

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

Programmierprojekt 1

Alle, die nach einigen erfolglosen Versuchen (!) etwas abgegeben haben, was die Referenz-KI nicht geschlagen hat:

Finale Deadline: Freitag 29.11. 10:00 Uhr

Korrekturen erst am Wochenende!

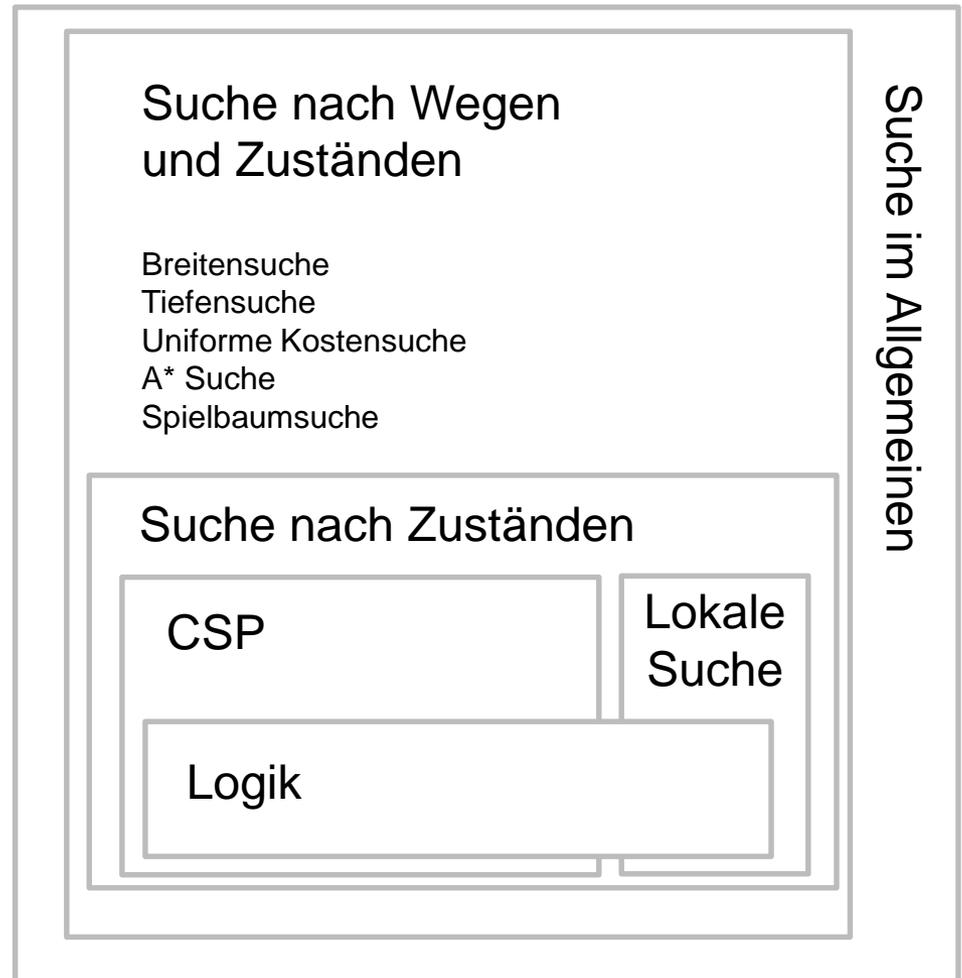
Morgen: Bohnen WM (Anmeldungen im Forum nicht vergessen)

Programmierprojekt 2

- Die nächste drei Vorlesungen:
 - Vorlesung 07: Constraint Satisfaction Problems
 - Vorlesung 08: Logik I
 - Woche: Vorlesung 09: Logik II
- Dann eine Woche Vorlesungspause:
 - Zweites Programmierprojekt
- Überlegung um Zulassungsvoraussetzungen zu erleichtern
 - Programmierprojekt => 20 Punkte Blatt, relevant für 60%
 - 20 Punkte, wenn sehr gute Abgabe
 - 10 Punkte, wenn Minimalanforderungen erfüllt werden, sonst 0 Punkte
 - Kein hartes Ausschlusskriterium

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



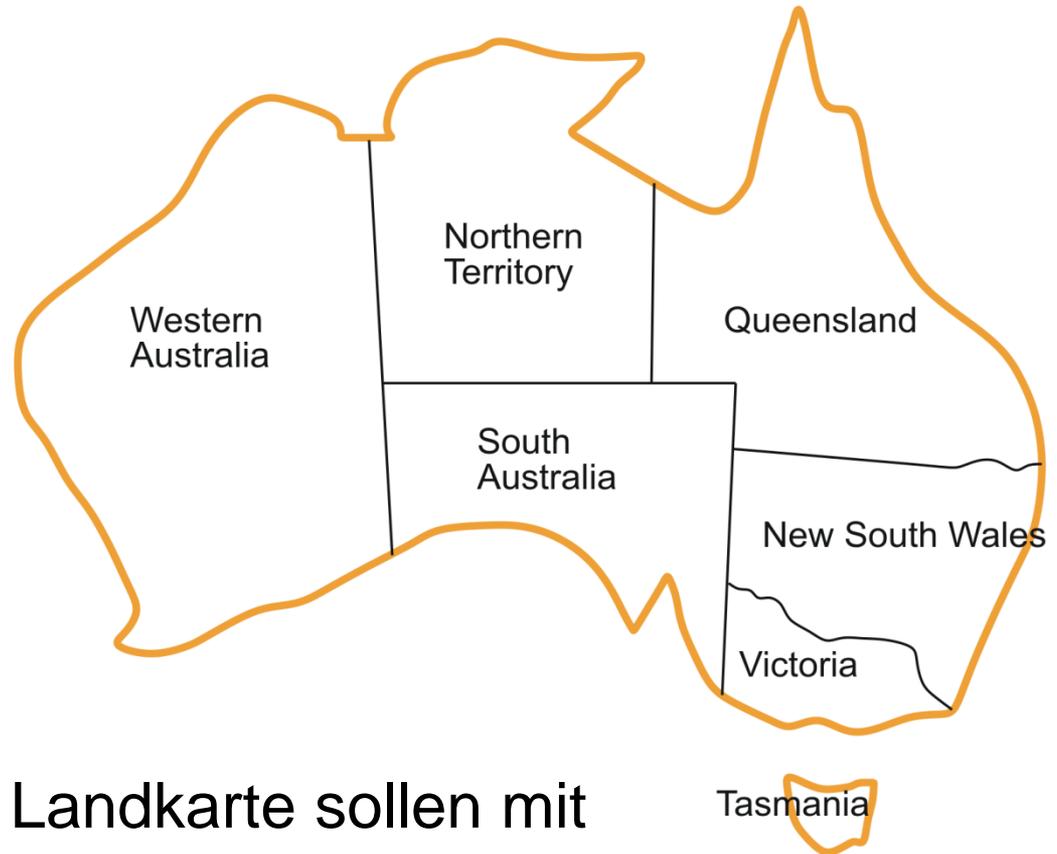
Suche vs. Constraints

- 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, \dots, X_n mit **Wertebereichen** D_1, \dots, D_n
 - Menge von **Constraints** C_1, C_2, \dots, C_m (Einschränkungen der möglichen Werte von X_1, X_2, \dots, 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:

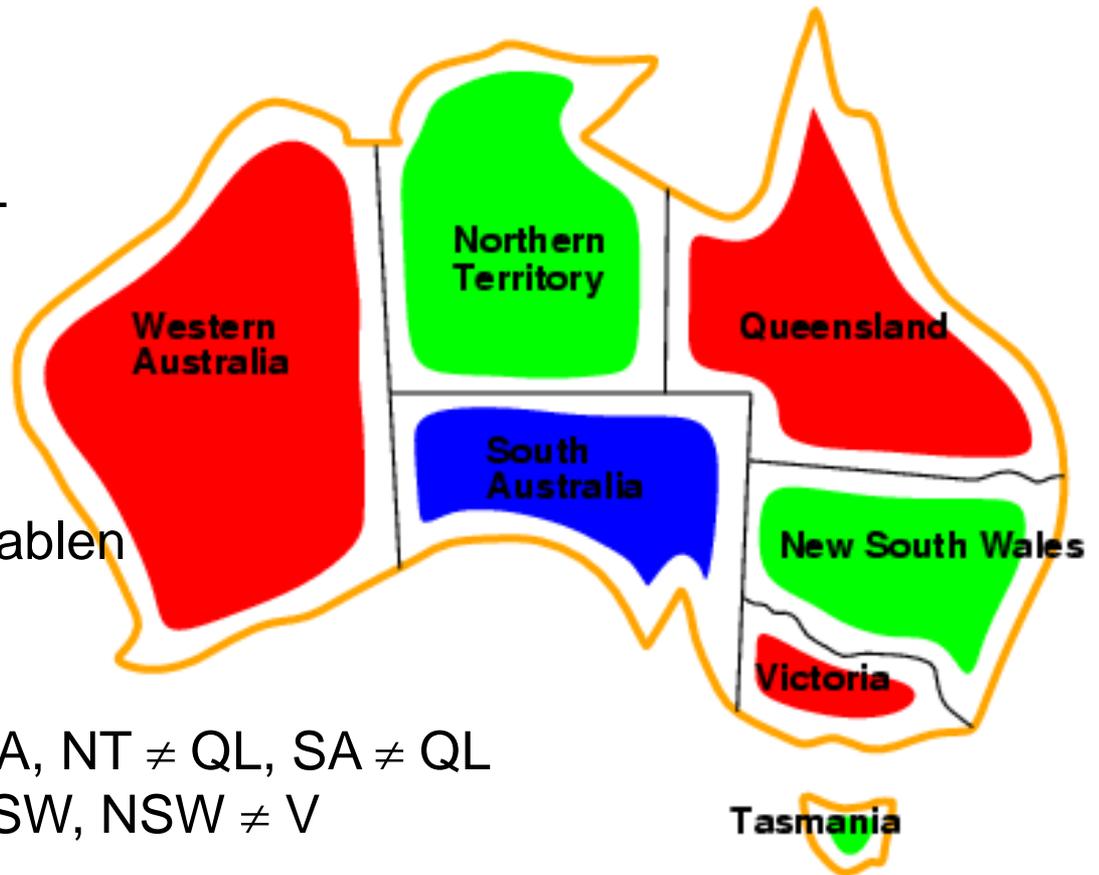
$D_{WA} = \{\text{rot, grün, blau}\}$ und analog für die anderen Variablen

Constraints:

$WA \neq NT$, $WA \neq SA$, $NT \neq SA$, $NT \neq QL$, $SA \neq QL$
 $SA \neq NSW$, $SA \neq V$, $QL \neq NSW$, $NSW \neq V$

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 größeren Tiefe liegt als die optimale Lösung

- Zeitkomplexität:

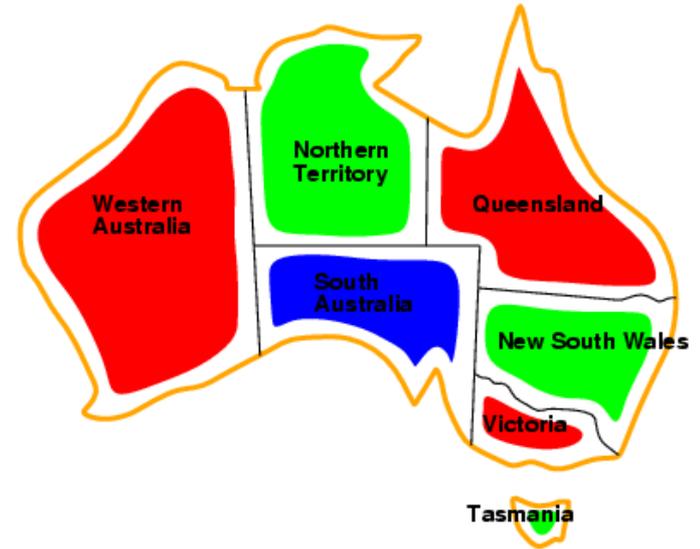
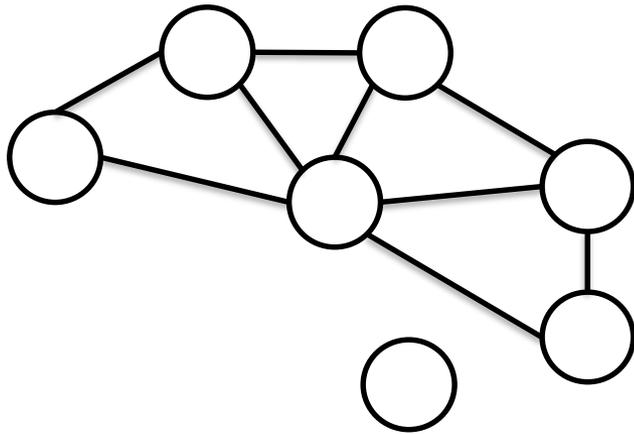
$O(b^m)$: Es müssen (fast) alle möglichen Knoten eines Zweigs expandiert werden, auch wenn keine Lösung enthalten ist.

- Speicherkomplexität:

$O(b \cdot m)$: 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

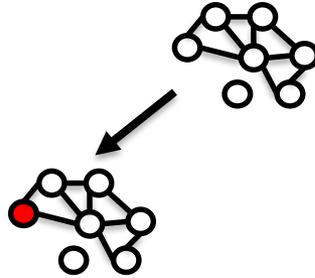
Backtracking Beispiel



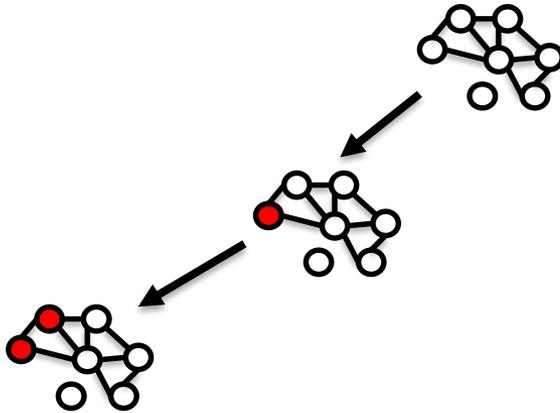
Backtracking Beispiel



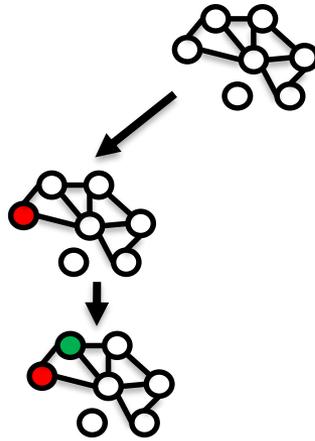
Backtracking Beispiel



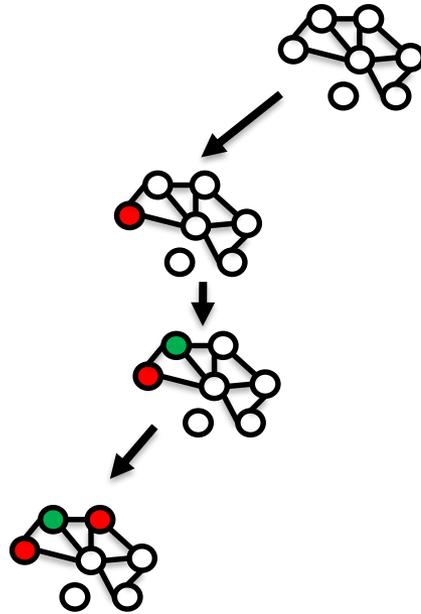
Backtracking Beispiel



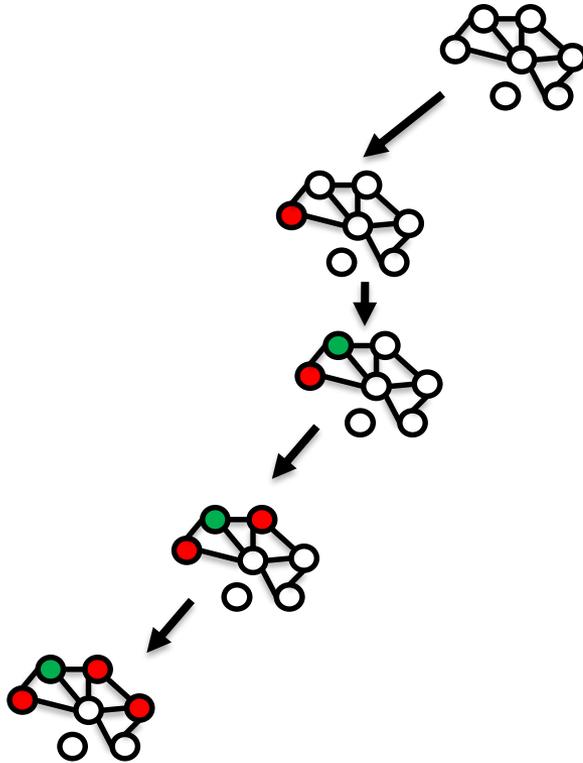
Backtracking Beispiel



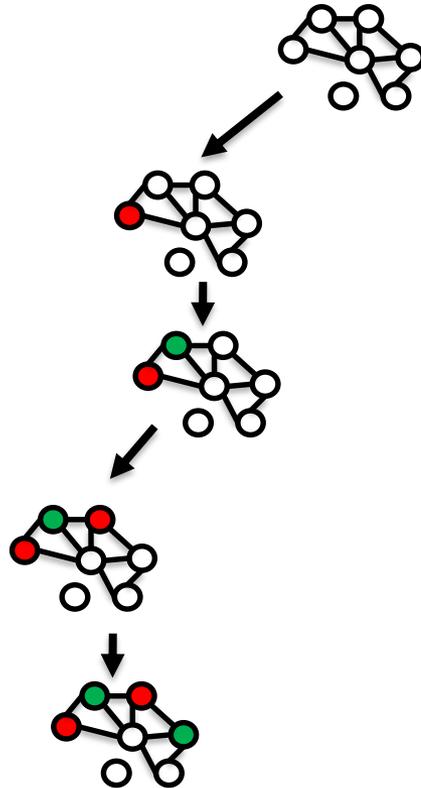
Backtracking Beispiel



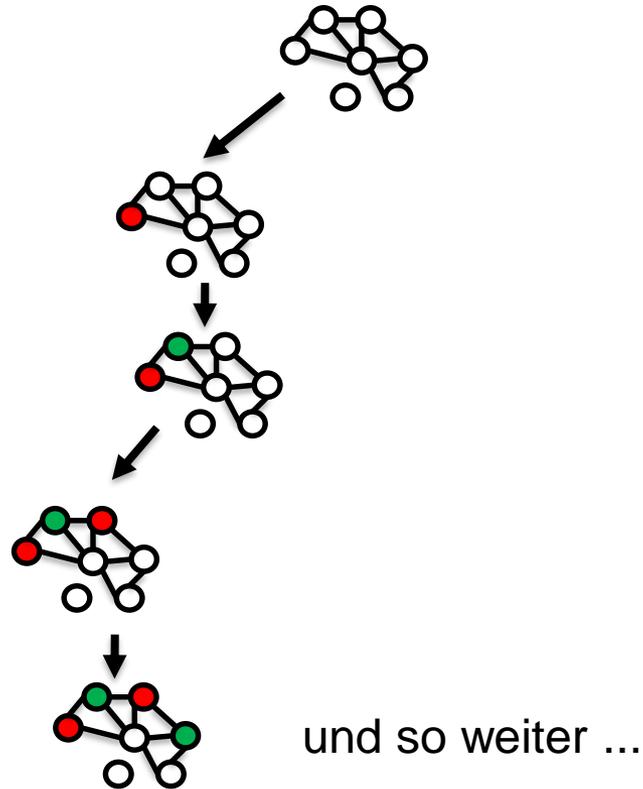
Backtracking Beispiel



Backtracking Beispiel



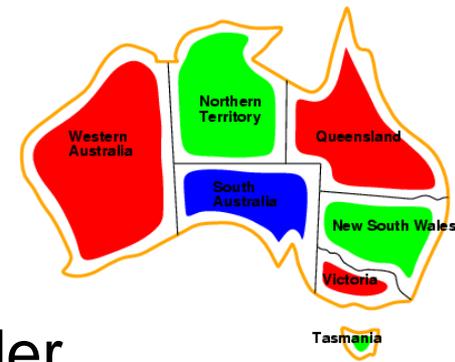
Backtracking Beispiel



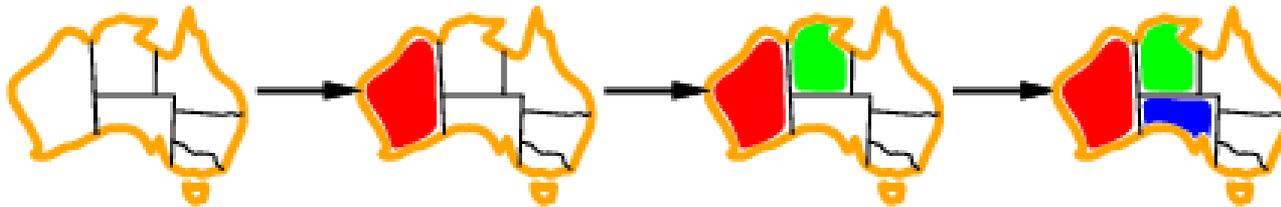
Effizienz von Backtracking

- Backtracking ist im allgemeinen uneffizient:
 - Laufzeitkomplexität: $O(b^m)$:
- 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

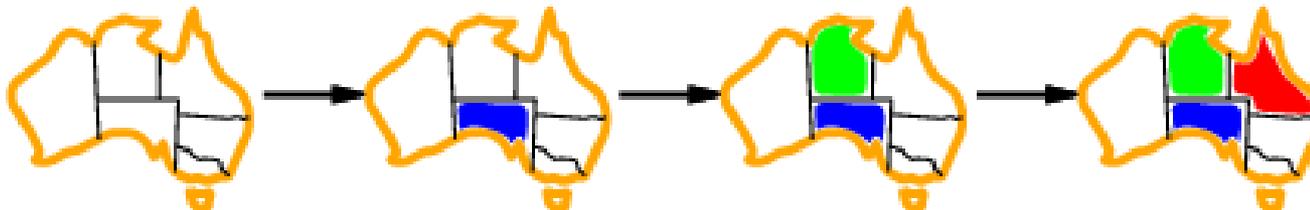
Heuristiken für Variablenwahl



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



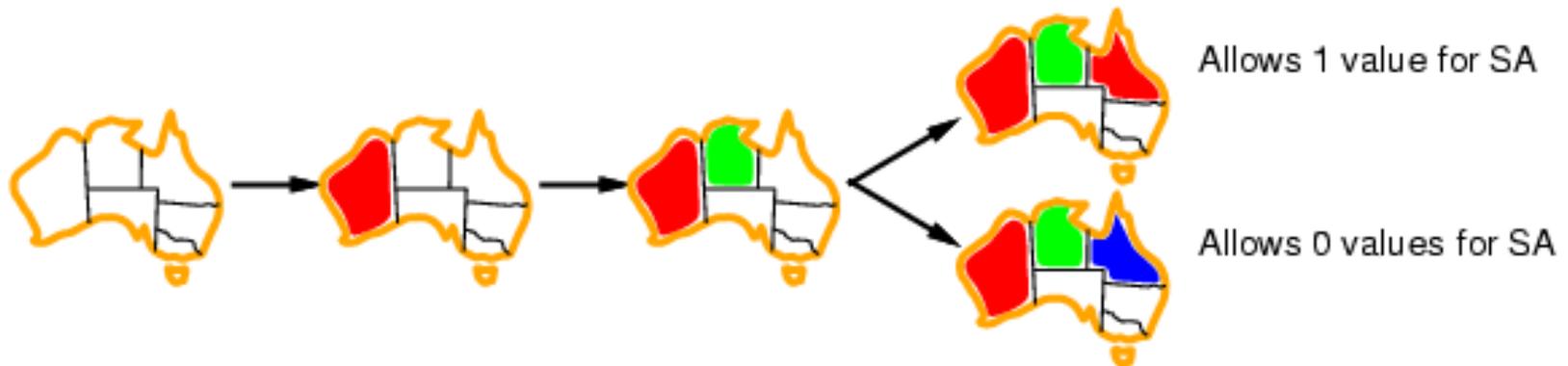
- **Degree Heuristic:** Variable, die in den meisten Constraints anderer, nicht zugewiesener Variablen enthalten ist (SA wird als erstes belegt)



Heuristiken für Wertewahl

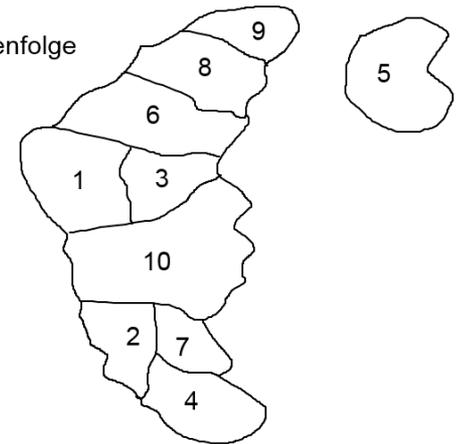


- **Least Constraining Value:** Wert, der andere Variablen am wenigsten einschränkt wird gewählt (NT wird mit rot und nicht blau belegt)



Beispiel

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



- Variablenwahl
 - (1) Degree Heuristic
 - (2) Minimum Remaining Values
- Wertewahl
 - Least Constraining Value
- In welcher Reihenfolge werden die Variablen belegt und welche Werte werden gewählt, wenn (1) vor (2) angewendet wird?

Intelligentes Backtracking I

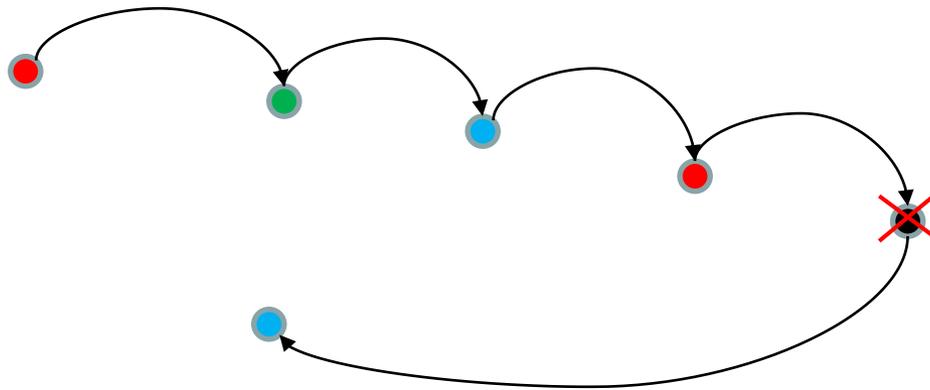
- Grundlegende Annahme:
 - Das Problem liegt bei der Belegung der direkten Vorgängervariablen
 - Es reicht einen Schritt zurück zu gehen, um das Problem zu lösen
- Die grundlegende Annahme ist oft falsch
 - Das eigentliche Problem wurde früher verursacht
 - Idee: Merke dir welche Werte zu einem Ausschluss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

Intelligentes Backtracking II

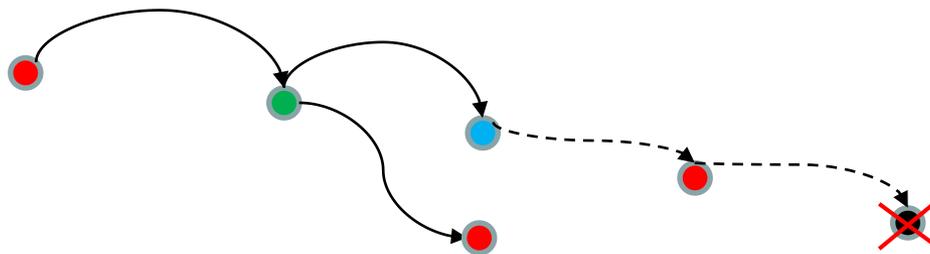
- Grundlegende Annahme:
 - Die Variablen müssen in einer vordefinierten Reihenfolge belegt werden
 - Es macht keinen Unterschied, wie ich diese Reihenfolge wähle
- Die grundlegende Annahme ist oft falsch
 - Aufgrund vorheriger Belegungen kann es Sinn machen, bestimmte Variablen als nächstes zu belegen
 - 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 $\text{conflict}(i)$ für jede Variable $V[i]$:
 - Füge zu $\text{conflict}(i)$ jeden ersten Index j hinzu der einen Wert verboten hat
 - Wenn kein Wert für $V[i]$ mehr verfügbar ist, dann, springe zu $h = \max(\text{conflict}(i))$ zurück
 - Setze in diesem Fall $\text{conflict}(h) := \text{conflict}(h) \cup \text{conflict}(i) - \{h\}$

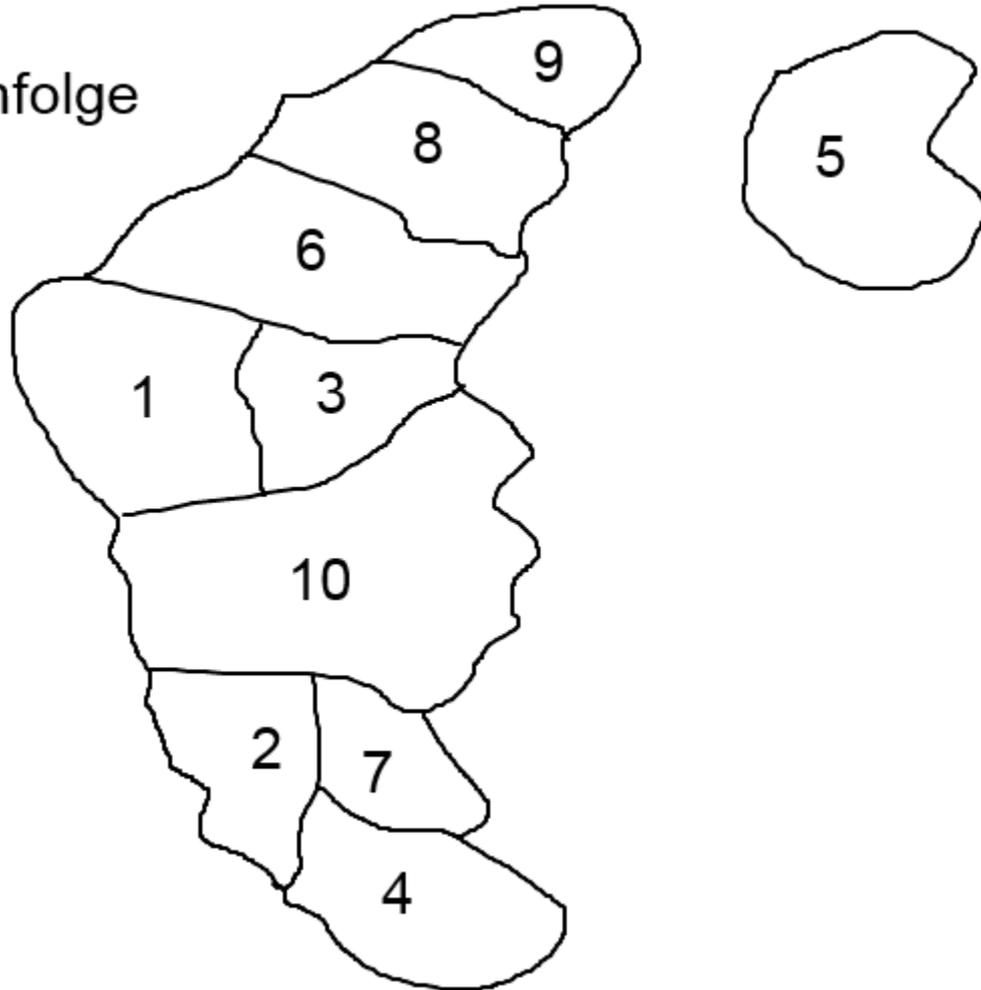
Beispiel

Wertereihenfolge

R (rot)

G (grün)

B (blau)



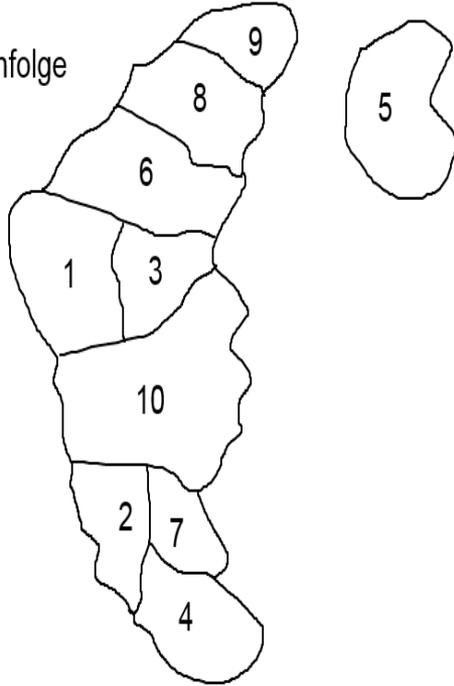
An der Tafel!

Wertereihenfolge

R (rot)

G (grün)

B (blau)



Backjumping (BJ)

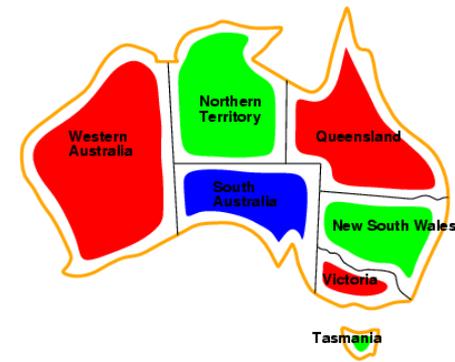
- Immer nur ein Rücksprung, danach normale „Backsteps“
- (Erster) Rücksprung analog wie bei CBJ
- Offensichtlich weniger effektiv

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

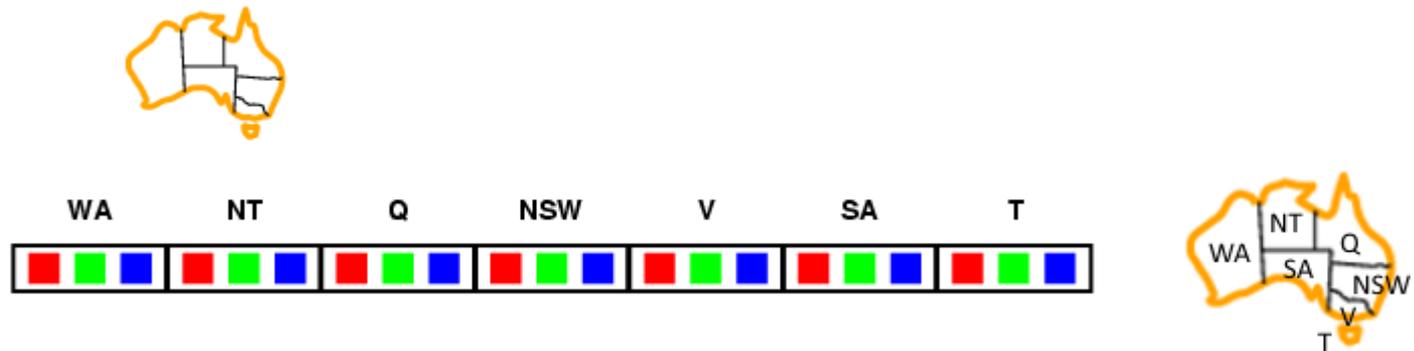
Pruning des Suchraums

- 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

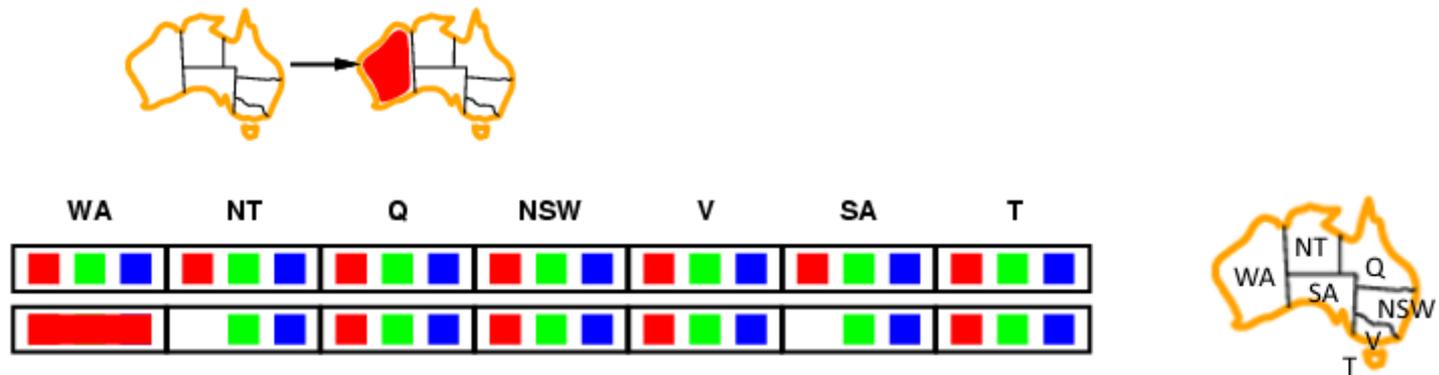


- Forward Checking:
 - Bei jeder neuen Belegung werden nicht mehr mögliche Werte nicht-belegter Variablen aus der entsprechenden Domäne gelöscht



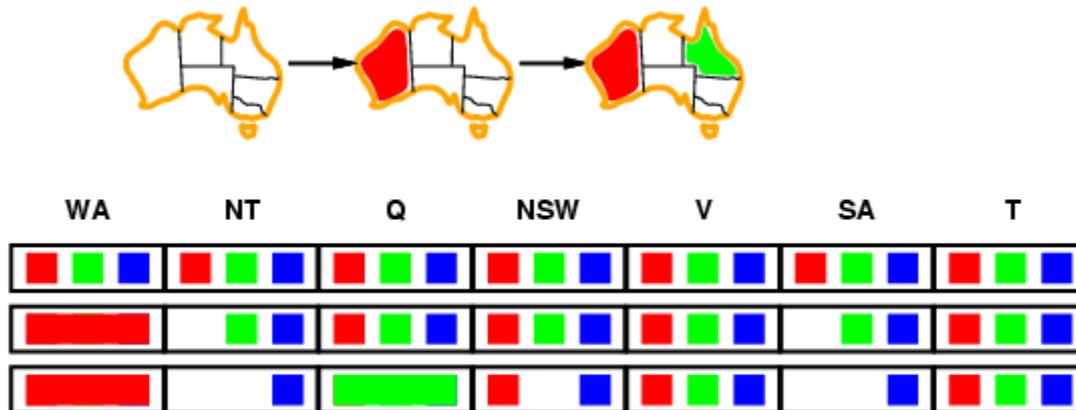
Pruning des Suchraums

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



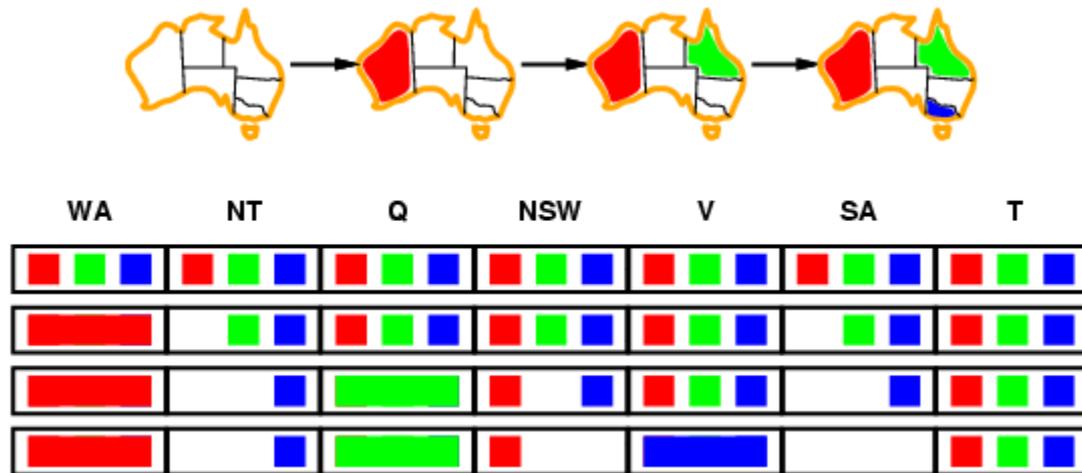
Pruning des Suchraums

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



Pruning des Suchraums

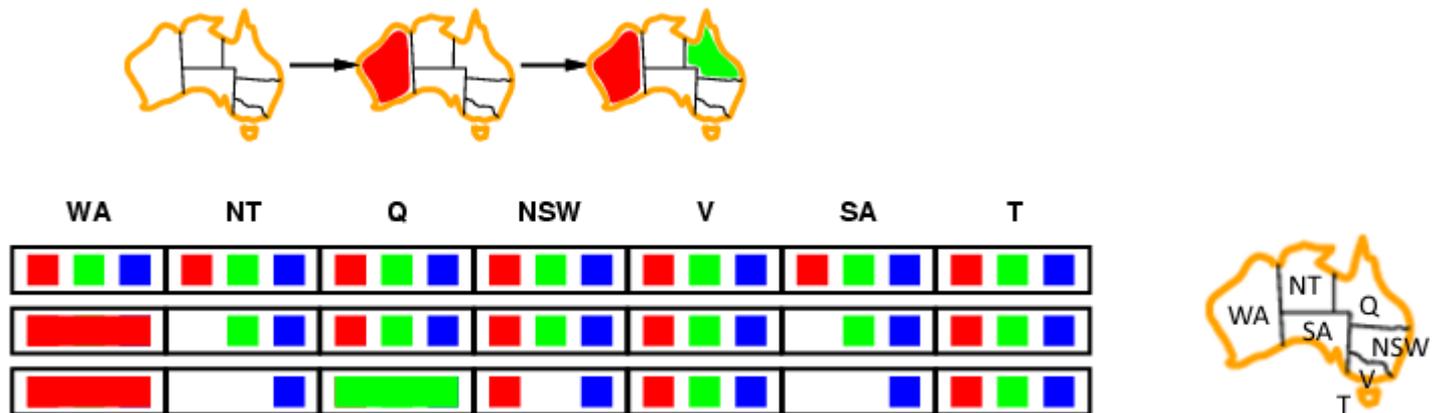
- Forward Checking:
 - Bei jeder neuen Belegung werden nicht mehr mögliche Werte nicht-belegter Variablen aus der entsprechenden Domäne gelöscht.



Ist eine Domäne leer muss zurückgegangen werden.

Erweitertes Pruning

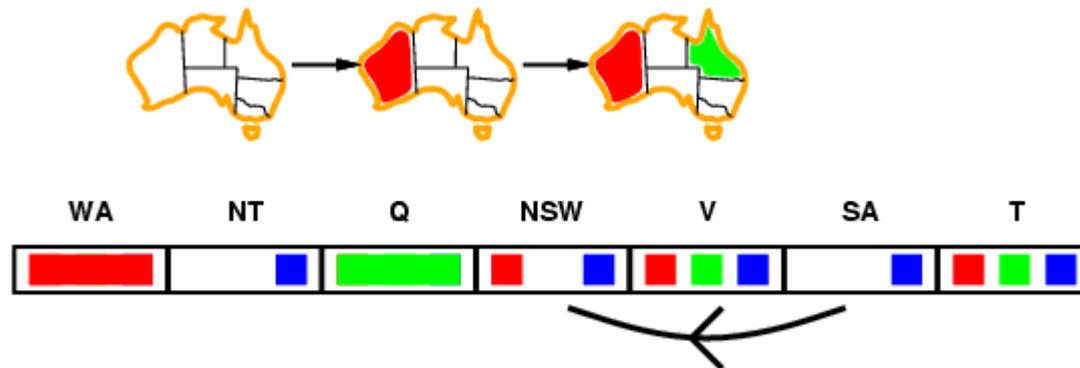
- Forward Checking findet nicht alle möglichen Konflikte. Beispiel:



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

Constraint Propagation

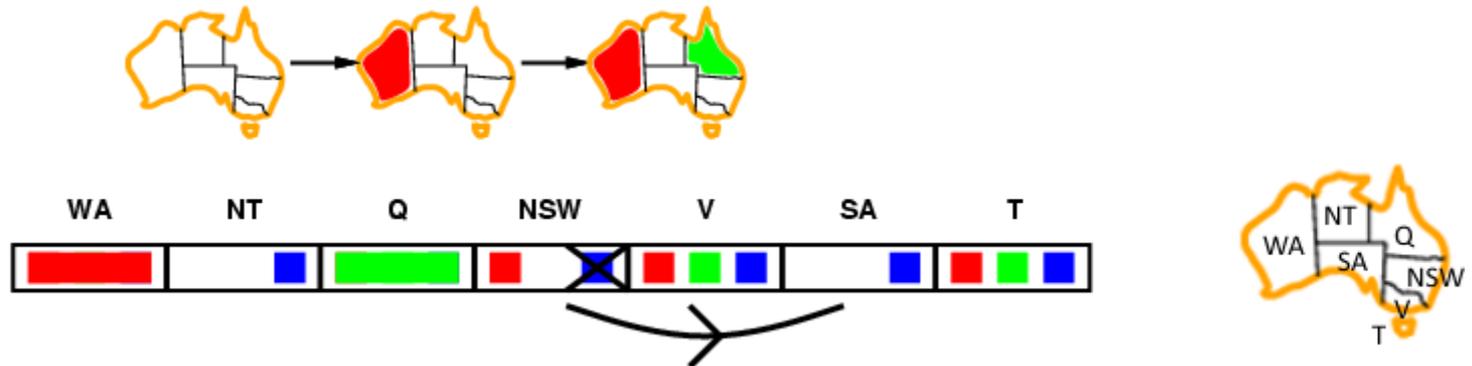
- Idee: schon vor weiteren Belegungen mögliche Auswirkungen aller Constraints berechnen:



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

Constraint Propagation

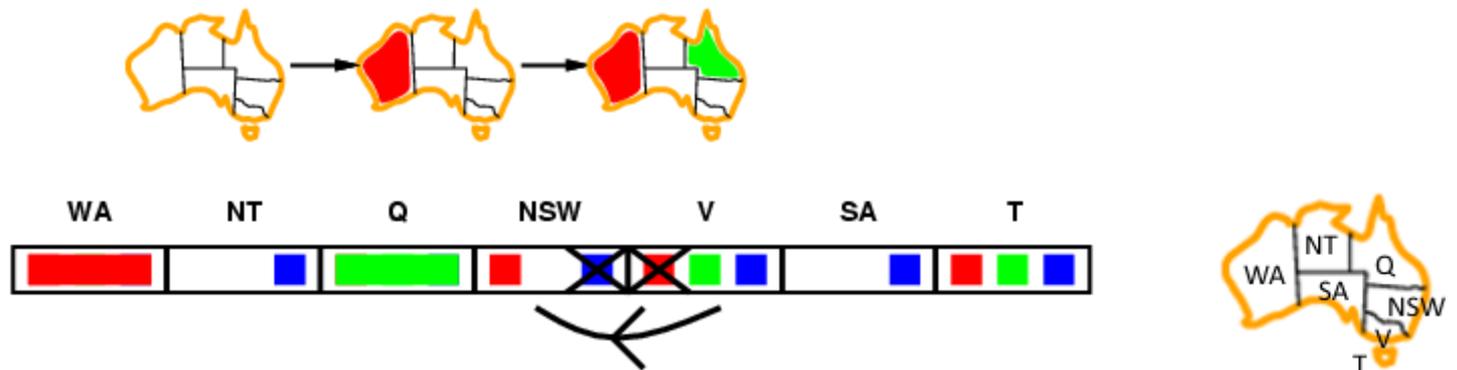
- Idee: schon vor weiteren Belegungen mögliche Auswirkungen aller Constraints berechnen:



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

Constraint Propagation

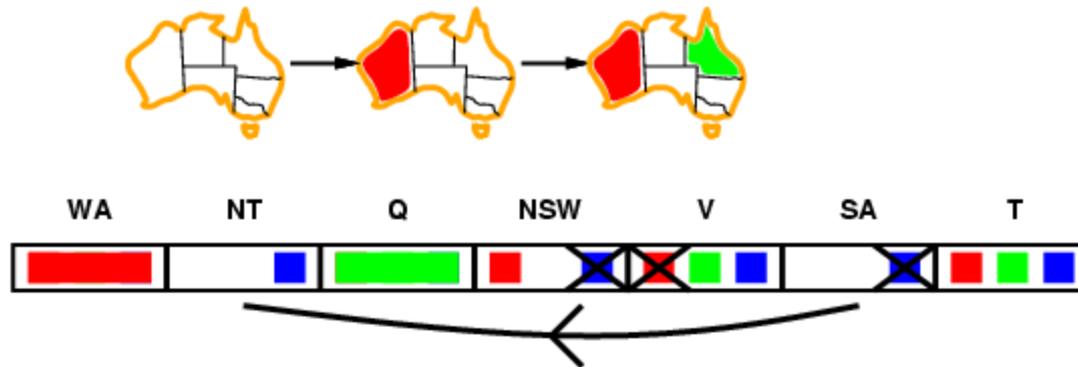
- Idee: schon vor weiteren Belegungen mögliche Auswirkungen aller Constraints berechnen:



- 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