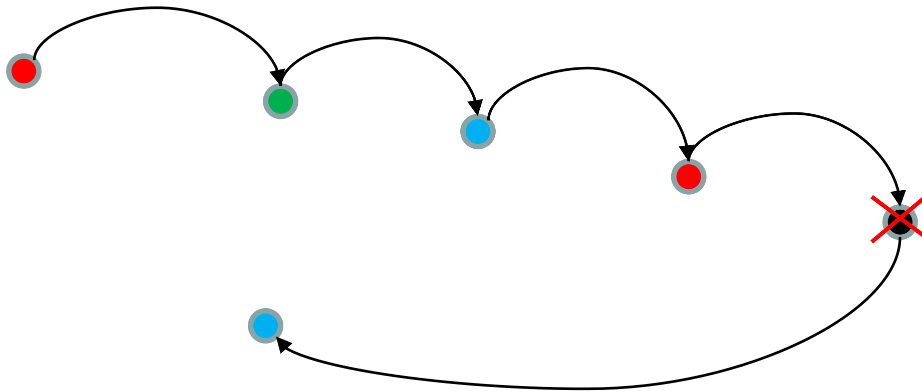
- Grundlegende Annahme:
  - Das Problem liegt bei der Belegung der direkten Vorgängervariablen
  - Es reicht einen Schritt zurück zu gehen, um das Problem zu lösen
- Die grundlegende Annahme ist oft falsch
  - Das eigentliche Problem wurde früher verursacht
  - Idee: Merke dir welche Werte zu einem Ausschluss geführt haben
  - Backjumping und Conflict Directed Backtracking

Intelligentes Backtracking II

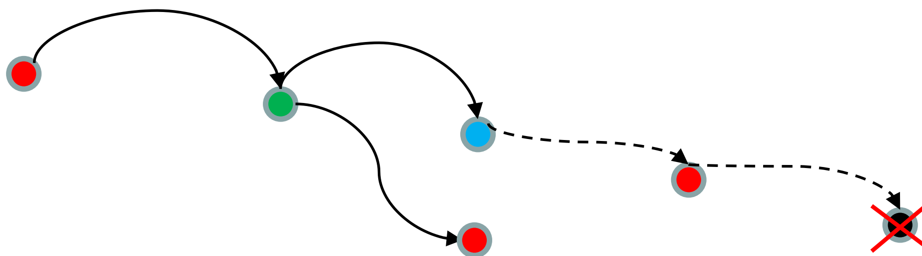
- Grundlegende Annahme:
  - Die Variablen müssen in einer vordefinierten Reihenfolge belegt werden
  - Es macht keinen Unterscheid, wie ich diese Reihenfolge wähle
- Die grundlegende Annahme ist oft falsch
  - Aufgrund vorheriger Belegungen kann es Sinn machen, bestimmte Variablen als nächstes zu belegen
  - Beispiel:Gegeben  macht es wenig Sinn die Länder ganz rechts als nächstes einzufärben

Intelligentes Backtracking

(I) „Rückwärtssprünge“ statt „Rückwärtsschritte“



(II) Entscheidungen „nach vorne propagieren“



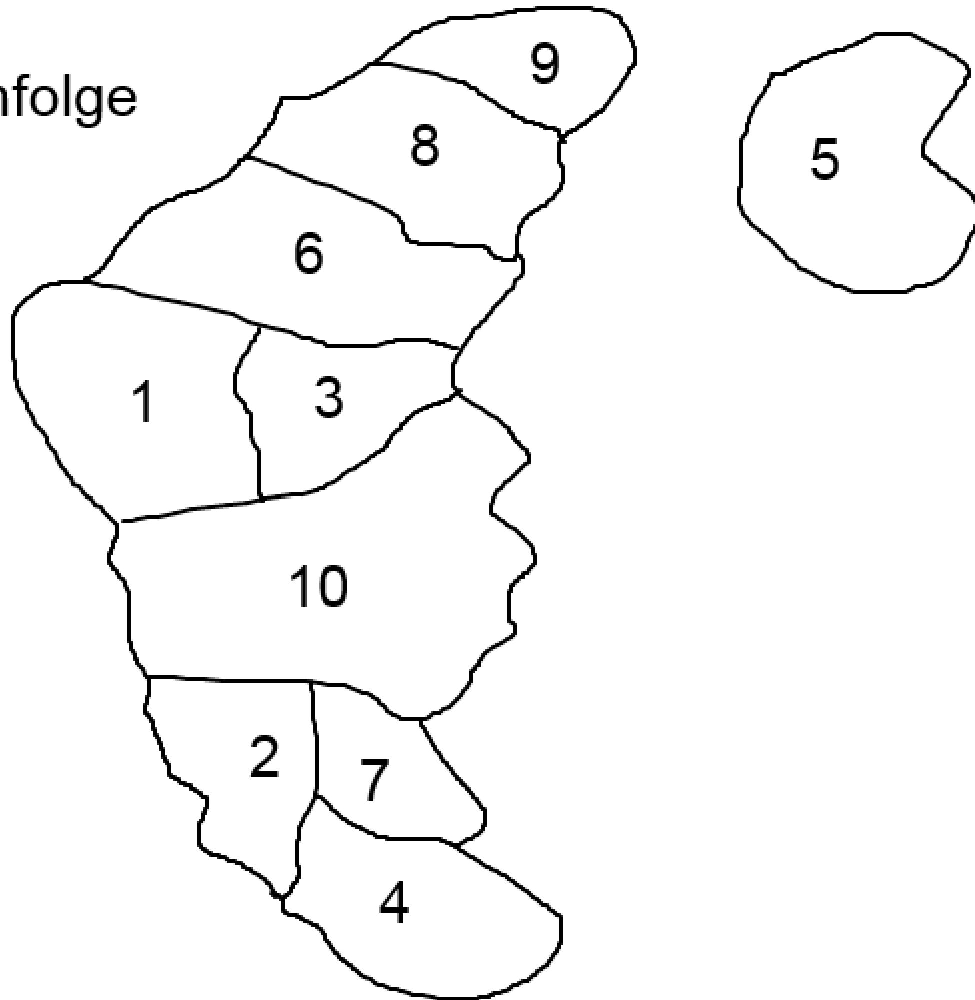
Conflict-Directed Backjumping (CBJ)

- Merke dir für jede Variable, welche anderen Variablen zu einem Konflikt geführt haben

- Verwendung von Konfliktmengen `conflict(i)` für jede Variable `V[i]`:
  - Füge zu `conflict(i)` jeden ersten Index `j` hinzu der einen Wert verboten hat
  - Wenn kein Wert für `V[i]` mehr verfügbar ist, dann, springe zu `h = max(conflict(i))` zurück
  - Setze in diesem Fall `conflict(h):= conflict(h) ∪ conflict(i) - {h}`

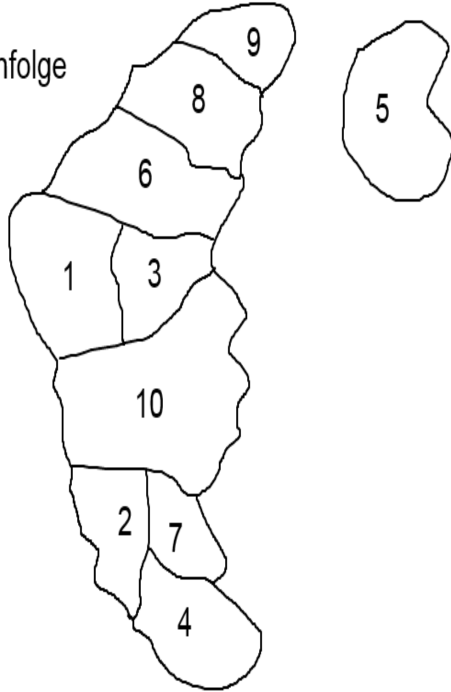
Beispiel

Wertereihenfolge
R (rot)
G (grün)
B (blau)



An der Tafel!

(Anwesenheit in der Vorlesung kann hilfreich sein)


Wertereihenfolge
R (rot)
G (grün)
B (blau)
9
8
5
6
1
3
10
2
7
4

Backjumping (BJ)

- Immer nur ein Rücksprung, danach normale „Backsteps“
- (Erster) Rücksprung analog wie bei CBJ
- Offensichtlich weniger effektiv

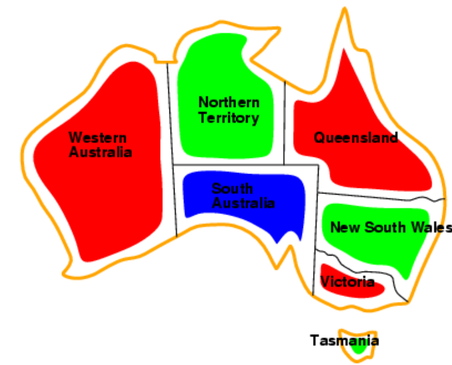
- CBJ ist eine Erweiterung von BJ bei der mehr Informationen gespeichert werden.
  - Möglichst viele Rücksprünge
  - Ohne eine mögliche Lösung zu überspringen

Pruning des Suchraums

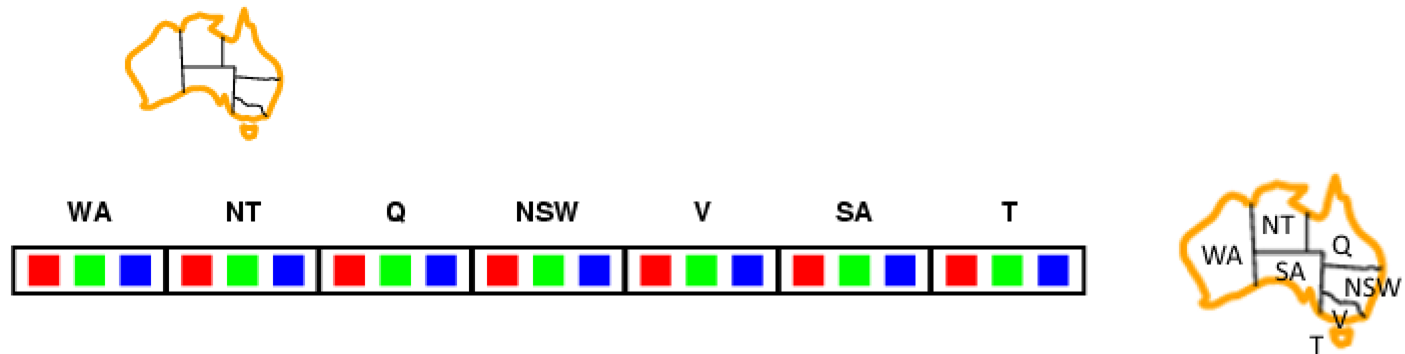
- Zwei Prozesse:
  - Suche: Belegung von Variablen
  - Pruning: Einschränkung möglicher zukünftiger Belegungen (Wertebereiche)

- Interaktion:
  - A priori: Suchraum wird eingeschränkt und anschließend gesucht
  - Integriert: Nach jeder Belegung werden die Auswirkungen auf den Suchraum bestimmt

Pruning des Suchraums

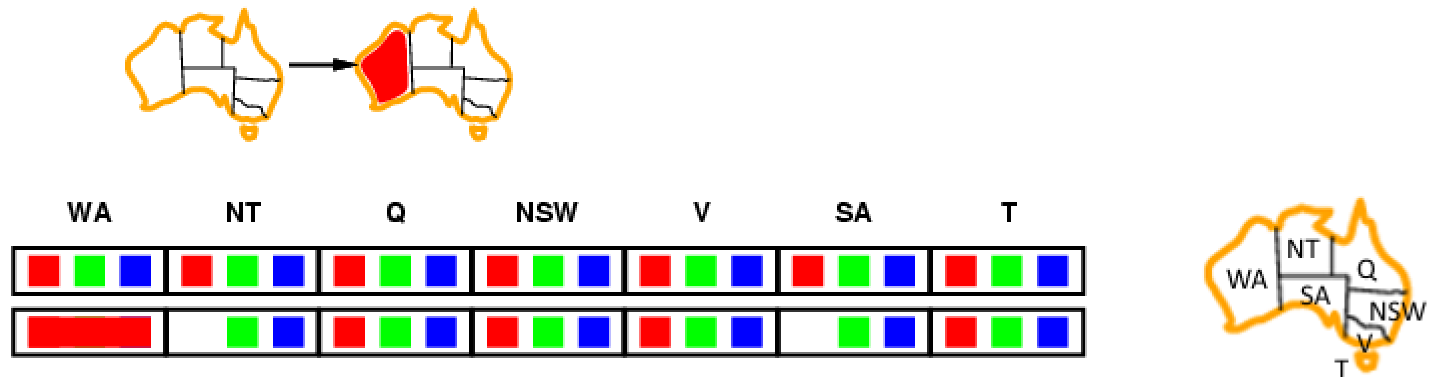


- Forward Checking:
  - Bei jeder neuen Belegung werden nicht mehr mögliche Werte nicht-belegter Variablen aus der entprechenden Domäne gelöscht



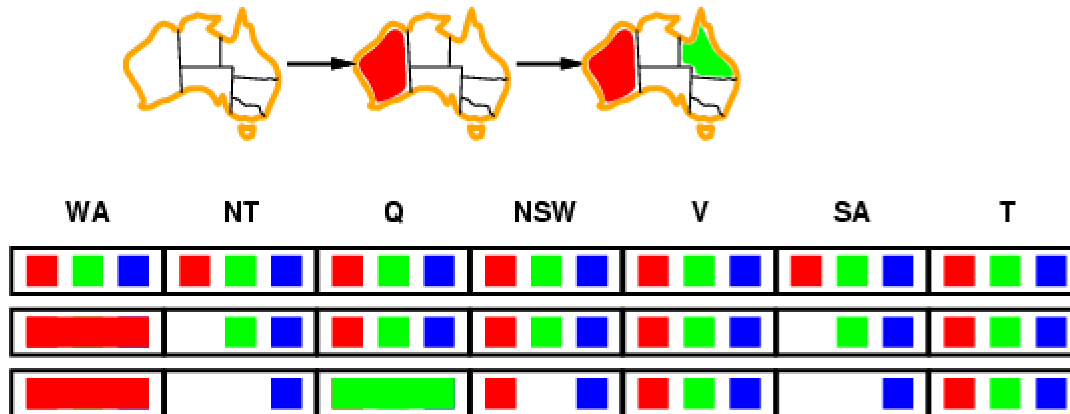
Pruning des Suchraums

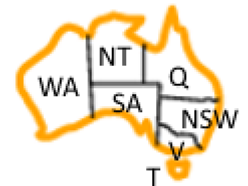
- Forward Checking:
  - Bei jeder neuen Belegung werden nicht mehr mögliche Werte nicht-belegter Variablen aus der entprechenden Domäne gelöscht



Pruning des Suchraums

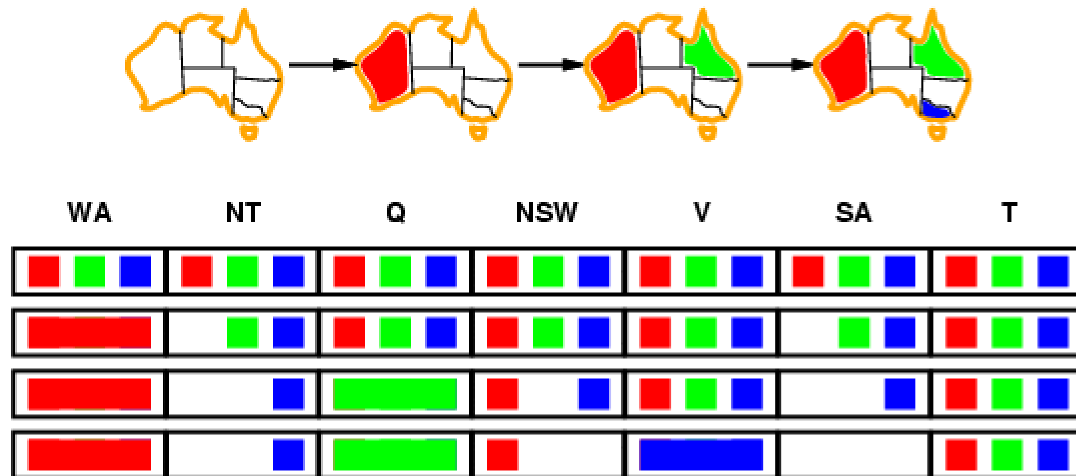
- Forward Checking:
  - Bei jeder neuen Belegung werden nicht mehr mögliche Werte nicht-belegter Variablen aus der entprechenden Domäne gelöscht

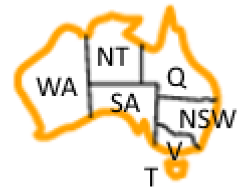




Pruning des Suchraums

- Forward Checking:
  - Bei jeder neuen Belegung werden nicht mehr mögliche Werte nicht-belegter Variablen aus der entprechenden Domäne gelöscht.

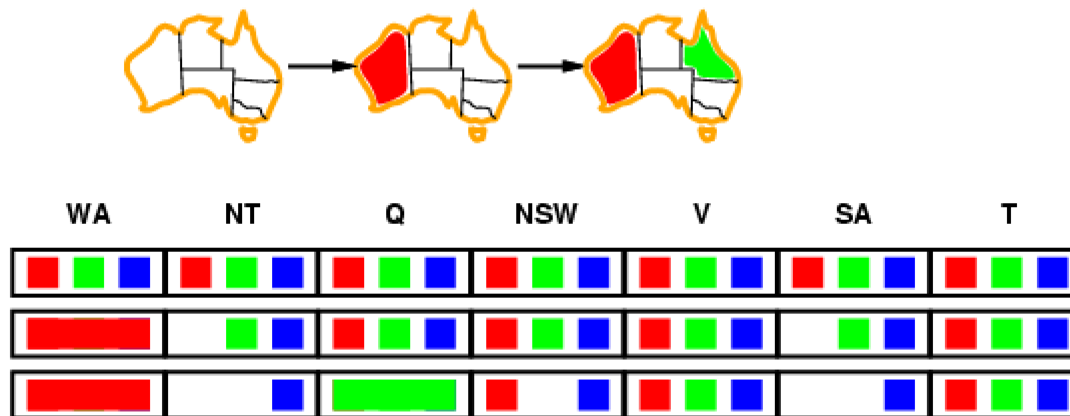


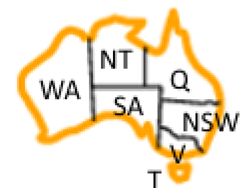


Ist eine Domäne leer muss zurückgegangen werden.

Erweitertes Pruning

- Forward Checking findet nicht alle möglichen Konflikte. Beispiel:

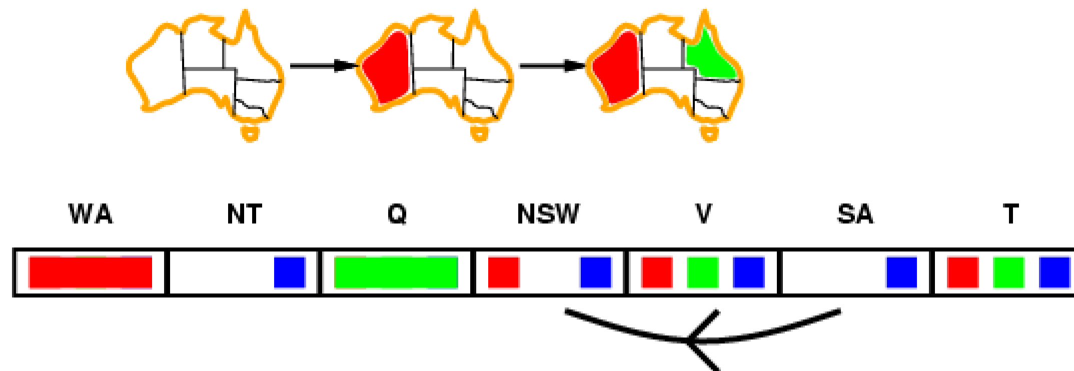


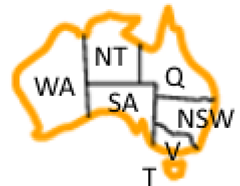


- Sowohl SA als auch NT müssten blau gefärbt werden → Konflikt
- Problem: Interaktion mehrerer Constraints

Constraint Propagation

- Idee: schon vor weiteren Belegungen mögliche Auswirkungen aller Constraints berechnen:

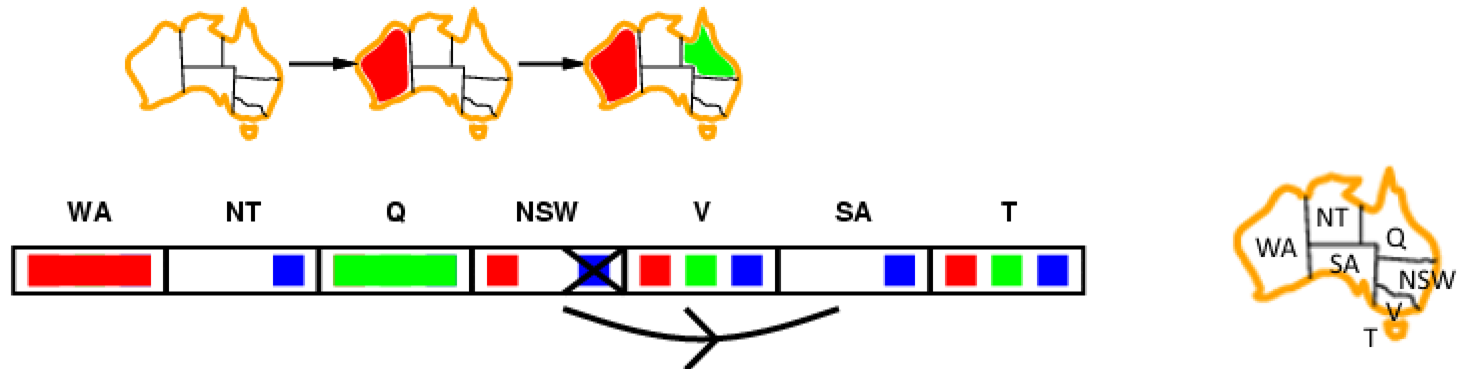




- SA müsste blau gefärbt werden, dies hat Auswirkungen auf benachtbarte Variablen
- NSW darf nicht blau sein !

Constraint Propagation

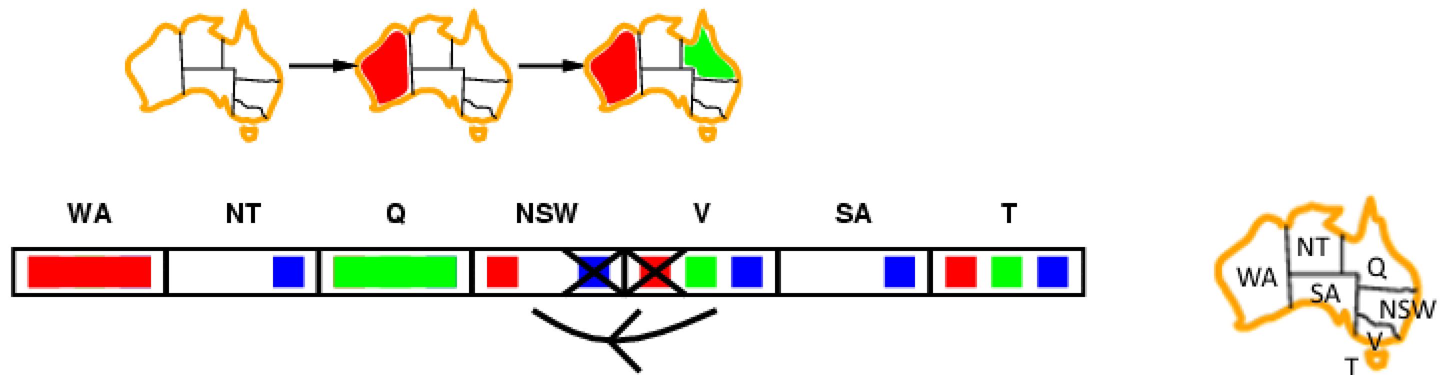
- Idee: schon vor weiteren Belegungen mögliche Auswirkungen aller Constraints berechnen:



- Wird eine Domäne eingeschränkt, müssen die benachbarten Variablen überprüft werden
- Da NSW nun rot sein muss, kann V nicht rot sein

Constraint Propagation

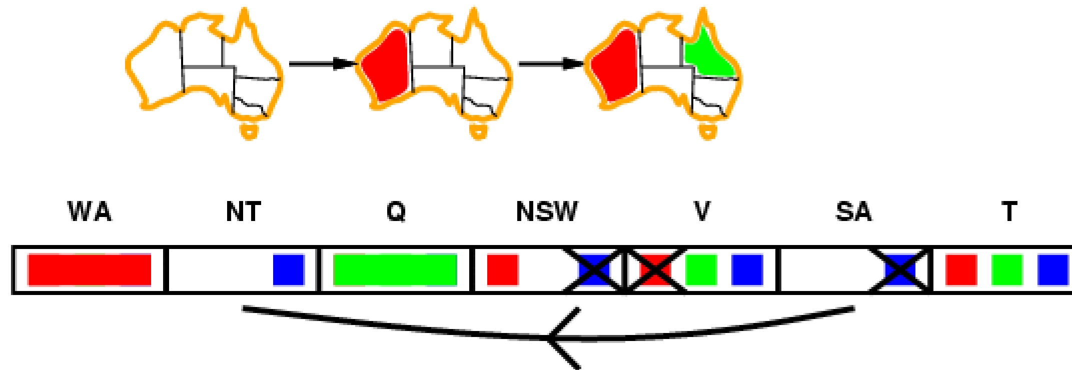
- Idee: schon vor weiteren Belegungen mögliche Auswirkungen aller Constraints berechnen:

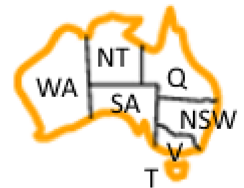


- V muss also grün oder blau sein
- Kein Problem für NSW oder SA

Constraint Propagation

- Idee: schon vor weiteren Belegungen mögliche Auswirkungen aller Constraints berechnen:





- Gleicher Test für andere Nachbarn von SA
- NT darf nicht blau sein
- Konflikt: Leere Domäne entweder bei NT oder SA

Zusammenfassung

- Suche vs. CSP
  - Was ist ein CSP?
  - Backtracking als Standardverfahren zur Lösung

- Heuristiken zur Vermeidung von Backtracking
  - Wertewahl & Variablenwahl
- “Rückwärts-Erweiterungen”
  - Backjumping
  - Conflict Directed Backjumping
- “Vorwärts-Erweiterungen”
  - Forward Checking
  - Constraint Propagation

Ausblick Restprogramm

- Nächste Woche Aussagenlogik

- Drei Vorlesungen zu Aussagenlogik
  - 29.10. Grundlagen der Modellierung: Syntax und Semantik
  - 05.11. Algorithmen um Erfüllbarkeit zu prüfen (Reasoning)
    - Ausgabe Programmierprojekt II
  - 12.11. Ein komplexes Beispiel lösen mittels Aussagenlogik
    - Freiwillig, kann fürs Programmierprojekt nützlich sein

- Planen

- Letzte Woche Fragestunde