DAI-Labor
TU Berlin

Grundlagen der Künstlichen Intelligenz

Problemlösen und Suche

27.10.2005

Brijnesh J Jain
bjj@dai-labor.de

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

Gliederung

- ⇨ Problemlösungsagenten
- ⇨ Problemformulierung
- ⇨ Suche nach Lösungen
- ⇨ Zusammenfassung und Ausblick

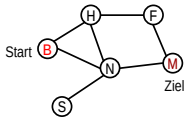
Problemlösungsagenten

- ⇨ Zentrale Fragen:
  - → Was ist ein Problem?
  - → Was ist eine Lösung?
  - → Wie findet man eine Lösung?
- ⇨ Antworten über agentenbasierten Ansatz:
  - → Problemlösungsagenten sind spezielle zielorientierte Agenten
  - → hier: PL-Agenten mit *formulate-search-execute* Design

Problemlösungsagenten

Beispiel für einen Problemlösungsagenten

- ⇨ Ziel formulieren (formulate I):
  - → Fahre von Berlin nach München
- ⇨ Problem formulieren (formulate II):
  - → Zustände: Städte B, F, H, M, N, S
  - → Aktionen: Fahre zwischen Städten
- ⇨ Suche (search):
  - → Finde Sequenz von Städten zum Ziel
  - → z.B.: B – N - M
- ⇨ Ausführen (execute):
  - → Führe gefundene Lösung aus



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

Problemlösungsagenten

- ⇨ Wichtige Entwurfsentscheidungen
  - → Problemformulierung
    - → Wie formuliert man Probleme?
  - → Suchalgorithmen
    - → Wie findet man eine Lösung?

Gliederung

- ⇨ Problemlösungsagenten
- ⇨ Problemformulierung
- ⇨ Suche nach Lösungen
- ⇨ Zusammenfassung und Ausblick

Problemformulierung

- Problemraum: Menge S von Zuständen (states)
- Problem: Tripel $P = (S_a, S_z, A)$, bestehend aus
  - einer Menge $S_a \subseteq S$ von Anfangszuständen
  - einer Menge $S_z \subseteq S$ von Zielzuständen
  - Einer Menge A von Aktionen
- Aktionen: $a \in A$, $a: S' \rightarrow S$, mit $S' \subseteq S$
  - Aktionen sind i.a. partiell definiert (Anwendungsbedingungen)

Problemformulierung

- Lösung: Sequenz $(s_0, \ldots, s_k)$ von Zuständen mit
  - $s_0 \in S_a$ ist ein Anfangszustand
  - $s_k \in S_z$ ist ein Zielzustand
  - es gibt Aktion a_i mit $a_i(s_i) = s_{i+1}$ für alle i = 0,..,k-1
- Optimale Lösung: Lösung minimaler Länge
- Problemlösen: Finden einer (optimalen) Lösung
  - Überführung eines Anfangszustands in einen Zielzustand durch Anwendung einer (minimalen) Folge von Aktionen

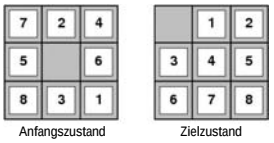
Problemformulierung — Beispiele

- Toy Problems
  - 8-Puzzle Problem
  - 8-Dame Problem
- Real-world Problems
  - Routen finden
  - Traveling Salesman Problem
  - VLSI Layout
  - Navigation von Robotern

Problemformulierung — Beispiele

8-Puzzle Problem

- Familie: sliding block puzzle
- NP-vollständig
- Standardtest in der KI



Anfangszustand Zielzustand

Problemformulierung

Problemraum	Jede Konfiguration der 8 Kacheln plus Leerstelle (*Blank*)
Anfangszustand	Jeder Zustand kann als Anfangszustand ausgewählt werden
Zielzustand	Konfiguration wie in der Grafik angezeigt
Aktionen	Jede zulässige Bewegung des Blanks (*links, rechts, oben, unten*)

Gliederung

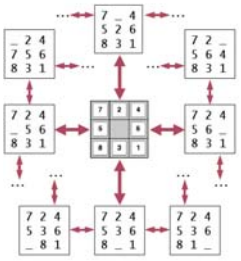
- Problemlösungsagenten
- Problemformulierung
- Suche nach Lösungen
  - Suchbäume
  - Leistungsmessung
  - Suchverfahren
- Zusammenfassung und Ausblick

Suche nach Lösungen — Suchbäume

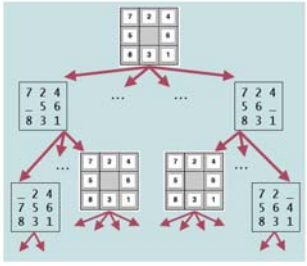
- Idee: Exploration des Zustandsraums durch Erzeugen von Folgezuständen bereits untersuchter Zustände (expanding states).
- Bei der Suche/Exploration wird ein Suchbaum aufgebaut
  - Wurzel = Anfangszustand
  - Knoten = Zustände
  - Kanten = Aktionen
- Eine Strategie bestimmt den zu expandierenden Knoten

Suche nach Lösungen

Problemraum als Graph



Suchraum als Baum



Suche: Traversiere Graph als Baum gemäß einer Strategie

Suche nach Lösungen

Informelle Beschreibung eines Suchbaum-Algorithmus:

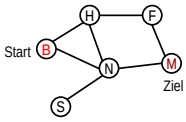
1. Form a one-node tree consisting of the initial state as root.
2. Color root as white.
3. Until there is no white leaf
   a) Select next white leaf as current leaf according to a strategy
   b) If the current leaf represents the goal state, return success. **Exit**
   c) Color current leaf as grey
   d) Expand current leaf and color its child-nodes as white leafs.
4. Return failure.

Suche nach Lösungen

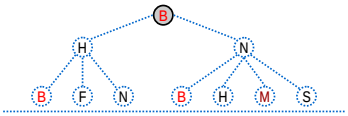
Beispiel

⇨Problem: Shortest Path

⇨Strategie: Lexikographisch



(a) Startzustand (nach Schritt 3c)

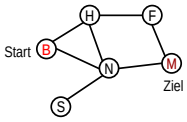


Suche nach Lösungen

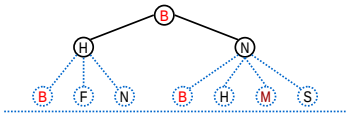
Beispiel

⇨Problem: Shortest Path

⇨Strategie: Lexikographisch



(b) nach Expansion von **B** (nach Schritt 3d)

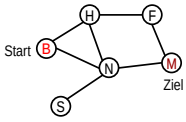


Suche nach Lösungen

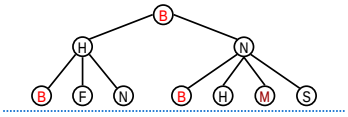
Beispiel

⇨Problem: Shortest Path

⇨Strategie: Lexikographisch



(c) nach Expansion von **H** und **N**



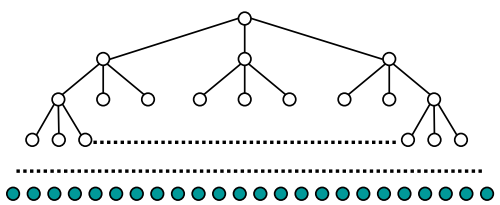
Suche nach Lösungen

⇨ Naive Suche

→ Alle Sequenzen von Aktionen der Reihe nach durchprobieren

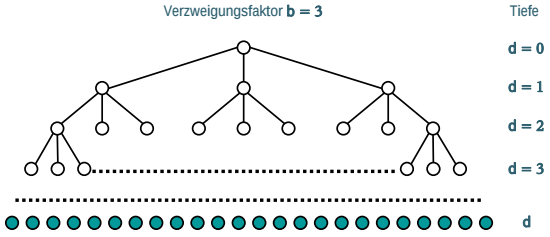
Verzweigungsfaktor **b = 3**

Tiefe



Suche nach Lösungen — Suchbäume

- # der Knoten der Tiefe **d**: b^d
- # der Knoten bis Tiefe **d**: $b^0 + b^1 + \ldots + b^d = (b^{d+1} - 1) / (b - 1)$



Suche nach Lösungen — Suchbäume

- Zeitkomplexität: $O(b^d)$
- Speicherkomplexität: $O(b^d)$ (alle Knoten der Tiefe d im Speicher)
- Gibt es bessere Lösungen?
  1. Leistungsmessung (was heißt besser?)
  2. Strategien (welche Suchverfahren gibt es?)

Gliederung

- Problemlösungsagenten
- Problemformulierung
- Suche nach Lösungen
  - Suchbäume
  - Leistungsmessung
  - Suchverfahren
- Zusammenfassung und Ausblick

Suche nach Lösungen — Leistungsmessung

- Vollständigkeit:
  - Findet die Strategie eine Lösung, wenn vorhanden?
- Optimalität:
  - Liefert die Strategie eine optimale Lösung?
- Zeitkomplexität:
  - Wie viel Zeit benötigt man für die Suche?
- Speicherkomplexität:
  - Wie viel Speicher benötigt man für die Suche?

Suche nach Lösungen — Leistungsmessung

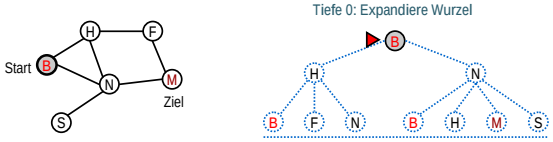
- Maß für Zeitaufwand:
  - Gesamtzahl der erzeugten Knoten
- Maß für Speicheraufwand:
  - Maximale Anzahl der Knoten im Speicher
- Faktoren zur Bestimmung des Zeit- und Speicheraufwands
  - b = Verzweigungsgrad (branching factor)
  - d = Tiefe (depth)
  - m = maximale Tiefe des Suchbaums

Gliederung

- Problemlösungsagenten
- Problemformulierung
- Suche nach Lösungen
  - Suchbäume
  - Leistungsmessung
  - Suchverfahren
    - Breitensuche
    - Tiefensuche
    - Limitierte Tiefensuche
    - Iterative Tiefensuche
- Zusammenfassung und Ausblick

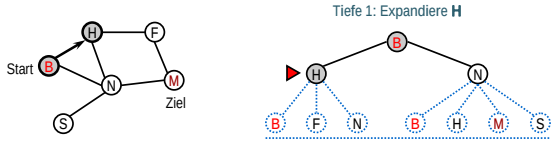
Suchstrategien — Breitensuche

- Idee: Expandiere Blatt mit minimaler Tiefe
- Implementierung: FIFO-Queue bestehend aus Teilpfaden
  - d.h. Nachfolger an das Ende der Queue



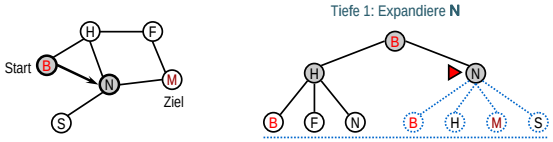
Suchstrategien — Breitensuche

- Idee: Expandiere Blatt mit minimaler Tiefe
- Implementierung: FIFO-Queue bestehend aus Teilpfaden
  - d.h. Nachfolger an das Ende der Queue



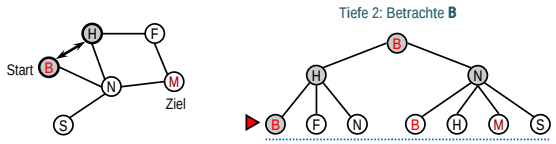
Suchstrategien — Breitensuche

- Idee: Expandiere Blatt mit minimaler Tiefe
- Implementierung: FIFO-Queue bestehend aus Teilpfaden
  - d.h. Nachfolger an das Ende der Queue



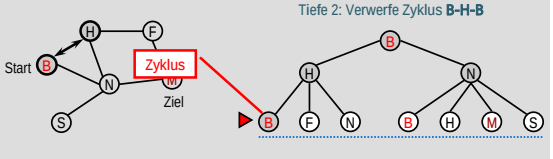
Suchstrategien — Breitensuche

- Idee: Expandiere Blatt mit minimaler Tiefe
- Implementierung: FIFO-Queue bestehend aus Teilpfaden
  - d.h. Nachfolger an das Ende der Queue



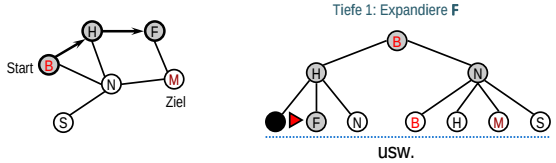
Suchstrategien — Breitensuche

- Idee: Expandiere Blatt mit minimaler Tiefe
- Implementierung: FIFO-Queue bestehend aus Teilpfaden
  - d.h. Nachfolger an das Ende der Queue



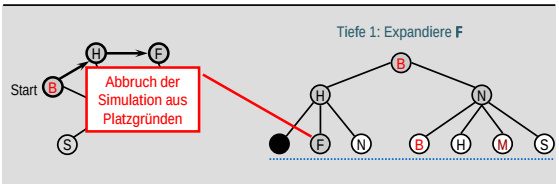
Suchstrategien — Breitensuche

- Idee: Expandiere Blatt mit minimaler Tiefe
- Implementierung: FIFO-Queue bestehend aus Teilpfaden
  - d.h. Nachfolger an das Ende der Queue



Suchstrategien — Breitensuche

- Idee: Expandiere Blatt mit minimaler Tiefe
- Implementierung: FIFO-Queue bestehend aus Teilpfaden
  - d.h. Nachfolger an das Ende der Queue



Suchstrategien — Breitensuche

Algorithmus zur Breitensuche [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) Append the new paths, if any, to the end of the queue.
3. If the goal node is found, announce success; otherwise announce failure.

Suchstrategien — Breitensuche

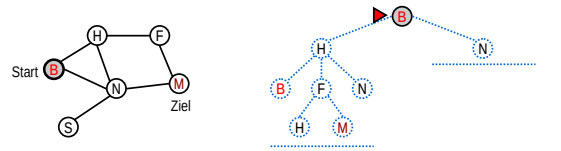
- Leistungsmessung der Breitensuche
  - b = Verzweigungsgrad
  - d = Tiefe des Suchbaumes entlang optimaler Lösung

Vollständigkeit	Ja
Optimalität	Ja
Zeitkomplexität	$O(b^{d+1})$
Speicherkomplexität	$O(b^{d+1})$

Für endlichen Verzweigungsgrad b und identische Kosten

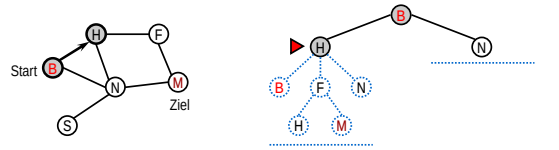
Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



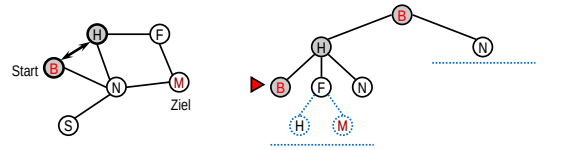
Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



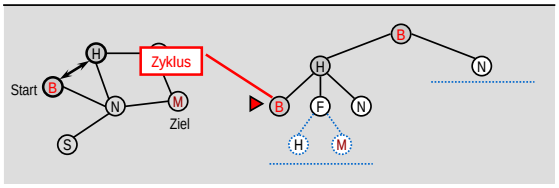
Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



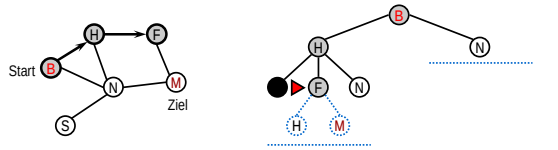
Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



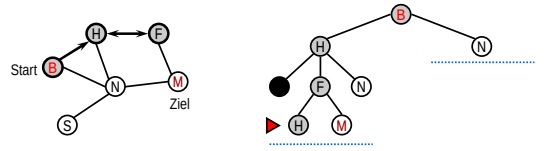
Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



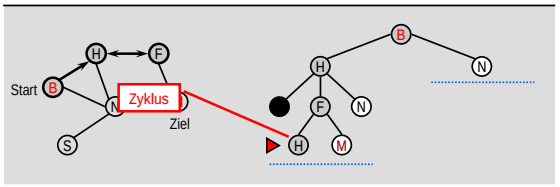
Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



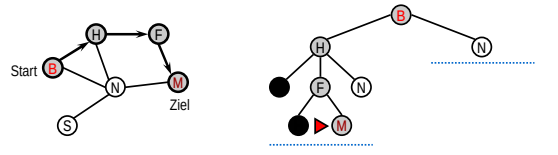
Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



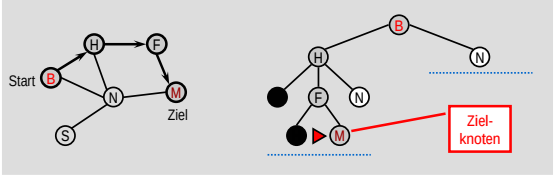
Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



Suchstrategien — Tiefensuche

- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



Suchstrategien — Tiefensuche

Algorithmus zur Tiefensuche [Winston 93]

1. Form a one-element stack consisting of a zero-length path that contains only the root node.
2. Until the first path in the stack terminates at the goal node or the queue is empty
   a) Remove the first path from the stack; create new paths by extending the first path to all neighbors of the terminal node.
   b) Reject all paths with loops.
   c) Push the new paths, if any, to the top of the stack.
3. If the goal node is found, announce success; otherwise announce failure.

Suchstrategien — Tiefensuche

- Leistungsmessung der Tiefensuche
  - b = Verzweigungsgrad
  - m = maximale Tiefe des Suchbaums

Vollständigkeit	Nein
Optimalität	Nein
Zeitkomplexität	$O(b^m)$
Speicherkomplexität	$O(bm)$

Vollständigkeit nur bei beschränkten Suchbäumen

Suchstrategien — Limitierte Tiefensuche

- Problem der Tiefensuche:
  - Exploration langer Teilpfade, die nicht zum Ziel führen
  - Folge: Hohe Zeitkomplexität
- Lösung: Limitierte Tiefensuche
  - Tiefensuche bis zu einer limitierten Tiefe l
  - Knoten der Tiefe l werden nicht weiter expandiert

Suchstrategien — Limitierte Tiefensuche

Algorithmus zur limitierten Tiefensuche mit Limit **l**

1. Form a one-element stack consisting of a zero-length path that contains only the root node.
2. Until the first path in the stack terminates at the goal node or the queue is empty
   a) Remove the first path from the stack.
   b) If the first path has length **l**, then continue with step 2.
   c) Create new paths by extending the first path to all neighbors of the terminal node.
   d) Reject all paths with loops.
   e) Push the new paths, if any, to the top of the stack.
3. If the goal node is found, announce success; otherwise announce failure.

Suchstrategien — Limitierte Tiefensuche

- Leistungsmessung der limitierten Tiefensuche
  - b = Verzweigungsgrad
  - l = limitierte Tiefe des Suchbaums

Vollständigkeit	Nein
Optimalität	Nein
Zeitkomplexität	$O(b^l)$
Speicherkomplexität	$O(bl)$

Suchstrategien — Iterative Tiefensuche

- Nachteile der Tiefensuche & limitierten Tiefensuche:
  - keine Vollständigkeit
  - keine Optimalität
- Lösung: Iterative Tiefensuche
  - Wiederholte limitierte Tiefensuche mit steigender Tiefe l
- Algorithmus zur iterativen Tiefensuche

  For $l = 0$ to ∞ do
  1. Limitierte Tiefensuche until depth l
  2. If the goal node is found, then announce success and exit.

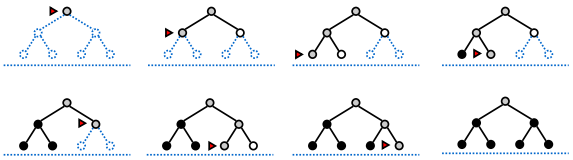
Suchstrategien — Iterative Tiefensuche

Limit = 0

Limit = 1

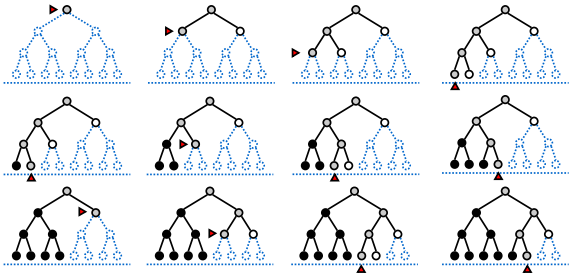
Suchstrategien — Iterative Tiefensuche

Limit = 2



Suchstrategien — Iterative Tiefensuche

Limit = 3



Suchstrategien — Iterative Tiefensuche

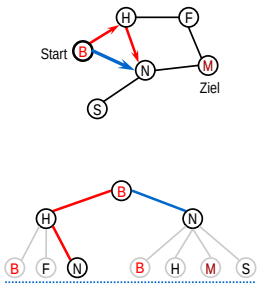
- Leistungsmessung der iterativen Tiefensuche
  - b = Verzweigungsgrad
  - d = Tiefe des Suchbaumes entlang optimaler Lösung

Vollständigkeit	Ja
Optimalität	Ja
Zeitkomplexität	$O(b^d)$
Speicherkomplexität	$O(bd)$

Für endlichen Verzweigungsgrad b und identische Kosten

Suchstrategien — Dynamische Programmierung

- Problem:
  - Suchverfahren verfolgen mehrere Teilpfade vom Startknoten B über einem Zwischenknoten N weiter.
  - Ineffizient bzgl. Zeit und Speicher
- Lösung: Dynamische Programmierung
  - Der kürzeste Pfad von B nach **M** via N setzt sich zusammen aus dem kürzesten Pfad von B nach N und dem kürzesten Pfad von N nach **M**.
  - Lösche alle Teilpfade von B nach N bis auf den Kürzesten.



Gliederung

- Problemlösungsagenten
- Problemformulierung
- Suche nach Lösungen
- Zusammenfassung und Ausblick

Zusammenfassung

- Bevor ein Agent nach Lösungen suchen kann, muss er das Problem formulieren.
- Ein Problem besteht aus einer Menge von Anfangszuständen, einer Menge von Endzuständen und einer Menge von Aktionen.
- Ein generischer Suchalgorithmus kann ein beliebiges Problem lösen. Verschiedene Suchalgorithmen unterscheiden sich in der Strategie mit der Knoten expandiert werden.
- Suchverfahren werden bezüglich Vollständigkeit, Optimalität, Zeitkomplexität und Speicherkomplexität bewertet.

Ausblick

- Diese Vorlesung: Uninformierte Verfahren mit identischen Kosten
- Nächste Vorlesung: Suchverfahren mit folgenden Erweiterungen:
  - Aktionen mit unterschiedlichen Kosten
  - Schätzungen des Restwegs zum Ziel (informierte Suche)

DAI-Labor
TU Berlin

Informierte Suchverfahren

nächster Termin: 03.10.2005

Brijnesh J Jain
bjj@dai-labor.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.