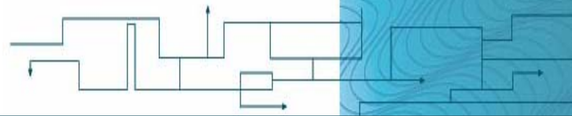
DAI-Labor
TU Berlin



Grundlagen der Künstlichen Intelligenz

Problemlösen II: Informierte Suche und Constrainterfüllung

27.10.2005

Brijnesh J Jain, Stefan Fricke
bjj@dai-labor.de stefan.fricke@dai-l-abor.de

A|O|T
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Lernziele: Wir diskutieren, wie ein Agent Entscheidungen durch systematisches Suchen trifft.

Gliederung

- ⇨ **Einleitung**
- ⇨ **Bewertungsfunktionen**
- ⇨ **Suchstrategien basierend auf Bewertungsfunktionen**
- ⇨ **Constrainterfüllung**
- ⇨ **Zusammenfassung und Ausblick**

Einleitung

- **Was haben wir letzte Vorlesung gemacht?**
  - Problemformulierung
    - Aktionen haben identische Kosten
  - **Uninformierte** Suchstrategien für **identische** Kosten
- **Probleme:**
  - Identische Kosten für viele Anwendungen nicht adäquat
    - z.B. Routen planen
  - Suche ohne Vorwissen häufig ineffizient

© B.J. Jain, S. Fricke

z.B. Routen planen: die Entfernung zwischen 2 Städten spielt eine Rolle und kann mit Kosten assoziiert werden.

Suche ohne Vorwissen wird auch als blinde Suche bezeichnet. Sie führt nur in seltenen Fällen schnell zum Ziel.

Worüber handelt diese Vorlesung?

1. **Suchstrategien basierend auf Bewertungsfunktionen $f = g + h$**
   - → g = Kantenbewertungsfunktion / Pfadkosten
   - → h = heuristische Funktion (Vorwissen)

2. **Constrainterfüllung**
   - → Probleme als eine Menge von Nebenbedingungen formulieren

Gliederung

- ⇨ Einleitung
- ⇨ **Bewertungsfunktionen**
- ⇨ Suchstrategien basierend auf Bewertungsfunktionen
- ⇨ Constrainterfüllung
- ⇨ Zusammenfassung und Ausblick

Bewertungsfunktionen

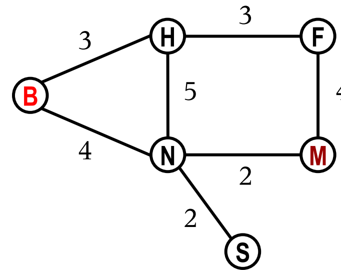
- **Bewertungsfunktion f misst, wie gut eine Teillösung ist**
- **Suchverfahren basierend auf Bewertungsfunktionen f**
  - **Gegeben:** Liste $(P_1, \ldots, P_k)$ von zu untersuchenden Teilpfaden
  - **Strategie:** Expandiere Teilpfad P* mit minimaler Bewertung

$$P^* = \arg\min_{1 \leq i \leq k} f(Pi)$$

- **Frage: Wie sieht f aus?**

Bewertungsfunktionen

- **Pfadkosten g**
  - Verschiedene Aktionen haben unterschiedliche Kosten
  - Pfadkosten sind die akkumulierten Kosten der entsprechende Sequenz von Aktionen



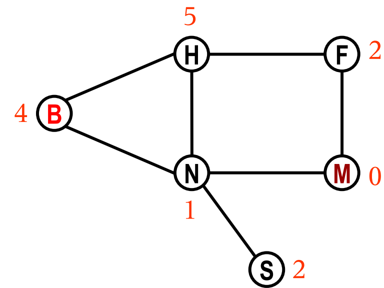
B	Berlin (Start)
F	Frankfurt
H	Hannover
M	München (Ziel)
N	Nürnberg
S	Stuttgart

- **Beispiel: Straßenentfernung**

$$g([B, N, S]) = g([B, N]) + g([N, S]) = 4 + 2 = 6$$

Bewertungsfunktionen

- Heuristische Funktion h
  - Schätzung der minimalen Kosten von einem Knoten zum Ziel
- Beispiel: Luftlinie
  - $h(B) = 4$
  - $h(N) = 1$
  - $h(S) = 2$



B	Berlin (Start)
F	Frankfurt
H	Hannover
M	München (Ziel)
N	Nürnberg
S	Stuttgart

Bewertungsfunktion

Heuristische Funktion

- Das Wort **Heuristik** (abgeleitet vom Griechischen ευρίσκω) bedeutet **ich finde.**
- Heuristik bezeichnet eine Strategie, die das Streben nach Erkenntnis und das Finden von Wegen zum Ziel planvoll gestaltet [wikipedia].
- Heuristiken sind Faustregeln, zur Steuerung der Suche.

[wikipedia]: http://de.wikipedia.org/wiki/Heuristik, zugegriffen am 30.10.2005.

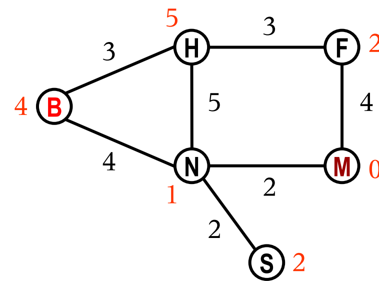
Für Optimierungsprobleme, deren optimale Lösung nur sehr aufwendig zu berechnen ist, erhofft man durch Anwendung von Heuristiken relativ gute Lösungen mit deutlich weniger Rechenaufwand zu erzielen. Z.B. statt den in der letzten VL behandelten systematischen, „blinden“ Verfahren zusätzliches Wissen nutzen. Dies kann auch trivialer Natur sein wie z.B. lexikographisch vorgehen und muss nicht unbedingt eine Verbesserung bringen. Normalerweise kann man bei Verwendung von Heuristiken keine Angaben darüber machen, wie gut die gefundene Lösung tatsächlich ist.

Heuristik ist die "Untersuchung der Mittel und Methoden des Aufgabenlösens" (G. Pólya).

Bewertungsfunktionen

- **Bewertungsfunktion f = g + h**
- **Spezialfälle**
  - $f = g \;(h = 0)$ werte nur Pfadkosten
  - $f = h \;(g = 0)$ werte nur Restkosten
- **Beispiel:**

$$f([B, N, S]) = g([B, N, S]) + h(S) = 6 + 2 = 8$$



B	Berlin (Start)
F	Frankfurt
H	Hannover
M	München (Ziel)
N	Nürnberg
S	Stuttgart

Bewertungsfunktion

- **Zulässigkeit:**
  - h unterschätzt die tatsächlichen Kosten h^*, d.h.
    $$0 \le h(X) \le h^*(X)$$
- **Dominanz:**
  - h_1 dominiert h_2 (ist „besser informiert“), wenn h_1, h_2 zulässig und
    $$h_1(X) \ge h_2(X)$$
    für alle Knoten X.
- **Folgerungen:**
  - Wenn h zulässig, dann ist $h(Z) = 0$, wobei Z ein Zielknoten ist
  - $h(X) = 0$ für alle Knoten X ist zulässig und maximal uninformiert

Unterschätzung der tatsächlichen Kosten entspricht einer optimistischen Herangehensweise. Die dominante Heuristik ist näher an der Lösung (d.h. schätzt die tatsächlichen Kosten exakter) als die dominierte.

$h(X) = 0$ für alle Knoten X entspricht der uninformierten Suche der letzten VL.

Beispiele: 8-Puzzle Problem

- ⇨ $h(X)$ = Zahl falsch liegender Ziffern
  - Zulässig
  - Beispiel: $h(X) = 6$
- ⇨ $h(X)$ = Manhattan Distanz
  - Zulässig
  - Beispiel: $h(X) = 14$

1	2	3	4	5	6	7	8	∑
4	0	3	3	1	0	2	1	14

7	2	4
5		6
8	3	1

Zustand X

1	2	3
4	5	6
7	8	

Zielzustand Z

Manhattan Distanz: Summe der Differenzbeträge der aktuellen Positionen von den Zielpositionen. Der Name rührt von der rechtwinkligen Anordnung der Straßen in Manhattan her, wo man sich entsprechend durch Geradeaus-Rechts-Links von A nach B bewegt.

Bewertungsfunktion

- **Problem: Wie findet man eine heuristische Funktion h?**
  - Intuition, Wissen, Genialität, …
  - Oftmals mehr eine Kunst als eine Wissenschaft

- **Anforderungen an h:**
  - Möglichst genau & zulässig
  - einfach zu berechnen

Gliederung

- ⇨ Einleitung
- ⇨ Bewertungsfunktionen
- ⇨ **Suchstrategien basierend auf Bewertungsfunktionen**
- ⇨ Constrainterfüllung
- ⇨ Zusammenfassung und Ausblick

Im Folgenden wird der A*-Algorithmus vorgestellt, der eine heuristische Suche durchführt und eine Weiterentwicklung von Branch&Bound (bekannt aus Operations Research) darstellt.

Suchstrategien **A* Algorithmus**

- **Idee:**
  - Expandiere Teilpfad mit minimaler Bewertung $f = g + h$
  - Verwende Prinzip der dynamischen Programmierung
- **Implementierung:**
  - Sortierte Liste von Teilpfaden
  - Reihenfolge: Aufsteigend bzgl. Bewertung f

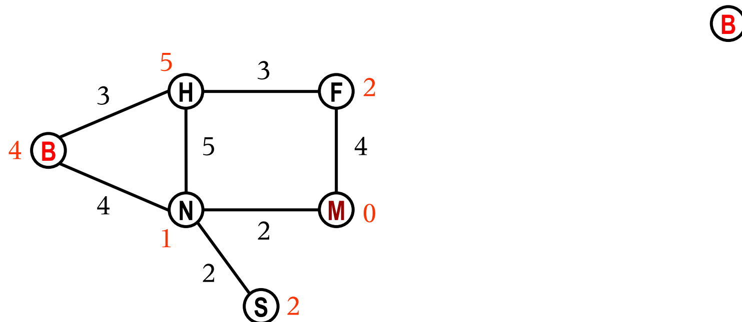
Der A*-Algorithmus berechnet den kürzesten Pfade zwischen einem Start- und einem Endknoten in einem kantengewichteten Graphen (positive Gewichte).

Prinzip der dynamischen Programmierung (siehe letzte VL): Von A nach B nur jeweils den bekannten besten Pfad nehmen, suboptimale Lösungen für [A->B] aus dem Gedächtnis löschen. Durch dieses Beschneiden des Suchbaums kann der Suchraum bedeutend verkleinert werden.

Suchstrategien: Simulation

A* Algorithmus

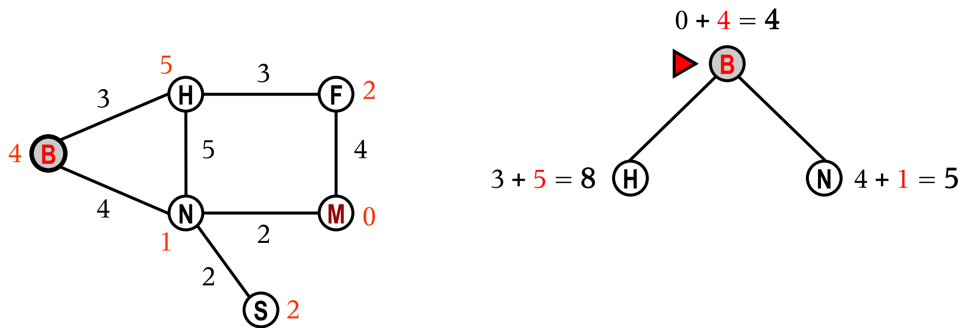
⇨ **Initialzustand**



Die Knotenbeschriftung (rote Farbe) beschreibt die direkte Entfernung (Luftlinie) zum Zielort (von Berlin nach München: 4; von Nürnberg: 1). Durch Anwendung dieser heuristischen Funktion h ergibt sich eine zielgerichtete Suche in Richtung auf den Endknoten.

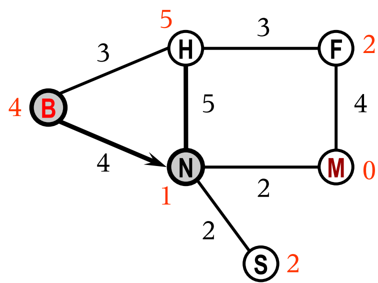
Die Kantenbeschriftungen (g) kennzeichnen die tatsächlichen Kosten zwischen 2 Orten (B->N: 4; N->M: 2).

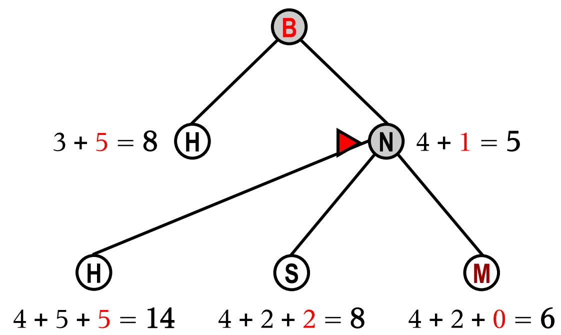
⇨ Simulation: Expansion 1: Neuberechnung der Knotenwerte



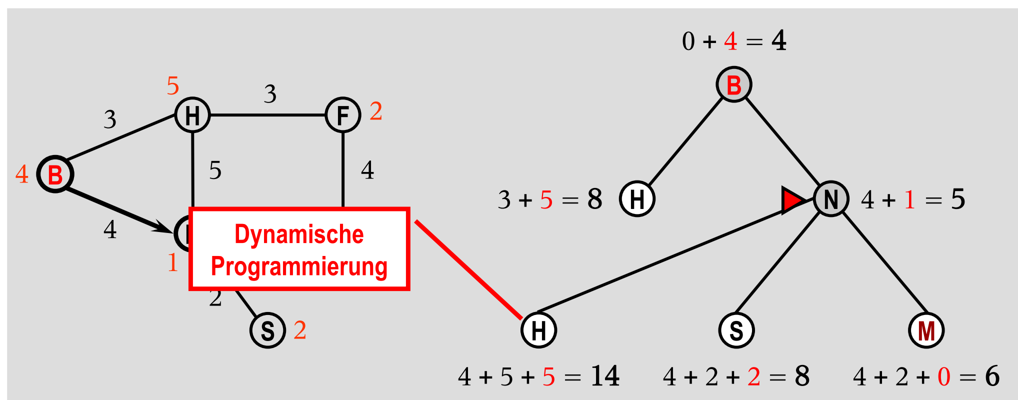
Suchstrategien: Simulation

- ⇨ Expansion 2 nach Wahl des billigsten Knotens
- ⇨ Neuberechnung der expandierten Knoten

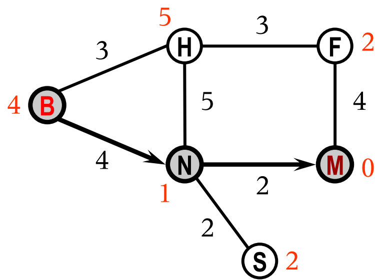


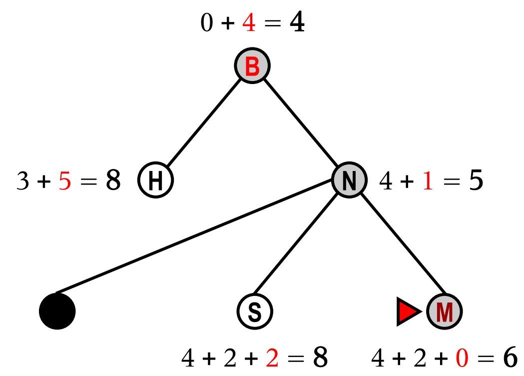


⇨ **Dynamische Programmierung: Entfernen suboptimaler Teilbäume**

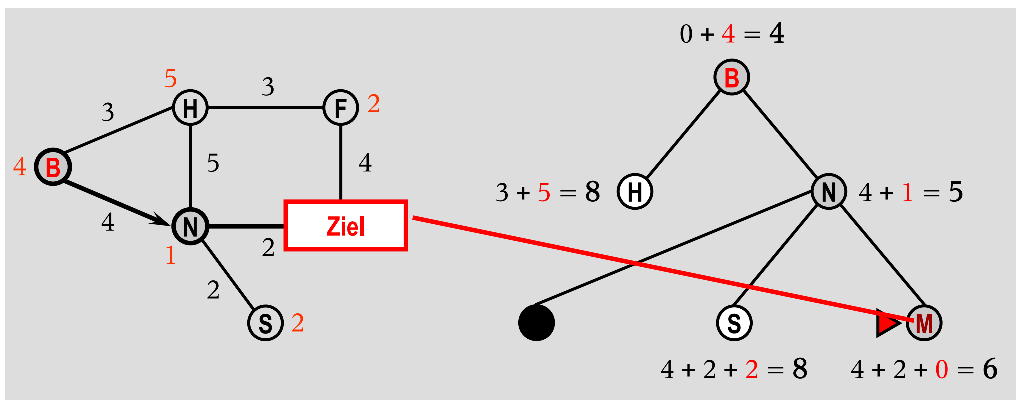


⇨ **Nächster zu expandierender Knoten ist M.**





⇨ **Ziel erreicht, Kosten = 6**



Suchstrategien

Algorithmus A* [Winston 93]

1. Form a one-element queue consisting of a zero-length path that contains only the root node.
2. Until the first path in the queue terminates at the goal node or the queue is empty
   a) Remove the first path from the queue; create new paths by extending the first path to all neighbors of the terminal node.
   b) Reject all paths with loops.
   c) If two or more paths reach a common node N, delete all those paths except the one that reaches N with the minimum cost. (*Dynamic Programming*)
   d) Add the remaining new paths, if any, to the queue.
   e) Sort the entire queue by ascending order of the evaluation function $f = g + h$
3. If the goal node is found, announce success; otherwise announce failure.

Der A*-Algorithmus beschreitet immer den viel versprechendsten Weg, ohne dabei die Vollständigkeitsbedingung zu verletzen (es werden notfalls alle möglichen Wege durch den Problembaum durchwandert).

Suchstrategien: Eigenschaften von A*

⇨ **A* ist optimal effizient für jede beliebige Heuristikfunktion h.**

→ Das heißt, kein anderer Algorithmus kann garantieren, dass er mit h weniger Knoten aufspannt als A*.

Unter der Bedingung

$|h(n) - h^*(n)| \leq O(\log h^*(n))$, wobei h das Zulässigkeitskriterium erfüllt

ist die Zeitkomplexität subexponentiell.

Vollständigkeit	Ja
Optimalität	Ja
Zeitkomplexität	$O(b^m)$
Speicherkomplexität	$O(b^m)$

A|O|T Grundlagen der Künstlichen Intelligenz © B.J. Jain, S. Fricke 23

Der A*-Algorithmus ist vollständig, d.h. wenn ein Pfad zum Zielknoten existiert, so wird dieser auch gefunden. Außerdem ist er optimal, d.h. es gibt keinen kürzeren Pfad zum Zielknoten als den gefundenen.

Zudem ist A* optimal effizient, d.h. jeder andere optimale und vollständige Algorithmus, der dieselbe Heuristik verwendet, muss mindestens ebenso viele Knoten betrachten wie A*, um eine Lösung zu finden.

Leider ist auch A* nicht die beste Lösung für viele Probleme, wegen der Tatsache, dass die Zahl der Knoten i.A. exponentiell zur Baumtiefe steigt.

b = Verzweigungsgrad

m = maximale Tiefe des Suchbaums

A* ist subexponentiell (Zeitkomplexität), wenn der Fehler der heuristischen Funktion (Abweichung von h(n) von der optimalen Heuristik h*(n)) nicht schneller wächst als der Logarithmus der aktuellen Pfadkosten (die wahren Kosten h*).

Die Speicherkomplexität ist hoch, weil alle generierten Knoten im Speicher gehalten werden. In der Praxis ist die Komplexität deutlich niedriger, zum einen wegen des Abschneidens suboptimaler Bäume (dynamische Programmierung).

- **Best-First Search:** Variation von A* mit
  - $f = h$
  - ohne dynamische Programmierung
- **Branch and Bound:** Variation von A* mit
  - $f = g$
  - ohne dynamische Programmierung
- **Bemerkung:**
  - Algorithmen im Anhang

© B.J. Jain, S. Fricke

Ausblick

- **Local Search behandelt Suchalgorithmen, für Optimierungsprobleme, bei denen der Lösungspfad unwichtig ist**
  - Z.B. 8-Damen-Problem, Design von Schaltkreisen, Netzwerkoptimierung

- **Wird in der Übung behandelt: Simulated Annealing**

Gliederung

- ⇨ Einleitung
- ⇨ Bewertungsfunktionen
- ⇨ Suchstrategien basierend auf Bewertungsfunktionen
- ⇨ **Constrainterfüllung**
- ⇨ Zusammenfassung und Ausblick

Constraint Satisfaction Problems (CSP)

- **Probleme lösen mittels Suche im Problemraum:**
  - Formuliere das Problem als Graph, finde Zielzustand.
- **Probleme lösen mittels CSP:**
  - Finde *„zufrieden stellende"* Konfiguration.
  - Idee: Formuliere das Problem durch eine Menge von Nebenbedingungen (Constraints) und werte diese aus.

Auch CSP sind Suchprobleme!

CSP — Beispiel

- **Gegeben:**
  - Variablen x, y mit Wertebereich $\mathbb{R}$
  - Constraints (Nebenbedingungen) C_1, C_2 mit
    - C_1: $2x + y = 10$
    - C_2: $x + y = 7$
- **Ziel:**
  - Finde Belegung $\{x = a, y = b\}$ mit $a, b \in \mathbb{R}$, sodass beide Constraints C_1, C_2 erfüllt sind.

Bei CSPs kommt es darauf an, eine Lösung zu finden (alternativ: alle Lösungen zu generieren). Häufig gibt es viele Lösungen (zumeist unendlich viele bei linearen Gleichungen) oder auch gar keine, dann spricht man von einem overconstrained problem.

CSP — Formalisierung

- **Constraintnetz**
  - Variablen x_i mit Werten aus Wertebereichen D_i $(1 \leq i \leq n)$
  - Constraints $C_1, \ldots C_m$
- **Constraint (Neben- / Randbedingung)**
  - Ein k-stelliges Constraint C beschreibt eine Relation

$$R_C \subseteq D_{i1} \times \ldots \times D_{ik}$$

- **Constraint Satisfaction Problem (CSP)**
  - Finde Belegung

$$\{x_1 = v_1, \ldots, x_n = v_n\},$$

sodass $v_i \in D_i$ und alle Constraints $C_1, \ldots C_m$ erfüllt sind.

A|O|T Grundlagen der Künstlichen Intelligenz © B.J. Jain, S. Fricke 29

D steht für Domain. Das können reelle oder natürliche Zahlen sein, aber auch so genannte finite domains, das sind endlich aufzählbare Wertebereiche (z.B. Farben rot, grün, blau).

- **Belegung**
  - Belegung einer Teilmenge von Variablen x_i mit Werten aus D_i
- **Konsistente Belegung**
  - Belegung, die keine Constraints verletzt
- **Vollständige Belegung**
  - Belegung aller Variablen x_i mit Werten aus D_i
- **Lösung des CSP**
  - Vollständige und konsistente Belegung

CSP

- **Problem:**
  - Wie können wir CSPs lösen?
- **Idee:**
  - Umformulierung von CSP als Suchproblem
  - Jeder Algorithmus, der ein Suchproblem löst, löst auch ein CSP
- **Frage:**
  - Wie können wir CSP als Suchproblem formulieren?

CSP als inkrementelles Suchproblem

- **Problemraum**
  - Menge aller Belegungen
- **Anfangszustand**
  - Leere Belegung { }
- **Zielzustände**
  - Vollständig konsistente Belegungen
- **Aktionen (inkrementelle Formulierung)**
  - Belege eine freie Variable, sodass Konsistenz erhalten bleibt

CSP Beispiel: Färbeproblem

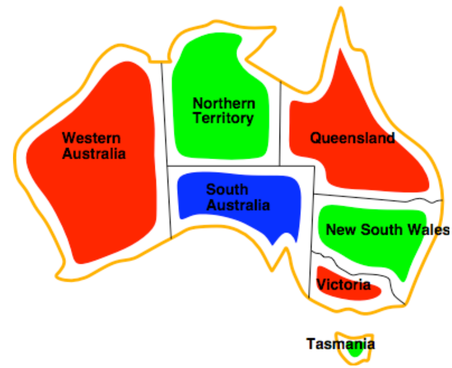
Färbe die Regionen einer Karte mit 3 Farben so ein, dass keine zwei benachbarten Regionen dieselbe Farbe haben.

- **Variablen:**
  - WA, NT, Q, NSW, V, SA, T
- **Wertebereich:**
  - {rot, grün, blau}
- **Constraints:**
  - Benachbarte Regionen haben unterschiedliche Farbe

CSP: Lösung des Färbeproblems

⇨ **Vollständige konsistente Belegung**

$$\{ WA = rot, NT = grün, Q = rot, NSW = grün, V = rot, SA = blau, T = grün \}$$



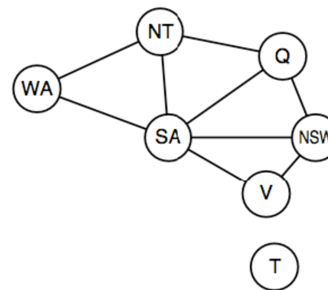
CSP Repräsentation des Constraintgraphen

⇨ **Visualisierung des CSP**

- Knoten repräsentieren Variablen
- Kanten repräsentieren Constraints

⇨ **Beispiel: Färbeproblem**

- Binäres CSP (2-stellige Constraints)



⇨ **Bemerkung:**

- Bei k-stelligen Constraints ist Constraintgraph ein Hypergraph.

© B.J. Jain, S. Fricke

CSP

- **Problem:**
  - Finde Lösung, d.h. vollständig konsistente Belegung
- **Ansatz:**
  - Tiefensuche (Backtracking)
- **Warum keine Breitensuche?**
  - Jede Lösung hat Pfadlänge n (n = Anzahl der Variablen)
  - Breitensuche exploriert den gesamten Suchraum

CSP Backtracking-Algorithmus (Tiefensuche)

1. Form a 1-element stack consisting of an empty assignment of the variables.
2. Until the first assignment in the stack is complete and consistent or the stack is empty
   a) Remove the first assignment from the stack; create new assignments by assigning a value to a single unassigned variable in all possible ways.
   b) Reject all inconsistent assignments.
   c) Push the new assignments, if any, to the top of the stack.
3. If a complete and consistent assignment is found, announce success; otherwise announce failure.

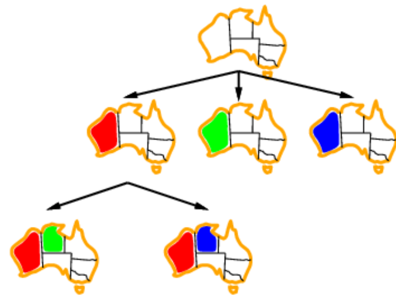
Nacheinander werden alle Variablen mit allen zulässigen Werten belegt.

CSP Tiefensuche

Schritt 0

© B.J. Jain, S. Fricke

CSP Tiefensuche

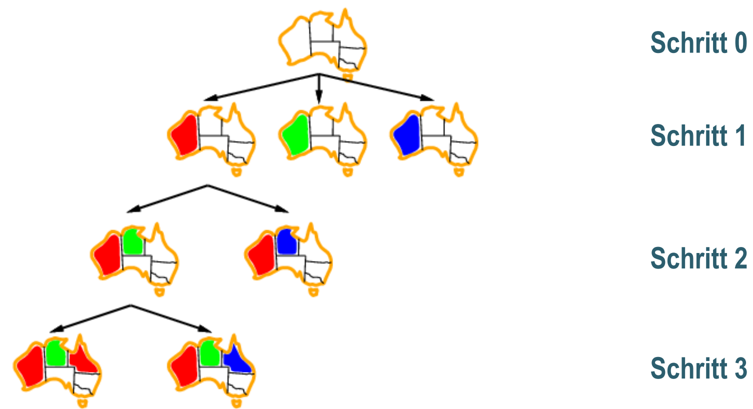


Schritt 0

Schritt 1

Schritt 2

CSP Tiefensuche



CSP Leistungsmessung der Tiefensuche

- d = Wertebereichsgröße
- n = Anzahl der Variablen

☺	**Vollständigkeit**	**Ja**	**Für endliche diskrete Wertebereiche**
☺	**Optimalität**	**Ja**	**Trivial, denn jede Lösung ist optimal**
😐	**Zeitkomplexität**	$O(d^n)$	
☺	**Speicherkomplexität**	$O(n^2 d^2)$	**Für binäre Constraints**

CSP: Diskussion der Tiefensuche

- **Tiefensuche ist für reale Probleme zu ineffizient**
- **Frage:**
  - Können wir Tiefensuche verbessern?
- **Antwort:**
  - Ja, sogar ohne domänenspezifisches Wissen (heuristische Funktion)
  - General-purpose methods

Tiefensuche wird häufig als Referenzverfahren verwendet.

CSP: General-Purpose Methods basierend auf Backtracking

Wichtige Entwurfsentscheidungen für bessere Effizienz:

1. Welche Variable soll als nächste mit einem Wert belegt werden?
   → Verfahren: Minimum Remaining Values Heuristik

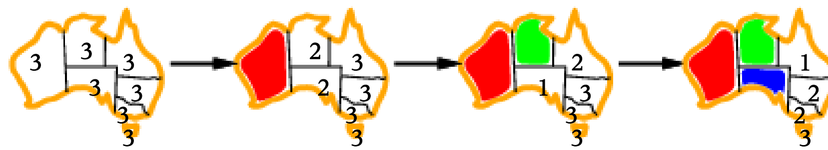
2. In welcher Reihenfolge sollen die Werte zugeordnet werden?
   → Verfahren: Least Constraining Value Heuristik

3. Können Misserfolge rechtzeitig erkannt werden?
   → Verfahren: Forward Checking und Constraint Propagation

© B.J. Jain, S. Fricke

CSP: Minimum Remaining Values (MRV) Heuristik

- ⇨ Auswahl der nächsten zu belegenden Variable
- ⇨ Früherkennung von Belegungen, die zu Inkonsistenz führen
- ⇨ Idee: Belege Variable mit minimaler Anzahl von „*plausiblen*“ Werten



Bemerkung:

1. Ein Wert ist für eine Variable bei gegebener konsistenter Belegung plausibel, wenn die Belegung durch die Zuordnung der Variable mit dem Wert konsistent bleibt.

Es wird immer diejenige Variable belegt, deren Wertebereich am weitesten eingeschränkt ist (die wenigsten Constraint erfüllenden Werte besitzt). Die Größe der Wertebereiche steht als Ziffer in den Karten.

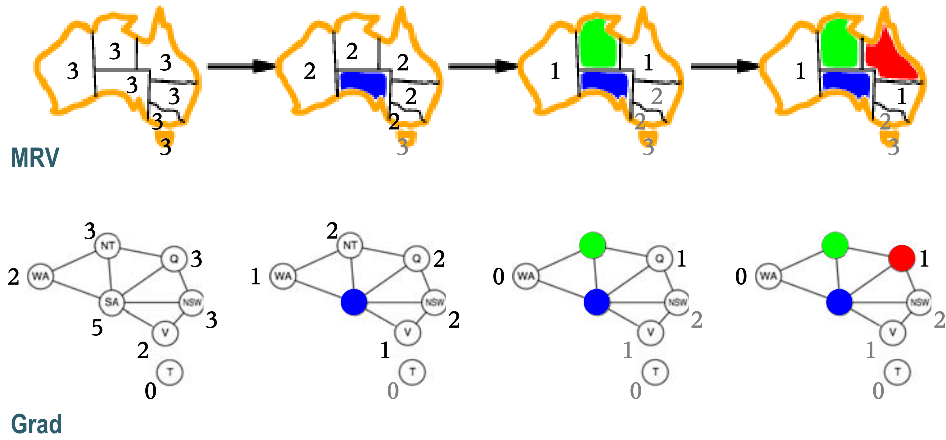
CSP: Tie-Breaker (Entscheidungshilfe) für MRV-Variablen

- **Problem:**
  - MRV-Variable ist i.A. nicht eindeutig bestimmt.
  - Welche MRV-Variable soll als nächstes belegt werden?

- **Lösung: Grad-Heuristik als Tie-Breaker**
  - Wähle diejenige MRV Variable mit den meisten Constraints zu freien (d.h. noch nicht belegten) Variablen.

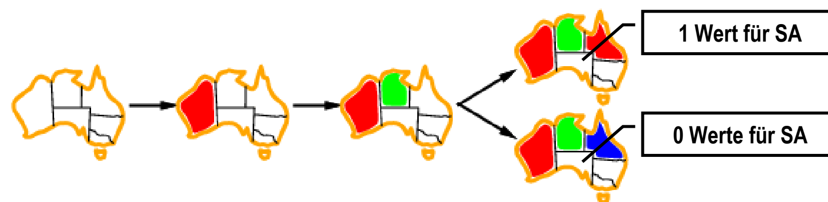
CSP: Tie-Breaker (Entscheidungshilfe) für MRV-Variablen

Vernetzungsgrad der Constraintvariablen als Entscheidungshilfe



CSP: Least Constraining Values (LCV) Heuristik

- Problem: Auswahl des nächsten Werts, um freie Variable zu belegen
- Idee: Wähle denjenigen Wert, der die wenigsten Werte für benachbarte freie Variablen eliminiert.



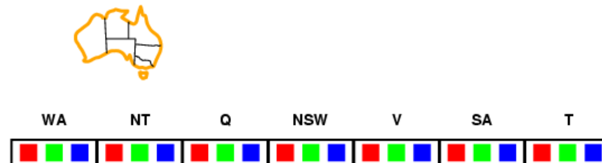
 © B.J. Jain, S. Fricke

Mit Blick auf die noch nicht instantiierten Variablen wird für die gewählte Variable ein solcher Wert genommen, der den geringsten Einfluss auf die Wertebereiche der verbleibenden Variablen hat.

CSP: Forward Checking

- ⇨ Protokolliere alle „plausiblen“ Werte für freie Variablen.
- ⇨ Terminiere, wenn es eine Variable ohne „plausible“ Werte gibt.

Initialisierung: Alle Variablen besitzen ihre initialen Wertebereiche.

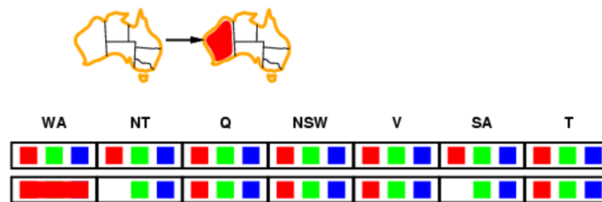


Forward checking propagiert die Einschränkungen, die durch Instantiierung einer Variablen geschehen, an alle benachbarten Variablen. Auf diese Weise können die Wertebereiche vieler Variablen durch Belegung einer einzigen Constraintvariable stark eingeschränkt werden. Das führt dann zu einem wesentlich kleineren Suchbaum.

Szenario: zu Beginn sind alle Variablen unbeschränkt (Wertebereich jeweils (rot, grün, blau)).

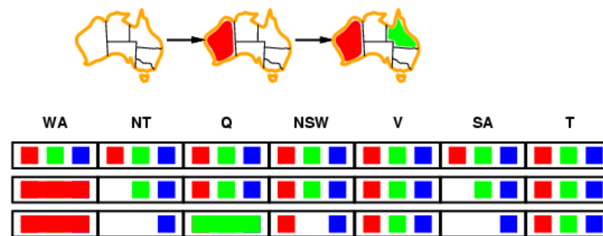
CSP: Forward Checking

⇨ **Durch Instantiierung der Variable WA mit rot wird dieser Wert aus den benachbarten Variablen NT und SA entfernt.**



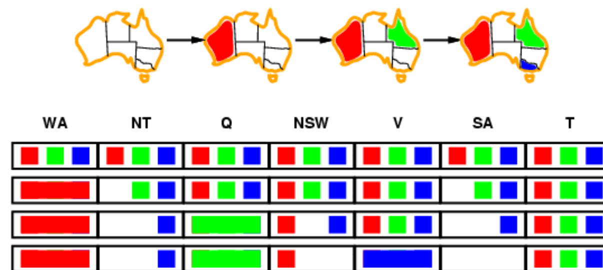
CSP: Forward Checking

⇨ **Belegung von Q mit grün führt zu Wertebereichseinschränkungen für NT, NSW und SA**



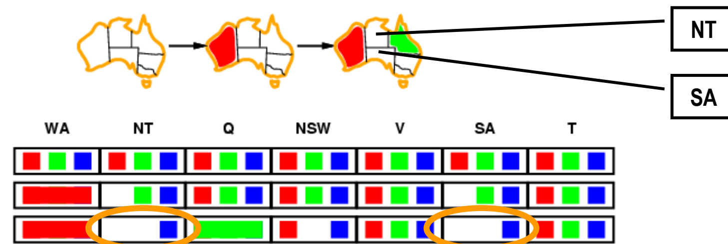
CSP: Forward Checking

⇨ **Belegung von V mit Blau führt zum Konflikt: SA hat einen leeren Wertebereich**



CSP: Problem des Forward Checking

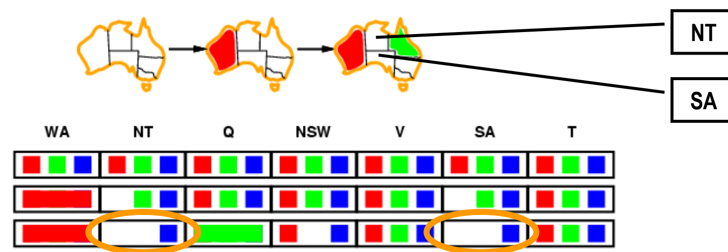
- ⇨ FC propagiert Informationen von belegten zu unbelegten Variablen
- ⇨ Entdeckt frühzeitig einige aber nicht alle Inkonsistenzen



- ⇨ NT und SA können nicht beide mit blau belegt werden

CSP: Constraint Propagation

- Constraint Propagation propagiert alle Implikationen durch das gesamte Constraintnetz.
  - Der Propagierungsprozess endet, sobald das Netz im Ruhezustand ist



- Auch das Constraint zwischen NT und SA wird ausgewertet.

CSP: Constraint Propagation und Arc Consistency

- Für jede Kante (arc) im Constraintgraph zwischen Variablen a und b ist durch Constraint Propagation sichergestellt, dass für jeden Wert von a mindestens ein konsistenter Wert für b existiert (und vice versa).
- Arc Consistency am Beispiel a + b = c (3-Konsistenz):

Vorher:	Propagierung:	Nachher:
a: {1,3,4,5,6}	a: {1,3,~~4~~,6}	a: {1,3,6}
b: {1,3,7,9,19}	b: {~~1~~,3,7,~~9~~,19}	b: {3,7,19}
c: {8,9,22,50}	c: {8,9,22,~~50~~}	c: {8,9,22}

CSP — Anwendungen

- ⇨ **Assignment und Scheduling Probleme**
- ⇨ **Konfiguration und Layout**
- ⇨ **Bemerkung:**
  - → Die meisten Anwendungen haben kontinuierliche Variablen

Z.B. das Schienennetz der Deutschen Bahn und die Belegung mit Zügen, Belegungsplanung im Krankenhaus, Einsatz von Lehrern in Schulen.

z.B. Konfiguration von Fahrstühlen, Layout von Chips.

Zusammenfassung Informed Search

- **Informed Search ist Suche unter Verwendung von Heuristiken.**
  - Pfadkosten und geschätzte Kosten zum Ziel.
- **A* ist optimal effizient.**
- **Das Finden guter Heuristiken ist oft schwierig.**

Zusammenfassung Constraint Satisfaction Problems

- ⇨ Beschreibung des Problems als eine Menge von Nebenbedingungen
- ⇨ CSP als Suche mittels Backtracking durchführbar
- ⇨ Variablenauswahl und Werteauswahl zur Effizienzsteigerung nutzen
- ⇨ Forward Checking propagiert Wertebereichseinschränkungen an benachbarte Variablen
- ⇨ Constraint Propagation propagiert Wertebereichseinschränkungen durch das gesamte Constraintnetz

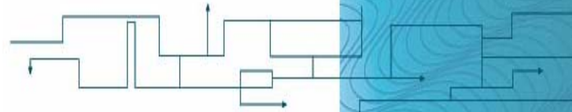
Ausblick

- **Diese Vorlesung:**
  - Informierte Suchverfahren mit gewichteten Kanten und unter Einsatz von Heuristiken
  - Constraint Satisfaction Problems als Suche mittels Backtracking
  - CSPs zur Suchraumbegrenzung durch Propagierung von Wertebereichseinschränkungen
- **Nächste Vorlesung:**
  - Planen

 © B.J. Jain, S. Fricke

Planen: Aktionsfolgen finden, um Zielzustände zu erreichen. Der Weg vom Startzustand zum Ziel spielt also eine wichtige Rolle, genauso wie in den Suchverfahren. Aktionen/Handlungen beziehen sich dabei auf (komplexe) Weltzustände, die typischerweise nicht einfach als Punkte/Knoten im Grafen darstellbar sind, sondern z.B. prädikatenlogisch umschrieben werden.

DAI-Labor
TU Berlin



Informierte Suchverfahren

nächster Termin: 03.10.2005

Planen

Brijnesh J Jain, Stefan Fricke
bjj@dai-labor.de stefan.fricke@dai-l-abor.de

A|O|T
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Referenzen

1. S.J. Russell, P. Norvig: *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc., 2003.
2. P.H. Winston: *Artificial Intelligence*. Addison-Wesley, 1993.

Anhang

- **Algorithmen**
  - A*
  - Best First Search
  - Branch and Bound

© B.J. Jain, S. Fricke

Algorithmus A* [Winston 93]

1. Form a one-element queue consisting of a zero-length path that contains only the root node.
2. Until the first path in the queue terminates at the goal node or the queue is empty
   a) Remove the first path from the queue; create new paths by extending the first path to all neighbors of the terminal node.
   b) Reject all paths with loops.
   c) If two or more paths reach a common node N, delete all those paths except the one that reaches N with the minimum cost. (*Dynamic Programming*)
   d) Add the remaining new paths, if any, to the queue.
   e) Sort the entire queue by ascending order of the evaluation function $f = g + h$
3. If the goal node is found, announce success; otherwise announce failure.

Algorithmus Best First Search [Winston 93]

1. Form a one-element queue consisting of a zero-length path that contains only the root node.
2. Until the first path in the queue terminates at the goal node or the queue is empty
   a) Remove the first path from the queue; create new paths by extending the first path to all neighbors of the terminal node.
   b) Reject all paths with loops.
   c) Add the remaining new paths, if any, to the queue.
   d) Sort the entire queue by ascending order of the evaluation function $f = h$
3. If the goal node is found, announce success; otherwise announce failure.

© B.J. Jain, S. Fricke

Algorithmus Branch and Bound [Winston 93]

1. Form a one-element queue consisting of a zero-length path that contains only the root node.
2. Until the first path in the queue terminates at the goal node or the queue is empty
   a) Remove the first path from the queue; create new paths by extending the first path to all neighbors of the terminal node.
   b) Reject all paths with loops.
   c) Add the remaining new paths, if any, to the queue.
   d) Sort the entire queue by ascending order of the evaluation function $f = g$