

Seminar: Künstliche Intelligenz

Constraintsysteme

25.05.2005

Stefan Schmidt (ii5558)

Inhalt

1. Einführung
2. Grundlegendes über Constraints
3. Lösen von CSP
4. Constraint-Hierarchien
5. Constraint Logic Programming
6. Zusammenfassung
7. Quellen

1. Einführung

- Aufgabe:

Es stehen drei Farben zur Auswahl, um die einzelnen Bundesstaaten von Australien einzufärben. Zwei benachbarten Bundesstaaten darf jedoch nicht dieselbe Farbe zugewiesen werden.



formaler Ansatz:

Variablen:

Wertebereich:

Einschränkungen:

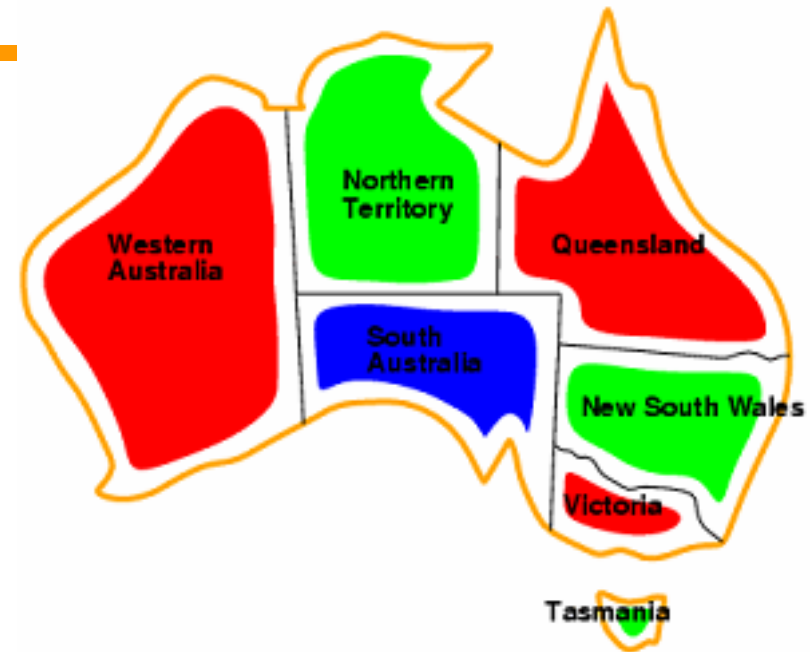
WA, NT, SA, Q, NSW, V, T

{rot, grün, blau}

$WA \neq SA \wedge WA \neq NT \wedge NT \neq SA \wedge$
 $NT \neq Q \wedge Q \neq SA \wedge Q \neq NSW \wedge$
 $NSW \neq SA \wedge NSW \neq V \wedge V \neq SA$

1. Einführung

- Aufgabe:
Es stehen drei Farben zur Auswahl, um die einzelnen Bundesstaaten von Australien einzufärben. Zwei benachbarten Bundesstaaten darf jedoch nicht dieselbe Farbe zugewiesen werden.



formaler Ansatz:

Variablen:

Wertebereich:

Einschränkungen:

WA, NT, SA, Q, NSW, V, T

{rot, grün, blau}

$WA \neq SA \wedge WA \neq NT \wedge NT \neq SA \wedge$
 $NT \neq Q \wedge Q \neq SA \wedge Q \neq NSW \wedge$
 $NSW \neq SA \wedge NSW \neq V \wedge V \neq SA$

1. Einführung

- Wo werden Constraintsysteme in der Praxis eingesetzt?
 - Künstliche Intelligenz
 - Zeitplanung
 - Operations Research
 - Datenbanken
 - Gen-Technik
 - Bildverarbeitung
 - Hardware-Tests
 - Elektrotechnik
 - ...

...meistens aber keine Rede von „Constraint“

2. Grundlagen - Constraint / Constraintsystem

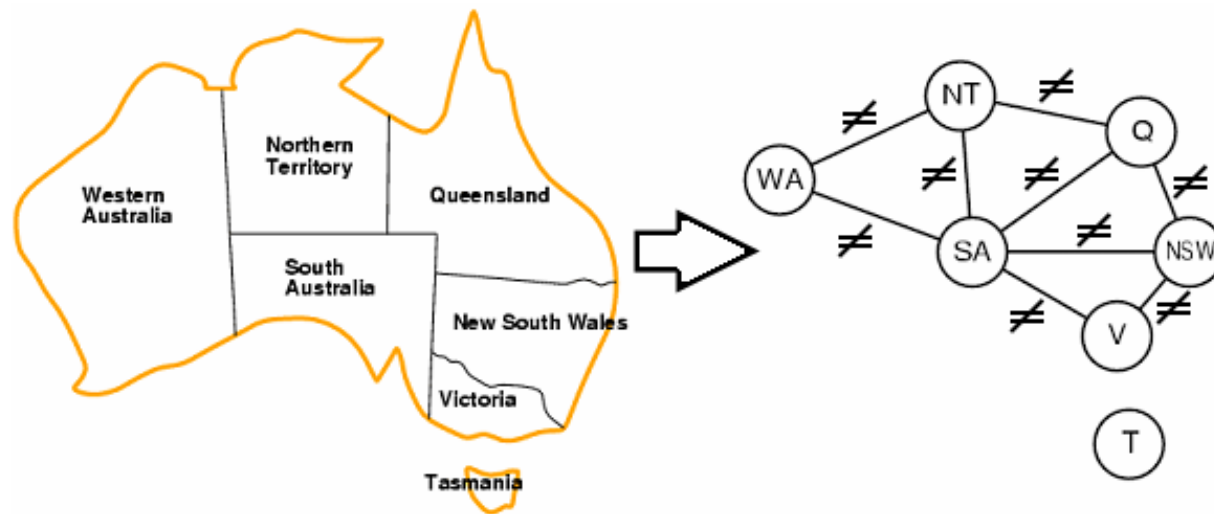
- Was ist ein Constraint?
 - eine Randbedingung auf einer Menge von Variablen
 - drei Komponenten:
 - Menge von Variablen
 - Wertebereich (Domain) für jede Variable
 - entscheidbare Relation
 - Beispiel: $x \in R, x < 4$
- Was ist ein Constraintsystem?
 - eine Menge von Randbedingungen auf einer Menge von Variablen
 - Menge von Constraints
 - Beispiel: $x, y \in R, x < 4 \wedge y > 2 \wedge x + y = 6$

2. Grundlagen – Begriffe

- Stelligkeit:
 - Anzahl der Variablen eines Constraints
- unärer Constraint:
 - nur eine Variable (Bsp.: $x < 4$)
- binärer Constraint:
 - zwei Variablen (Bsp.: $x = 3 - y$)
- binäres Constraintsystem:
 - besteht nur aus unären oder binären Constraints
 - kann als Constraintgraph dargestellt werden

2. Grundlagen – Begriffe

- Constraintgraph:
 - Knoten: Variablen
 - Kanten: Relationen



- bei Systemen mit höherer Stelligkeit: Hypergraph

2. Grundlagen – Begriffe

- Variablenbelegung:
 - Variablen werden Werte zugewiesen
- Erfüllung eines Constraints:
 - die aktuelle Belegung steht nicht im Widerspruch zum Constraint
- Lösung eines Constraintsystems:
 - die aktuelle Belegung erfüllt alle Constraints des Systems

2. Grundlagen - Constraint-Satisfaction-Problem (CSP)

- Zwei wichtige Aufgabenstellungen:
 - Gibt es eine Lösung für das Constraintsystem?
 - Ausgabe einer Lösung, falls eine existiert.
- Gesucht:
 - Algorithmen, um Constraintsysteme zu lösen (Solver)

3. Lösen von CSP – Eigenschaften von Solvern

- zwei Kategorien:
 - vollständig (lösbar/ unlösbar)
 - unvollständig (lösbar/ unlösbar / keine Aussage)
- „well-behaved“:
 - unabhängig von Reihenfolge der einzelnen Constraints (set-based)
 - nicht abhängig von Variablennamen (name independent)
 - \wedge -Verknüpfung von Constraints beachten (monotonic)
 - Beispiel: wenn C_1 unerfüllbar, dann auch $C_1 \wedge C_2$
- vollständige Solver sind immer „well-behaved“

3. Lösen von CSP – CSP-Algorithmen

- für Constraintsysteme, bei denen die Wertemengen der Variablen nicht endlich sind:
 - einen allgemeinen (vollständigen) Algorithmus gibt es nicht ☹
 - aber: „Spezialalgorithmen“
 - dann aber Einschränkungen an die zulässigen Relationen (z.B. nur Variablen in erster Potenz)
 - Beispiele:
 - Gauß-Algorithmus für LGS
 - Simplex für lineare Ungleichungssysteme
 - an Problem angepasst, daher oftmals gute Laufzeit
 - Constraint-Solving

3. Lösen von CSP – CSP-Algorithmen

- für Constraintsysteme, bei denen die Wertemengen der Variablen endlich sind:
 - es gibt immer einen vollständigen Algorithmus 😊
 - CSP ist NP-vollständig
 - im Worst-Case exponentielle Laufzeit
 - aber: ebenfalls „Spezialalgorithmen“
 - Einschränkungen bei Relationen
 - bessere Laufzeit

3. Lösen von CSP – Konsistenz

- Idee:
 - Alle Werte aus den Wertemengen der Variablen, für die das Constraintsystem nicht erfüllbar ist, werden entfernt.
- Knotenkonsistenz (node consistency):
 - nur unäre Constraints werden betrachtet
 - Beispiel:
 - $(x \neq 4) \wedge (x + y = 2) \wedge (y < 2)$, mit $x, y \in \{1, 2, 3, 4\}$
 - nicht knotenkonsistent
 - $(x \neq 4) \wedge (x + y = 2) \wedge (y < 2)$, mit $x \in \{1, 2, 3\} \wedge y \in \{1\}$
 - knotenkonsistent
 - linearer Aufwand $O(nd)$

3. Lösen von CSP – Konsistenz

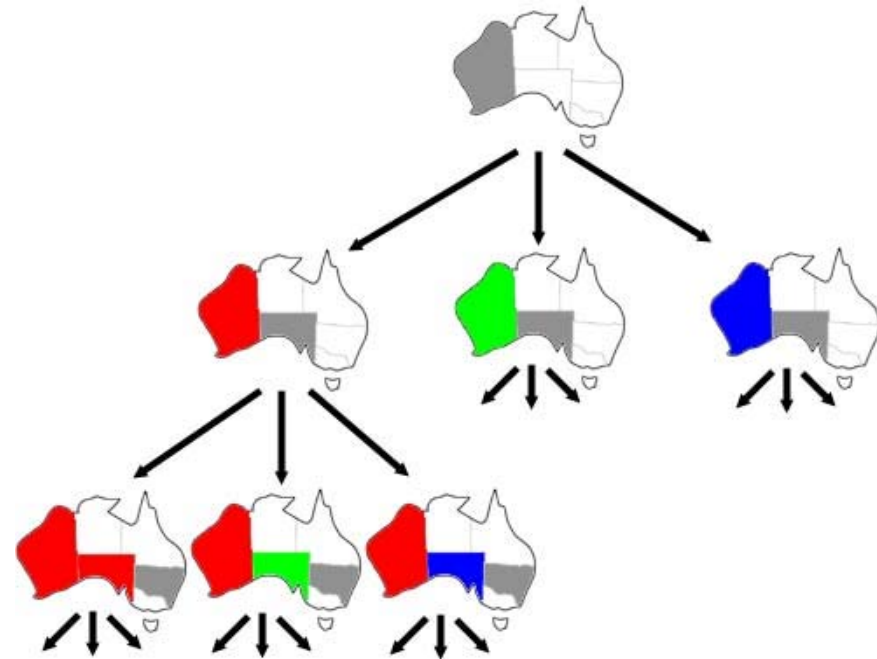
- Kantenkonsistenz (arc consistency):
 - nur binäre Constraints werden betrachtet
 - Beispiel:
 - $(x < y) \wedge (y < z)$ mit $x, y, z \in \{1,2,3,4\}$
 - nicht kantenkonsistent
 - $(x < y) \wedge (y < z)$ mit $x \in \{1,2\} \wedge y \in \{2,3\} \wedge z \in \{3,4\}$
 - kantenkonsistent
 - Nachteil: es wird immer nur ein Constraint betrachtet
 - Beispiel:
 - $(x \neq y) \wedge (y \neq z) \wedge (x \neq z)$ mit $x, y, z \in \{1,2\}$
 - kantenkonsistent, trotz offensichtlichem Widerspruch
 - quadratischer Aufwand $O(n^2d^2)$

3. Lösen von CSP – Konsistenz

- k-Konsistenz:
 - Constraints der Stelligkeit k werden behandelt
 - exponentieller Aufwand
 - keine praktische Relevanz
- unvollständiger Lösungsalgorithmus für CSP:
 - Anwendung von Knoten- und Kantenkonsistenz
 - unlösbar: wenn die Wertemenge einer Variablen leer ist
 - lösbar: wenn jede Wertemenge nur noch einen Wert enthält

3. Lösen von CSP – Generate and Test - Algorithmus

- Idee:
 - Generierung und Testen von Belegungen, solange bis Lösung gefunden oder alle Wertekombinationen ausprobiert
- Visualisierung:
 - durch Belegungsbaum
 - Tiefensuche vorteilhaft
- kein guter Lösungsweg:
 - wenn keine Lösung, kompletten Baum durchlaufen
 - exponentieller Aufwand $O(d^n)$



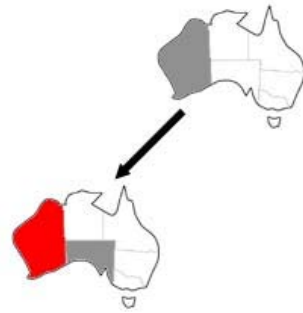
3. Lösen von CSP – Backtracking - Algorithmus

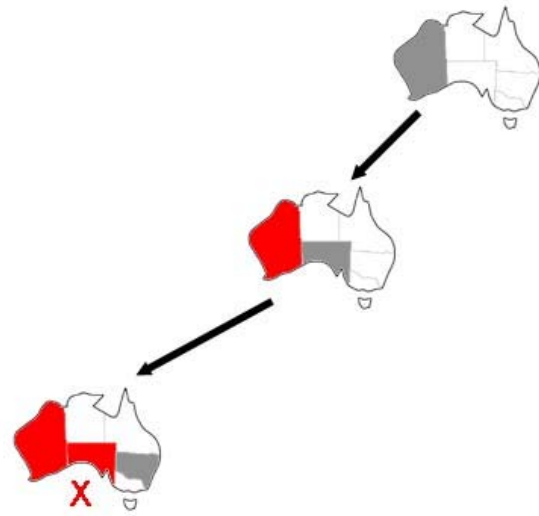
- Idee:
 - inkonsistente Belegungen früh erkennen
 - Anzahl der zu testenden Belegungen reduzieren
- Vorgehen:
 - nacheinander wird jeder Variablen ein Wert zugeordnet
 - nach einer Zuweisung wird geprüft, ob die aktuelle Belegung Inkonsistenzen hervorruft
 - ist dies der Fall, so wird der Wert der zuletzt belegten Variablen verworfen und ein anderer zugewiesen

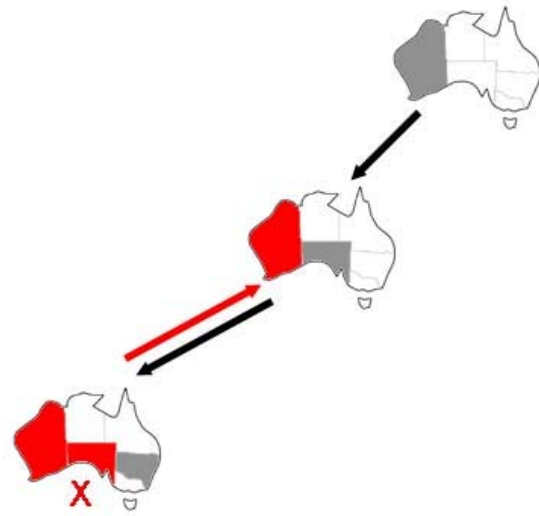
Beispiel für Backtracking

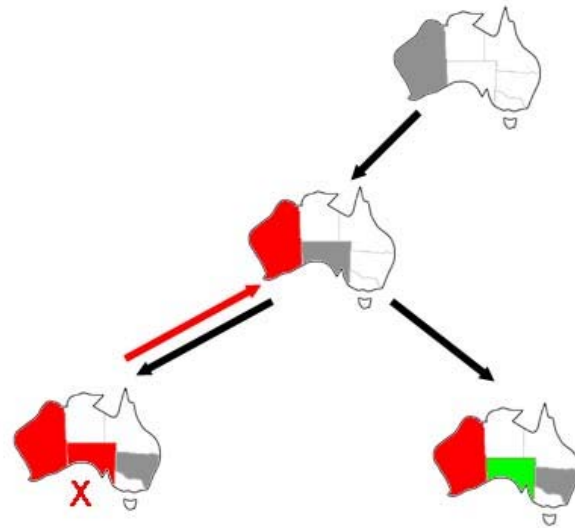


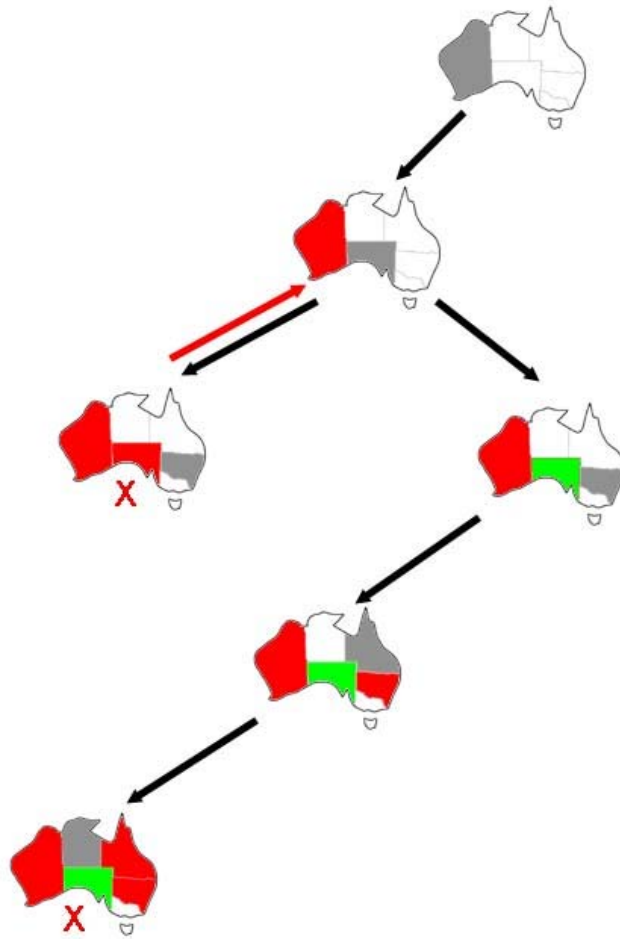
- Reihenfolge der Variablen: WA, SA, NSW, Q, NT, V, T
- Reihenfolge der Werte: rot, grün, blau

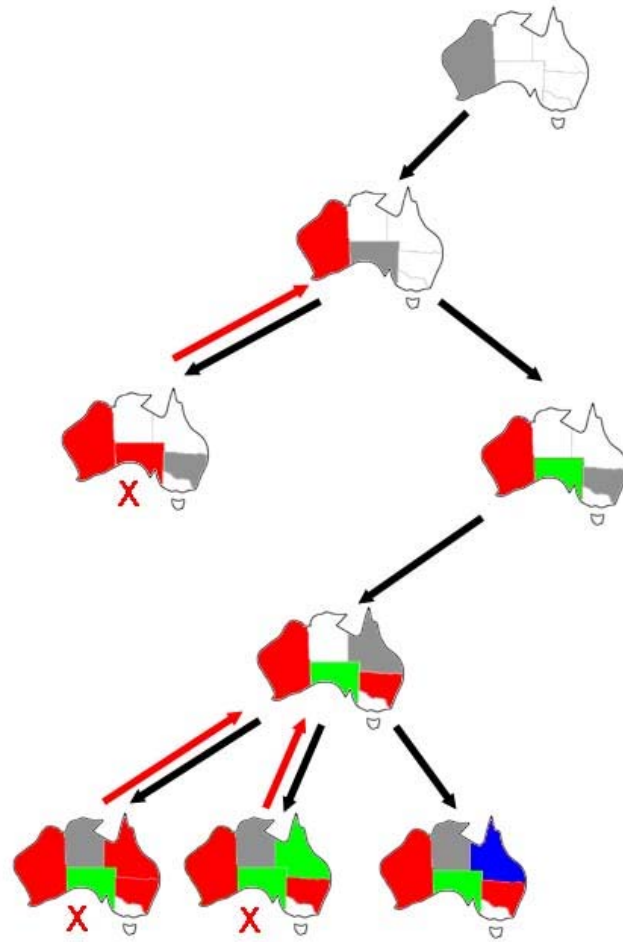


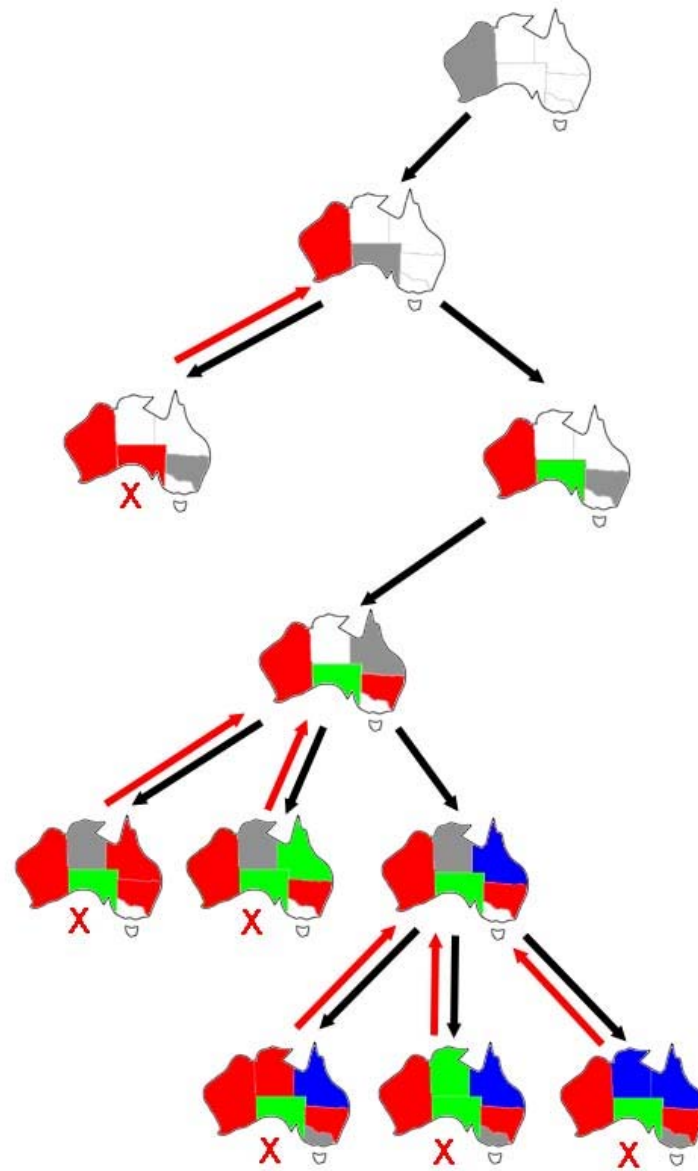


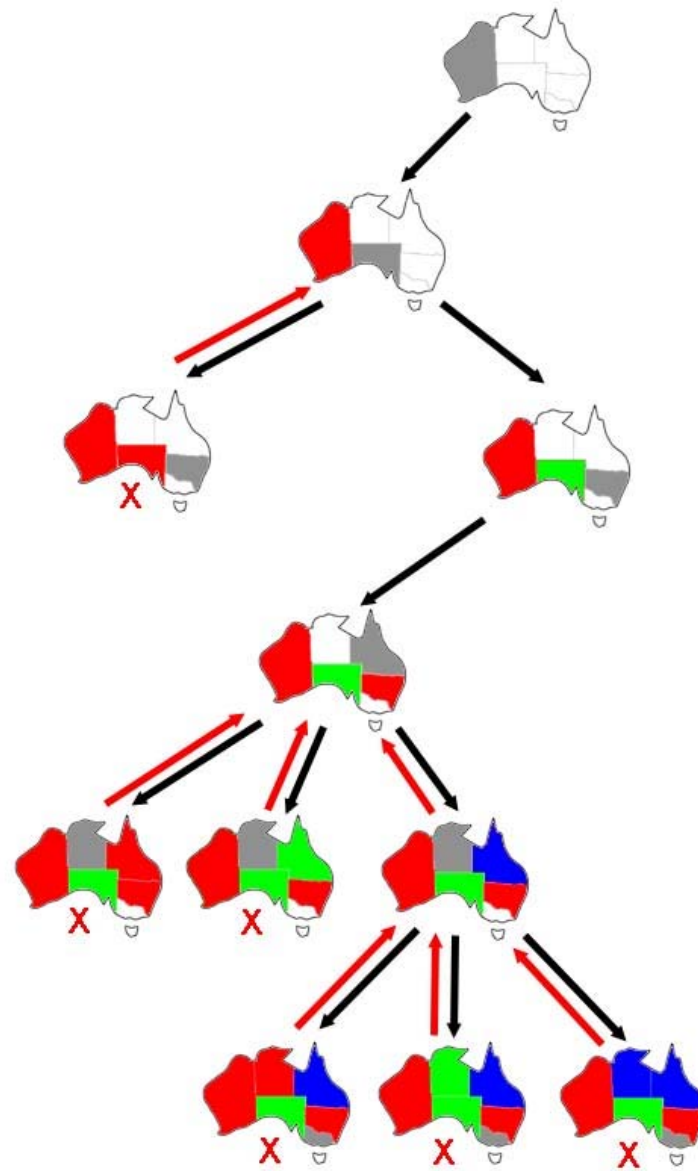


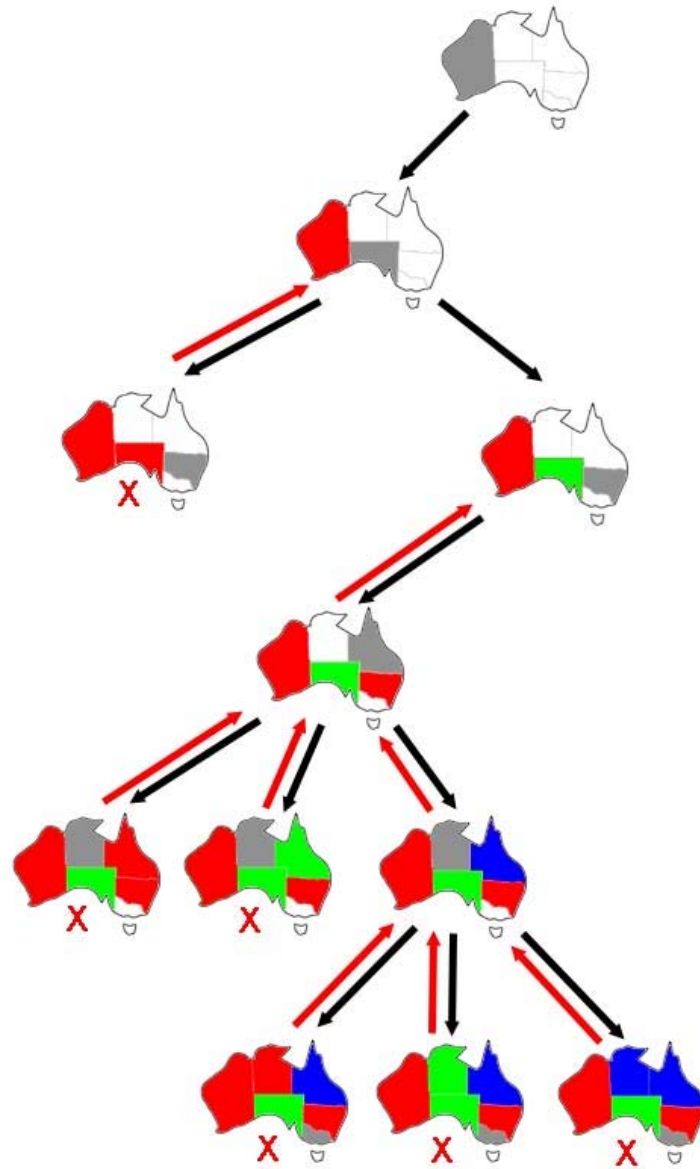


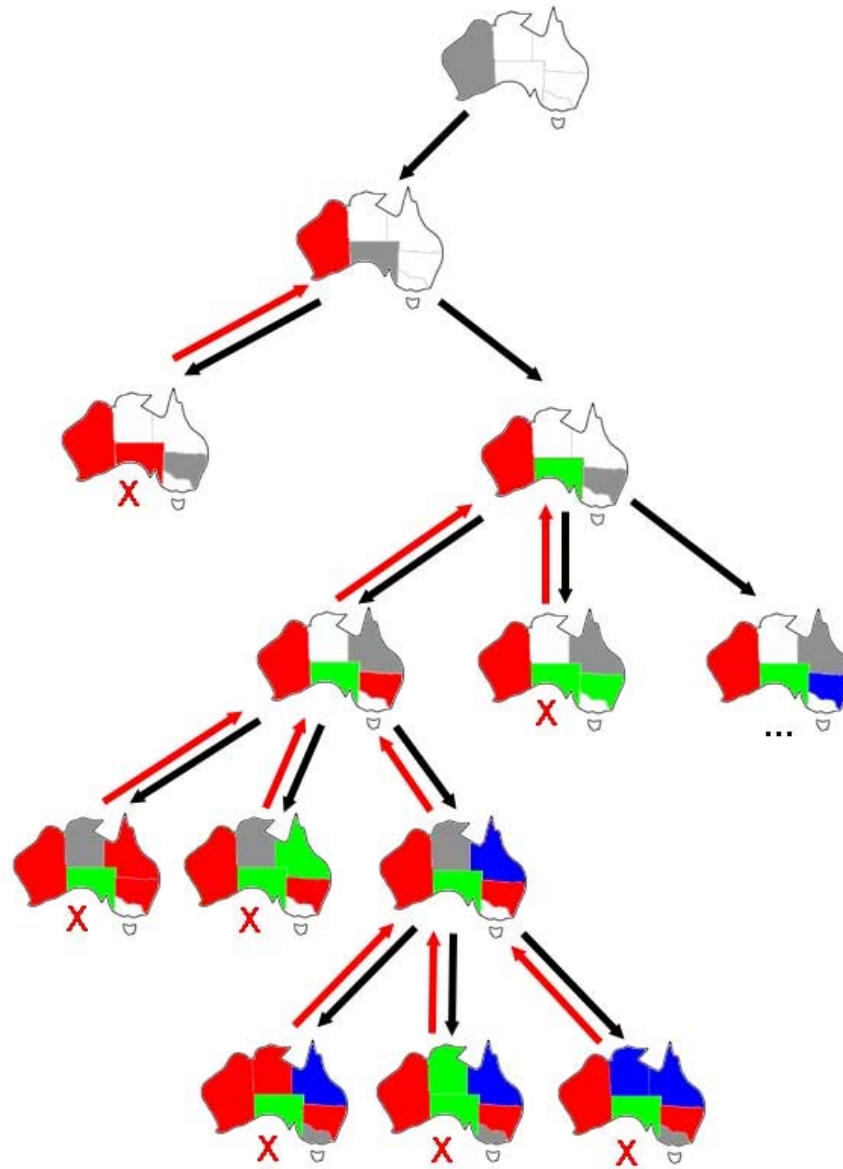












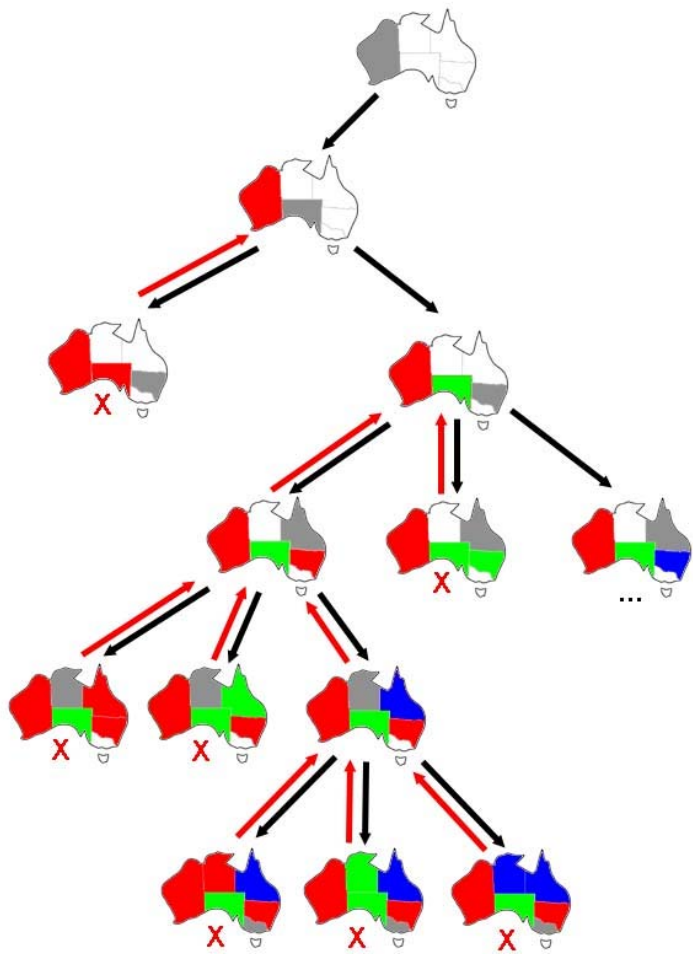
3. Lösen von CSP – Heuristiken für Backtracking

- Variablen-Ordnungsheuristiken:
 - Minimum-Remaining-Values (MRV):
 - die Variable mit den wenigsten zulässigen Werten ist auszuwählen
 - Maximum-Degree-Heuristic:
 - die Variable ist auszuwählen, die am meisten in Constraints vorkommt, in denen Variablen noch nicht festgelegt sind
- Werte-Ordnungsheuristiken:
 - Least-Constraining-Value:
 - Der Wert mit den wenigsten folgenden Einschränkungen ist zu nehmen.

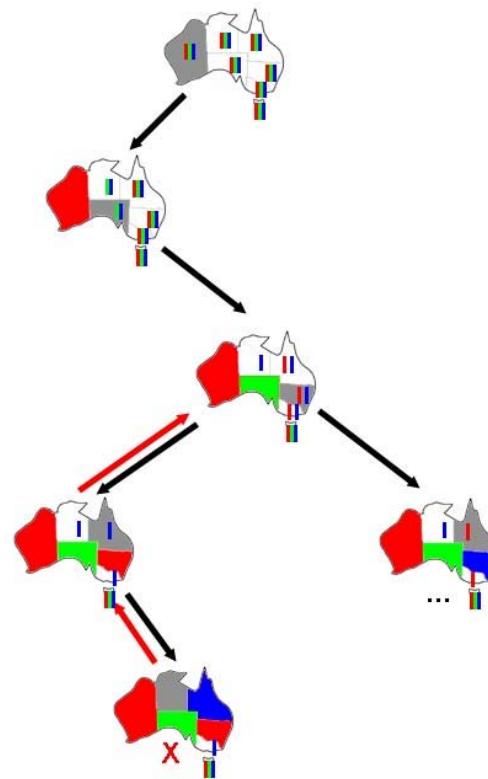
3. Lösen von CSP - Backtracking mit Forwardchecking

- entfernen aller inkonsistenten Werte aus den Wertemengen aller Variablen, die mit der zuletzt instanziierten durch ein Constraint verbunden sind
- nach jedem Instanzierungsschritt
- meist in Kombination mit MRV-Heuristik

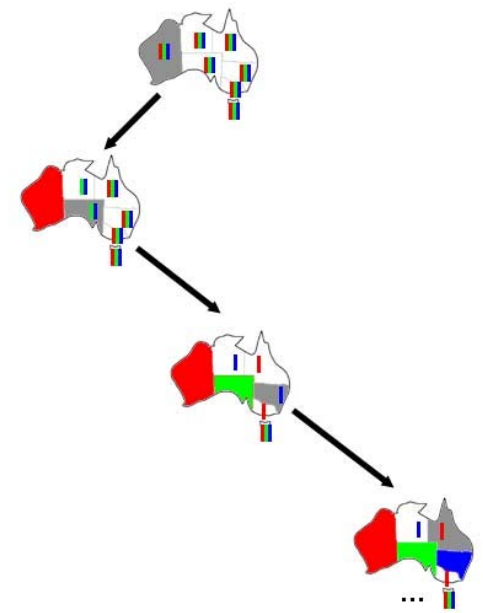
- Verbesserung:
 - Herstellen von Kantenkonsistenz nach jedem Instanzierungsschritt (MAC-Algorithmus)
 - weniger Suchschritte
 - mehr Rechenaufwand zur Konsistenzherstellung



Backtracking



Forwardchecking



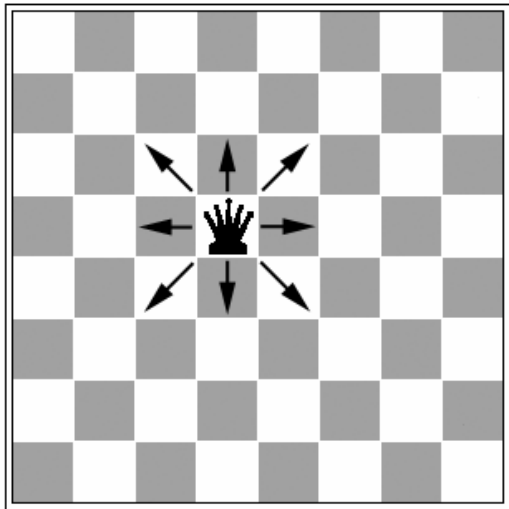
MAC

3. Lösen von CSP – Min-Conflicts - Algorithmus

- stochastische Suche
- Idee:
 - Start mit einer beliebigen Wertebelegung
 - Auswählen von Variablen und Zuweisung neuer Werte, die weniger Konflikte verursachen solange, bis System gelöst
- Vorteile:
 - bei vielen Praxis-Problemen gutes Laufzeitverhalten
 - „Reparieren“ bei kleinen Veränderungen des Systems
- Nachteile:
 - „Hängen bleiben“ in lokalen Minima
 - Gegenmaßnahmen: Random-Walk, Tabu-Liste, ...

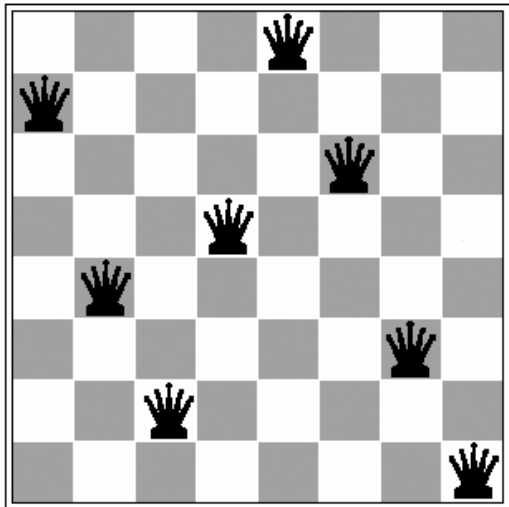
3. Lösen von CSP – Min-Conflicts - Algorithmus

- Beispiel für Min-Conflicts-Vorgehen:



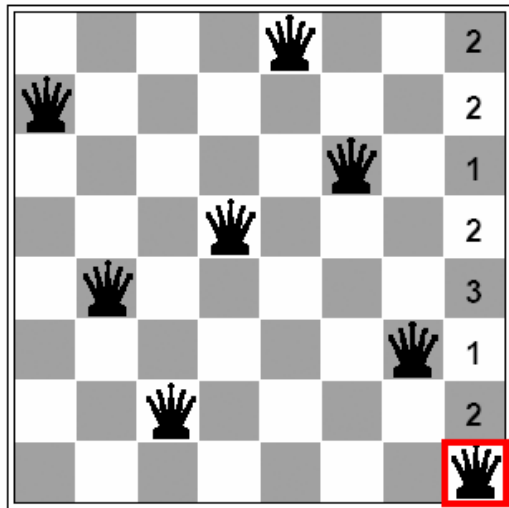
3. Lösen von CSP – Min-Conflicts - Algorithmus

- Beispiel für Min-Conflicts-Vorgehen:



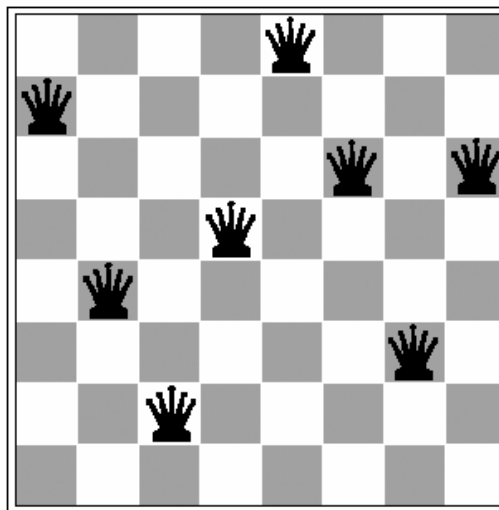
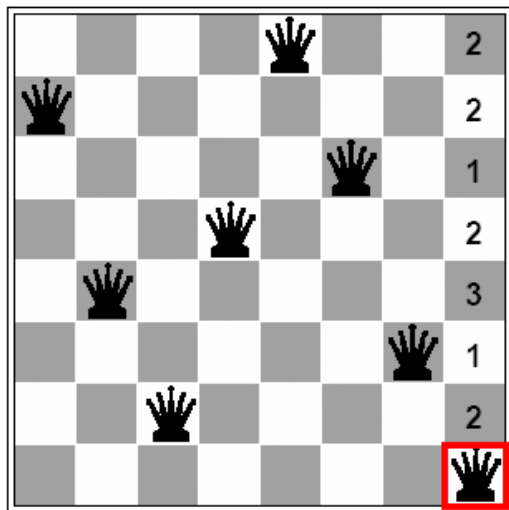
3. Lösen von CSP – Min-Conflicts - Algorithmus

- Beispiel für Min-Conflicts-Vorgehen:



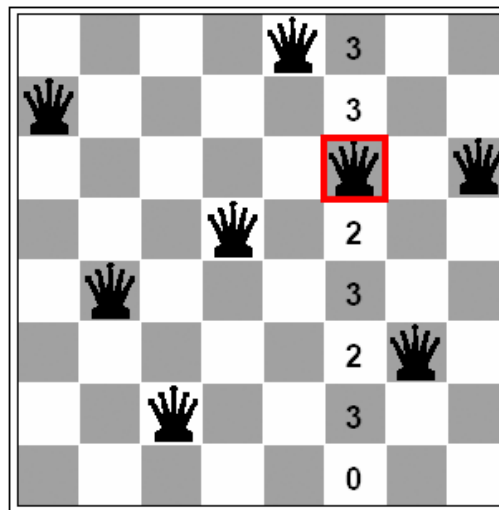
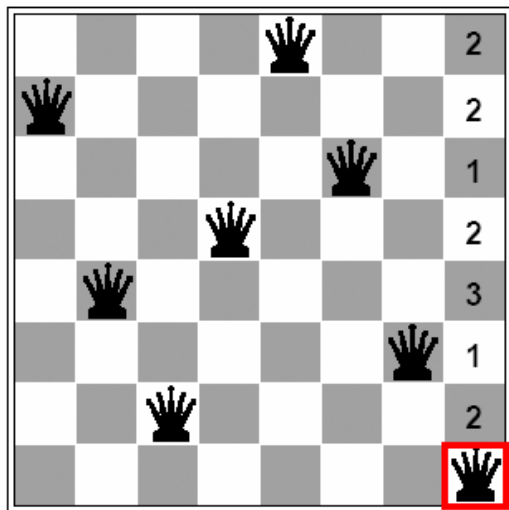
3. Lösen von CSP – Min-Conflicts - Algorithmus

- Beispiel für Min-Conflicts-Vorgehen:



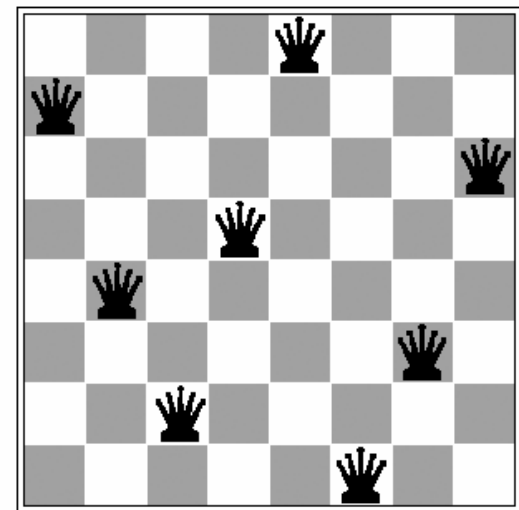
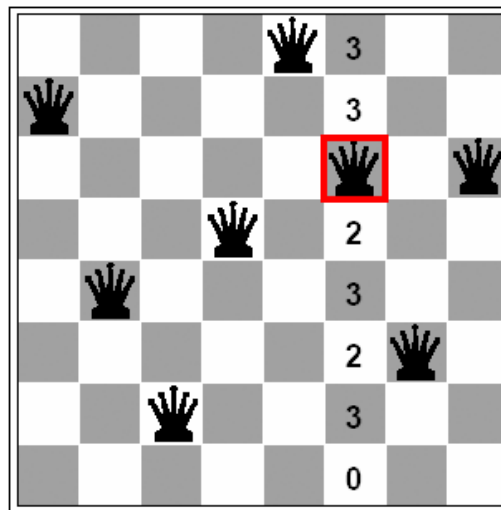
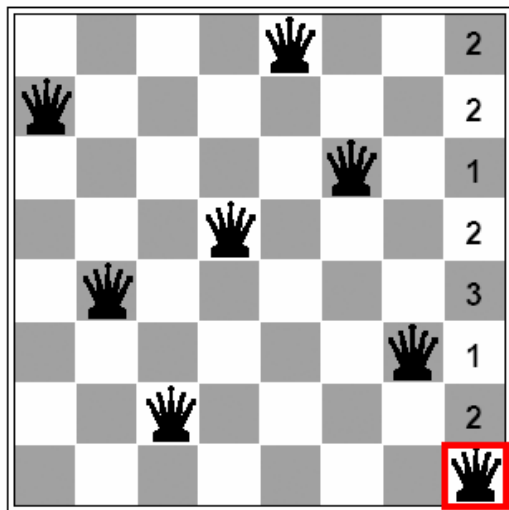
3. Lösen von CSP – Min-Conflicts - Algorithmus

- Beispiel für Min-Conflicts-Vorgehen:



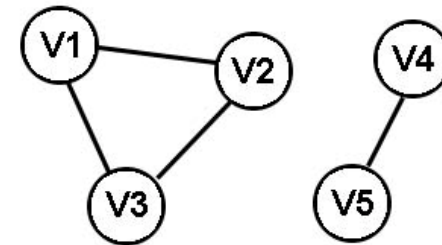
3. Lösen von CSP – Min-Conflicts - Algorithmus

- Beispiel für Min-Conflicts-Vorgehen:

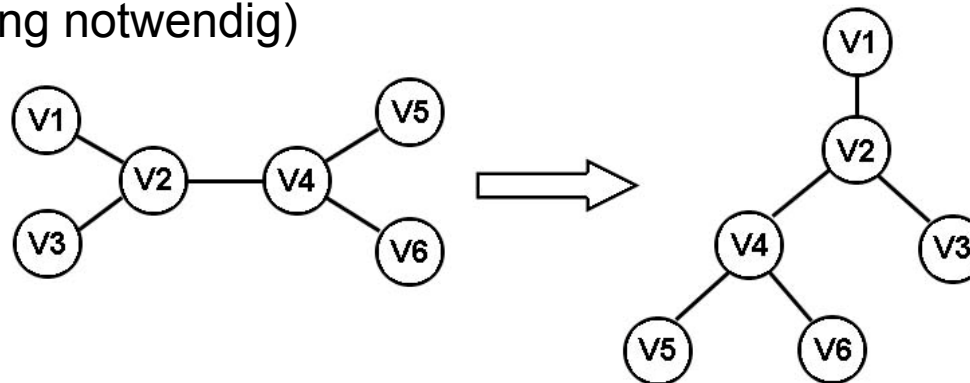


3. Lösen von CSP – Struktur der Constraintgraphen

- zwei nicht verbundene Graphen:
 - unabhängige Teilprobleme, getrennt lösbar



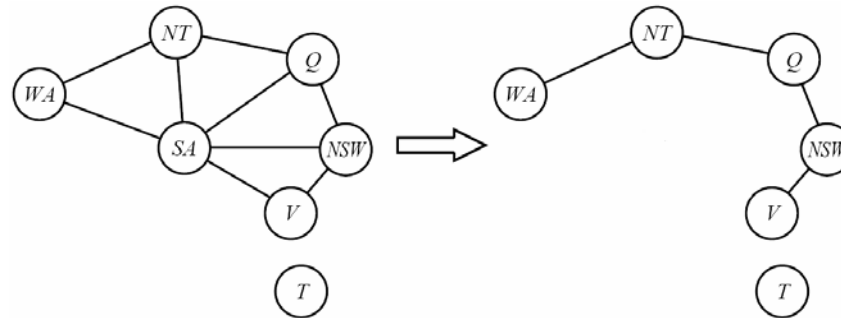
- Graph besitzt Baumstruktur:
 - schneller Lösungsalgorithmus ($O(nd^2)$)
 - Herstellen von Kantenkonsistenz von den Blättern zur Wurzel
 - Zuweisung von Werten von der Wurzel zu den Blättern (kein Backtracking notwendig)



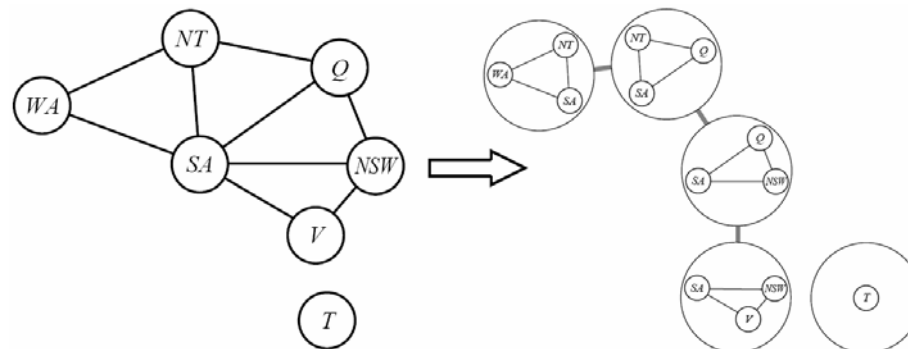
3. Lösen von CSP – Struktur der Constraintgraphen

- wenn Graph keine Baumstruktur:
 - Graphen modifizieren, so dass Baum entsteht

- Knoten entfernen



- Knoten zu Teilproblemen zusammenfassen, die dann untereinander einen Baum bilden



4. Constraint-Hierarchien

- viele Constraintsysteme sind überbestimmt (over-constrained)
- meistens ist man nur daran interessiert, dass ein Teil der Constraints erfüllt ist, die anderen sind optional
- Einführung von Hierarchien
 - das ursprüngliche Constraintsystem wird in mehrere Systeme zerlegt
 - den auf diesen Systemen wird eine Ordnung definiert (Level)
 - System mit Level 0: Constraints, die erfüllt sein müssen
 - Systeme mit Level k sind „wichtiger“ als Systeme mit Level k+1
 - zusätzlich wird noch eine Fehlerfunktion auf den Constraints in einer Hierarchieebene definiert (z.B. erfüllt = 0, nicht erfüllt = 1)

4. Constraint-Hierarchien

- Komparatoren, um Lösungen miteinander zu vergleichen:
 - locally-better
 - globally-better
 - globally-predicate-better
 - ...
- Lösungsalgorithmen:
 - meist nur auf einen Wertebereich / auf eine Vergleichsmethode bezogen
 - DeltaBlue (keine zyklischen Abhängigkeiten)
 - Incremental Hierarchical Constraint Solver (IHCS)

4. Constraint-Hierarchien – IHCS

- Vergleichsoperator: globally-predicate-better
- Constraints werden in drei Mengen aufgeteilt (Konfiguration):
 - AS (active store): alle Constraints, die erfüllt werden
 - RS (relaxed store): alle Constraints, die verletzt werden
 - US (unexplored store): alle noch nicht untersuchten Constraints
 - Schreibweise: <AS•RS•US>

4. Constraint-Hierarchien – IHCS

- Idee:
 - für eine Hierarchie ist die Konfiguration $\langle AS \cdot RS \cdot \{\} \rangle$ bekannt, es gilt dann die Konfiguration $\langle AS \cdot RS \cdot \{c\} \rangle$ zu optimieren
 - Forward-Rule: Constraints werden von US in AS eingefügt
 - Backward-Rule: bei Konflikten in AS werden Constraints zwischen AS und RS ausgetauscht, so dass AS ohne Konflikte und Konfiguration am besten
 - Ende, wenn $\langle AS \cdot RS \cdot \{\} \rangle$

4. Constraint-Hierarchien – IHCS (Beispiel)

Hierarchie:

Name	Constraint	Level
c1	$x + y = 15$	1
c2	$3 * x - y < 5$	1
c3	$x > y + 1$	2
c4	$x < 7$	2

Lösungsweg:

Aktion	Konfiguration	D(X)	D(Y)	Regel
add c1	$\{\} \bullet \{\} \bullet \{c1\}$	1..10	1..10	fw
	$\{c1\} \bullet \{\} \bullet \{\}$	5..10	5..10	
add c2	$\{c1\} \bullet \{\} \bullet \{c2\}$	5..10	5..10	fw
	$\{c1, c2\} \bullet \{\} \bullet \{\}$	-	-	bw
	$\{c1\} \bullet \{c2\} \bullet \{\}$	5..10	5..10	
add c3	$\{c1\} \bullet \{c2\} \bullet \{c3\}$	5..10	5..10	fw
	$\{c1, c3\} \bullet \{c2\} \bullet \{\}$	7..10	5..8	
add c4	$\{c1, c3\} \bullet \{c2\} \bullet \{c4\}$	7..10	5..8	fw
	$\{c1, c3, c4\} \bullet \{c2\} \bullet \{\}$	-	-	bw
	$\{c1, c3\} \bullet \{c2, c4\} \bullet \{\}$	7..10	5..8	

5. Constraint Logic Programming

- in vielen Anwendungen müssen Constraintsysteme gelöst werden
- die Implementierung eigener Constraint-Solver ist nicht leicht
- es existiert eine Vielzahl freier Solver-Systeme, meist auf der Basis von PROLOG
- Programmierer stellt nur Constraintsystem auf, Lösung wird automatisch ermittelt
- Beispiel für CLP-System: ECLiPS^e
(<http://www.icparc.ic.ac.uk/eclipse/>)

5. Constraint Logic Programming

Send-More-Money-Rätsel:

$$\begin{array}{r} \text{SEND} \\ +\text{MORE} \\ \hline =\text{MONEY} \end{array} \quad \longrightarrow \quad \begin{array}{r} 9567 \\ +1085 \\ \hline =10652 \end{array}$$

Beispiel für CLP-Programm unter ECLiSP^e:

```
sendmore(Digits) :-  
    Digits = [S,E,N,D,M,O,R,Y],  
    Digits :: [0..9],  
    alldifferent(Digits),  
    S #\= 0,  
    M #\= 0,  
    1000*S + 100*E + 10*N + D  
    + 1000*M + 100*O + 10*R + E  
    #= 10000*M + 1000*O + 100*N + 10*E + Y,  
    labeling(Digits).
```

6. Zusammenfassung

- viele praktische Probleme lassen sich als CSP darstellen
- unterschiedlichste Bereiche
 - KI / Bildverarbeitung / OR / Gentechnologie / ...
- viele unterschiedliche Lösungsalgorithmen
 - Backtracking / Min-Conflicts / Spezial-Algorithmen / ...
- Constraint-Hierarchien / CLP breites Anwendungsfeld
- Forschung nicht abgeschlossen...

7. Quellen

- Günter Görz / Claus-Rainer Rollinger / Josef Schneeberger: Handbuch der Künstlichen Intelligenz, Oldenbourg 2003 (4. Auflage), ISBN 3-486-27212-8
- Stuart Russell / Peter Norvig: Artificial Intelligence: A Modern Approach, Pearson 2003 (2. Auflage), ISBN 0-13-080302-2
Internet: <http://aima.cs.berkeley.edu/> , Stand: 25.05.2005, Abruf: 25.05.2005
- Kim Marriot / Peter J. Stuckey: Programming with Constraint: An Introduction, MIT Press 1998, ISBN 0-262-13341-5
- Vijay Saraswat / Pascal von Hentenryk (Hrsg.): Principles and Practice of Constraint Programming, MIT Press 1995, ISBN 0-262-19361-2
- Brian Mayoh, Enn Tyugu, Jaan Penjam (Hrsg.): Constraint Programming
NATO ASI Series Vol. 131, Springer 1994, ISBN 3-540-57859-5
- o.V.: The ECLiPSe Constraint Logic Programming System,
Internet: <http://www.icparc.ic.ac.uk/eclipse/> ,
Stand: 16.12.2004, Abruf: 08.05.2005
- Roman Bartak: Online Guide to Constraint Programming,
Internet: <http://kti.mff.cuni.cz/~bartak/constraints/> , Stand 14.03.2005, Abruf 20.05.2005

Zeit für Fragen...