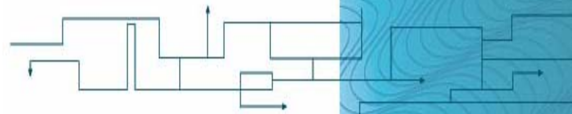
DAI-Labor
TU Berlin



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

Lernziele: Wir diskutieren, wie ein Agent Entscheidungen durch systematisches Suchen trifft. Entscheidungen werden getroffen, um ein vorgegebenes Ziel zu erreichen.

Gliederung

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

Die Vorlesung ist wie folgt strukturiert:

1. Problemlösungsagenten: Wir beschreiben Problemlösungsagenten, die zur Klasse der zielbasierten Agenten (s. letzte Vorlesung) gehören. Ein Problemlösungsagent ist ein Agent, der Entscheidungen in Form einer Folge von Aktionen trifft, um einen gewünschten Zielzustand zu erreichen.
2. Problemformulierung: Um ein konkretes Ziel zu erreichen, benötigen wir einen Formalismus der die „weltliche“ Zielformulierung in eine maschinen-verarbeitbare Form überführt. Diesen Prozess nennt man Problemformulierung. Problemformulierung legt Zustände und Aktionen, die einen Zustand in einen anderen überführen fest.
3. Im dritten Teil behandeln wir Suchverfahren, die von einem gegeben Anfangszustand eine Folge von Aktionen suchen, deren Anwendung den Anfangzustand in einen Zielzustand überführt.
4. Zusammenfassung: Abschließend fassen wir die wichtigsten Punkte zusammen und geben ein 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

Die zentralen Fragen mit denen wir uns beschäftigen sind

1. Was ist ein Problem?
2. Was ist eine Lösung?
3. Wie findet man eine Lösung?

Wir werden im folgenden die drei Fragen über einen agentenbasierten Ansatz beantworten. Dazu betrachten wir so genannte Problemlösungsagenten, die als eine spezielle Form von zielorientierten Agenten betrachtet werden können. Die hier betrachteten Problemlösungsagenten folgen einem formulate-search-execute Design, dass wir uns jetzt mal anhand eines Beispiels genauer anschauen wollen.

Problemlösungsagenten

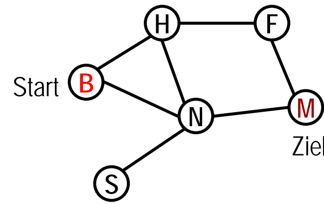
Beispiel für einen Problemlösungsagenten

⇨ **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 Aktionen 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

A|O|T Grundlagen der Künstlichen Intelligenz © B.J. Jain 4

Betrachten wir dazu das einfache Problem, mit dem Auto von Berlin nach München zu reisen. Die Graphik auf der linken Seite zeigt ein vereinfachtes Model einer Deutschlandkarte. Knoten repräsentieren Städte und Kanten Straßenverbindungen zwischen Städten. (Jetzt könnten erwähnt werden, welche Städte auf der Karte zu sehen sind) Entfernungen werden in dieser Vorlesung nicht berücksichtigt. Das behandeln wir nächste Woche.

Ein Problemlösungsagent mit *formulate-search-execute* Design verfährt nach folgendem Schema:

1. Zunächst wird das Ziel formuliert: Fahre von Berlin nach München
2. Anschließend transformiert der Agent die Zielformulierung in eine adäquate Problemformulierung, bei der die möglichen Zustände und Aktionen festgelegt werden (Zustände und Aktionen gemäß Folie erwähnen). Sowohl Ziel- als auch Problemformulierung gehören zur formulate-Phase des Agenten.
3. In der search-Phase sucht der Agent eine Sequenz von Aktionen, die angewendet auf den Anfangszustand (Berlin) in den Zielzustand (München) führen. Beispiel: Fahren von Berlin nach Nürnberg und dann von Nürnberg nach München.
4. Schließlich in der execute-Phase wird die gefundene Lösung usgeführt.

Das Beispiel sollte uns eine Idee eines Problemlösungsagenten mit formulate-search-execute Design geben.

Problemlösungsagenten

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

Die wichtigsten Entwurfsentscheidungen, die wir zu treffen haben sind die Problemformulierung und die Suche nach einer Lösung. Beides ist problemabhängig und mehr eine Kunst als eine Wissenschaft. Insbesondere von der Problemformulierung hängt die Leistungsfähigkeit eines Problemlösungsagenten ab. Bei einer idealen Problemformulierung fällt uns die Lösung gewissermaßen in den Schoß. Bei einer schlechten Problemformulierung wird die Suche nach einer Lösung zu einem komplizierten und aufwändigen Unterfangen.

Die Wahl oder der Entwurf eines Suchverfahrens hängt nicht nur vom Problem selbst sondern u.a. auch von der Problemformulierung ab.

Problemformulierung und Suchalgorithmen sind die zentralen Themen mit denen wir uns im folgenden auseinandersetzen.

Gliederung

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

Wir beginnen mit der Problemformulierung. Dazu werden wir einige Formalismen zur Beschreibung oder Formulierung von Problemen einführen.

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)

A|O|T Grundlagen der Künstlichen Intelligenz © B.J. Jain 7

Folie vorlesen.

Bemerkungen:

1. Es kann mehrere Anfangszustände geben. Beispiel: Finde Wege von Berlin und Hannover nach München.
2. Es kann mehrere Zielzustände geben. Beispiel: 8-Dame Problem (siehe Norvig). Positioniere 8 Damen auf dem Schachbrett, so dass keine Dame eine andere schlagen kann. Für dieses Problem gibt es mehrere Konfigurationen. Jede dieser Konfigurationen ist ein Zielzustand.
3. Eine Aktion muss nicht nur auf einen Zustand sondern kann auf einer Teilmenge von Zuständen definiert sein. Zum Beispiel auf einem Schachbrett den König um eine Position nach links zu schieben, kann von mehreren Feldern ausgeführt werden, sofern die Anwendungsbedingungen gemäß der Regeln des Schachs erfüllt sind.

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

Wenn wir Zustände und Aktionen definiert haben, können wir uns nun damit befassen, wie man von einem gegebenen Anfangszustand in einen Zielzustand kommt.

Punkte auf Folie vorlesen.

Problemformulierung

Beispiele

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

Anwendungsbeispiele: Siehe Russel & Norvig, Kapitel 3.2, Seite 64ff, für Beschreibungen der Probleme

Wir betrachten als Beispiel dafür, wie man ein Problem formuliert das 8-Puzzle Problem. Wie bereits erwähnt ist eine adäquate Problemformulierung eine wichtige Entwurfsentscheidung, die abhängig von der jeweilige Anwendung ist. Deswegen ist es völlig in Ordnung, Spielzeugbeispiele wie das 8-Puzzle Problem statt echte Anwendungsbeispiele zur Illustrierung von Konzepten zu betrachten.

Problemformulierung — Beispiele

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

7	2	4
5		6
8	3	1

Anfangszustand

	1	2
3	4	5
6	7	8

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*)

Punkte der Folie vorlesen.

8-Puzzle Problem beschreiben (s. Russel & Norvig, S.64f)

Bemerkungen:

1. Ein Zustand kann durch die Koordinaten (Zeile, Spalte) der Kacheln und der Leerstelle beschrieben werden.
2. Der Endzustand kann nicht von jedem Anfangszustand erreicht werden.
3. Ein zulässiger Zug ist ein Zug dessen Resultat ein zulässigen Zustand ist. Man darf also das Blank nicht nach oben bewegen, wenn es bereits in der obersten Reihe ist.
4. Andere Endzustände sind möglich

Gliederung

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

Soeben haben wir die erste Entwurfsentscheidung diskutiert. Wie abstrahiert man eine Zielformulierung zu einer Problemformulierung. Im zweiten Hauptteil dieser Vorlesung beschäftigen wir uns nun damit, wie man zu einem gegebenen Problem eine Lösung findet. Dieses machen wir durch Suche im Problemraum. Die systematische Suche generiert Suchbäume. Wir geben Kriterien an, die uns sagen, wie effizient die systematische Suche ist. Unterschiedliche Verfahren unterscheiden sich in der Strategie, mit der ein Suchbaum aufgebaut wird.

Suche nach Lösungen

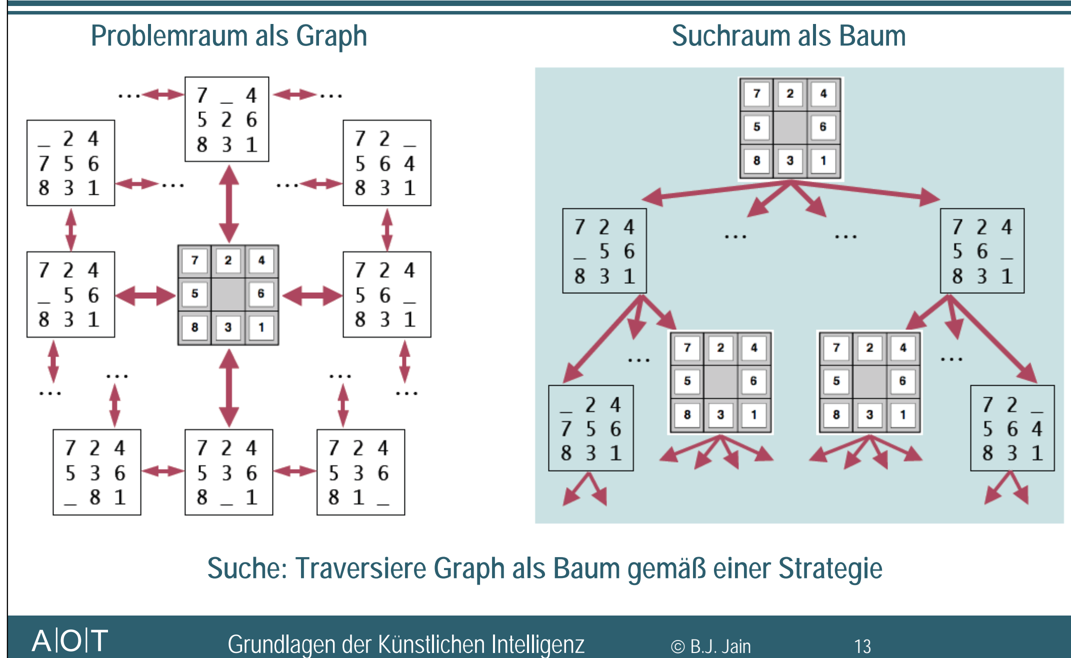
Suchbäume

- **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**

Wir gehen im folgenden davon aus, dass sowohl die Menge der Anfangszustände als auch die Menge der Zielzustände aus jeweils genau einem Element bestehen.

Wie können wir eine Lösung finden, d.h. Eine Folge von Aktionen, die den Anfangzustand in einen Endzustand überführt? Die Idee ist Exploration des Zustandsraums durch Erzeugen von Folgezuständen bereits untersuchter Zustände.

Weitere Punkte der Folie abarbeiten.



Lasst uns die Suche veranschaulichen. Wir können uns den Problemraum als gerichteten Graphen vorstellen. Knoten repräsentieren Zustände und Kanten Aktionen. Hier sehen wir auf der linken Seite einen Ausschnitt des Problemraums repräsentiert als Graph. Die grau eingefärbte Konfiguration in der Mitte der Graphik ist unser Anfangszustand.

Auf der rechten Seite sehen wir einen Suchbaum. Knoten repräsentieren besuchte Zustände und Kanten angewendete Aktionen. Die Wurzel repräsentiert den Anfangszustand.

Such bedeutet, dass wir den Graphen als Suchbaum gemäß einer bestimmten Strategie traversieren.

Sehr wichtige Bemerkung: Weder der Graph noch der vollständige Suchbaum liegen i.a. explizit vor. Der Suchbaum wird während der Suche schrittweise aufgebaut.

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.
   c) Color current leaf as grey
   d) Expand current leaf and color its child-nodes as white leafs.
4. Return failure.

Nun wollen wir einen informelle Beschreibung eines Suchbaum-Verfahrens angeben. Wir färben Knoten des Suchbaumes weiß, wenn wir sie zwar besucht aber auf diesen noch keine Aktion angewendet haben. Wir färben Blätter grau, wenn wir sie zur Expansion ausgewählt haben.

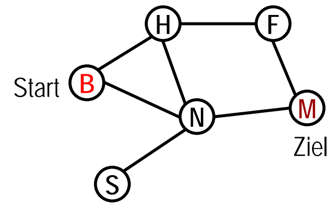
Algorithmus beschreiben (s. Folie).

Suche nach Lösungen

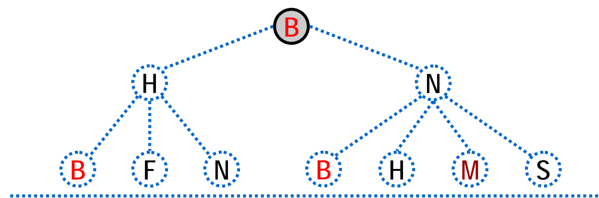
Suchbäume

Beispiel

⇨ Problem: Shortest Path



(a) **Startzustand** (nach Schritt 3c)

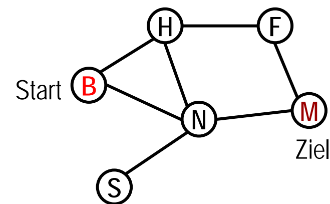


Betrachten wir als Beispiel unser Problem von Berlin nach München zu gelangen. Die Strategie mit der wir einen Knoten expandieren erfolgt nach lexikographischer Ordnung der Knoten-Bezeichner unserer Landkarte.

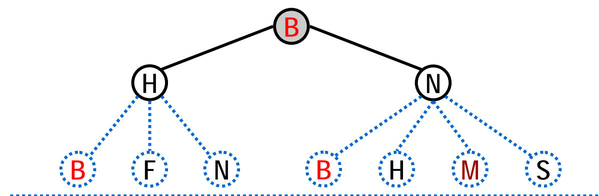
Anfangszustand ist der Knoten B für Berlin. Wir färben die Wurzel grau ein (nach Schritt 3c), weil wir im Begriff sind den Knoten B zu expandieren.

Beispiel

⇨Problem: Shortest Path



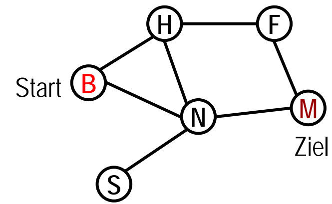
(b) nach Expansion von B (nach Schritt 3d)



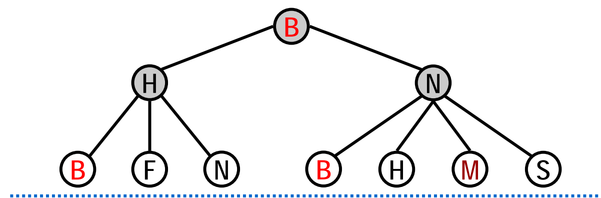
Wenden wir alle möglichen Aktionen auf den Knoten B an (Schritt 3d), erhalten wir die Knoten H (Hannover) und N (Nürnberg) als Nachfolger. Wir färben beide Knoten im Suchbaum weiß ein.

Beispiel

⇨Problem: Shortest Path



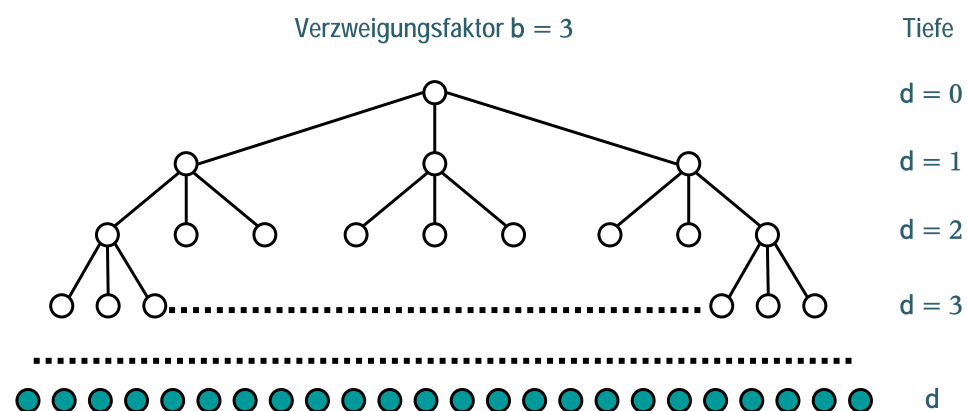
(c) nach Expansion von H und N



Hier sehen wir nun den Suchbaum, nachdem wir die Knoten H und N expandiert haben.

⇨ **Naive Suche**

→ Alle Sequenzen von Aktionen der Reihe nach durchprobieren

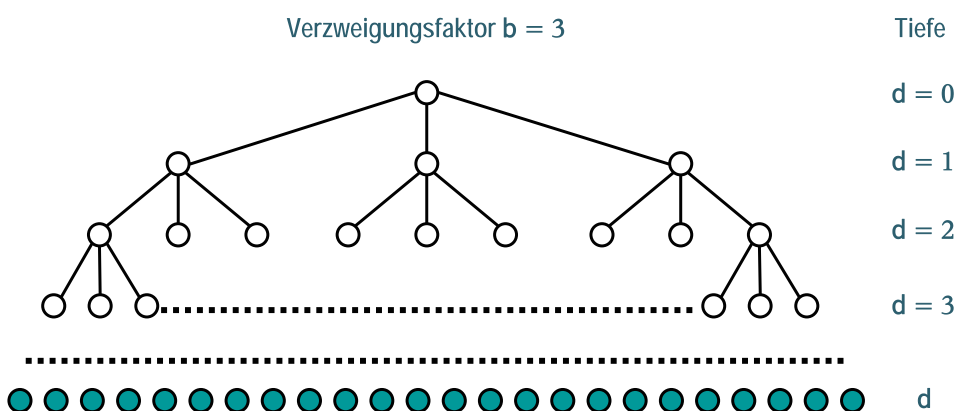


Bei einer naiven Suche probieren wir alle Sequenzen von Aktionen der Reihe nach durch und wählen dann die Sequenz aus, die das Problem löst. Welchen Aufwand haben wir dabei zu erwarten? Dazu betrachten wir einen Suchbaum mit Verzweigungsfaktor b und Tiefe d. Der Verzweigungsfaktor ist die maximale Anzahl der Nachfolger der Knoten im Suchbaum. Im Beispiel ist der Verzweigungsfaktor b = 3. Die Tiefe eines Baumes gibt die maximale Pfadlänge von Wurzel zu einem Blatt des Baumes an. Ein Blatt ist ein Knoten ohne Nachfolger.

Suche nach Lösungen

Suchbäume

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



A|O|T Grundlagen der Künstlichen Intelligenz © B.J. Jain 19

Die Anzahl der Knoten in Tiefe d bei einem Verzweigungsfaktor b ist **„b hoch d"** (b^d)

Am besten an der Tafel vorrechnen: Zum Beispiel für die Tiefe

d = 0 haben wir b^d = 3^0 = 1

d = 1 haben wir b^d = 3^1 = 3

…

Die Anzahl der Knoten von Tiefe 0 bis Tiefe d ist dann (siehe entsprechende Formel auf Folie). Der Bruch ergibt sich durch Induktion aus der endlichen Summe (Geometrische Reihe, s. Analysis 1).

Suche nach Lösungen

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

© B.J. Jain

Was haben die Anzahl der Knoten **in** bzw. **bis** Tiefe d mit Effizienz eines Suchbaum-Verfahrens zu tun?

Zeit messen wir hier durch die Anzahl der durch das Verfahren erzeugten Knoten. Das entspricht in unserem Fall die Anzahl der Knoten **bis** Tiefe d

Speicheraufwand messen wir durch die maximale Anzahl der Knoten im Speicher. Das würde der Anzahl der Knoten **in** Tiefe d entsprechen.

Somit haben wir O(b^d) als Zeit- und Speicheraufwand. Die Frage ist, gibt es bessere Suchstrategien? Dazu müssen wir zunächst klarstellen, was „besser" eigentlich bedeutet. Anschließend betrachten wir Suchstrategien und bewerten sie hinsichtlich ihrer Leistungsfähigkeit.

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?

Um sagen zu können wie leistungsfähig ein Algorithmus ist oder um von einem besseren Algorithmus sprechen zu können, benötigen wir Kriterien hinsichtlich derer wir Algorithmen bewerten und vergleichen. Wir betrachten dabei folgende Kriterien: Siehe Folie.

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 (**b**ranching factor)
  - d = Tiefe (**d**epth)
  - m = maximale Tiefe des Suchbaums

Für Zeit und Speicherkomplexität benötigen wir quantitative Maße. Dabei verwenden wir für den Zeitaufwand die Gesamtzahl der erzeugten Knoten und für den Speicheraufwand die maximale Anzahl der Knoten im Speicher.

Da der Suchbaum a priori nicht gegeben ist sondern für jedes Problem schrittweise aufgebaut ist, können wir den Aufwand nur abschätzen. Als Faktoren zur Bestimmung des Zeit- und Speicheraufwads verwenden wir b, d, m (siehe Folie).

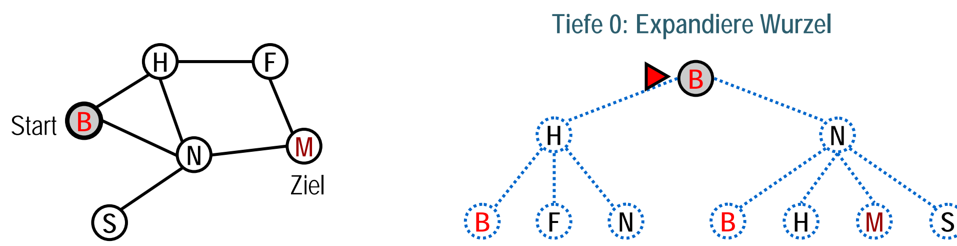
Gliederung

- 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



Als erstes Suchverfahren schauen wir die Breitensuche an.

Grundidee erwähnen (s. Folie).
Datenstruktur und Speicherprinzip erwähnen (s. Folie).

Wir illustrieren die Funktionsweise der Breitensuche an einem Beispiel.

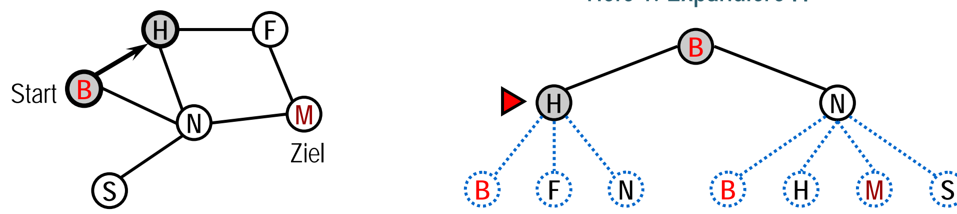
Simulation anhand dieser und der folgenden Folien erklären.

Bemerkungen zum Suchbaum:

1. <u>Der rote Pfeil zeigt auf das aktuelle zu expandierende Blatt.</u>
2. <u>Knoten des aktuellen Teilpfads sind grau gefärbt.</u>
3. Neu erzeugte aber nicht aktuelle_Blätter sind weiß gefärbt.
4. Blätter, die nicht weiter expandiert werden können sind schwarz gefärbt

Suchstrategien — Breitensuche

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



A|O|T Grundlagen der Künstlichen Intelligenz © B.J. Jain 26

Simulation erklären.

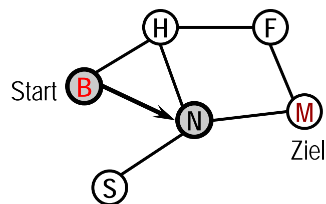
Bemerkungen zum Suchbaum:

1. <u>Der rote Pfeil zeigt auf das aktuelle zu expandierende Blatt.</u>
2. <u>Knoten des aktuellen Teilpfads sind grau gefärbt.</u>
3. <u>Neu erzeugte aber nicht aktuelle Blätter sind weiß gefärbt.</u>
4. Blätter, die nicht weiter expandiert werden können sind schwarz gefärbt

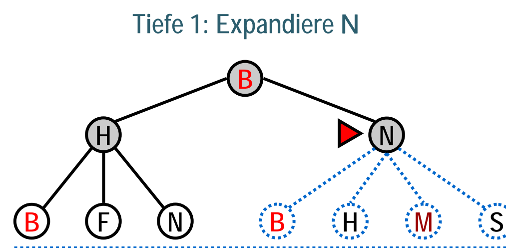
Suchstrategien

Breitensuche

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



Tiefe 1: Expandiere N



Simulation erklären.

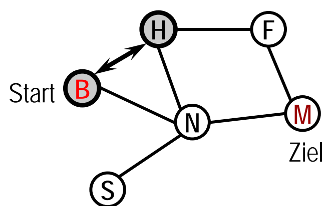
Bemerkungen zum Suchbaum:

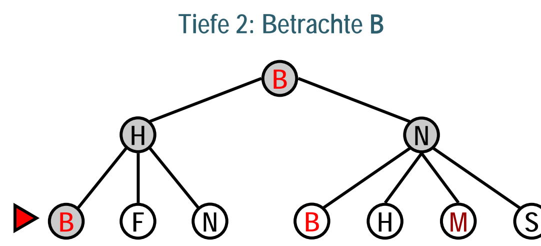
1. <u>Der rote Pfeil zeigt auf das aktuelle zu expandierende Blatt.</u>
2. <u>Knoten des aktuellen Teilpfads sind grau gefärbt.</u>
3. <u>Neu erzeugte aber nicht aktuelle Blätter sind weiß gefärbt.</u>
4. Blätter, die nicht weiter expandiert werden können sind schwarz gefärbt

Suchstrategien

Breitensuche

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





Simulation erklären.

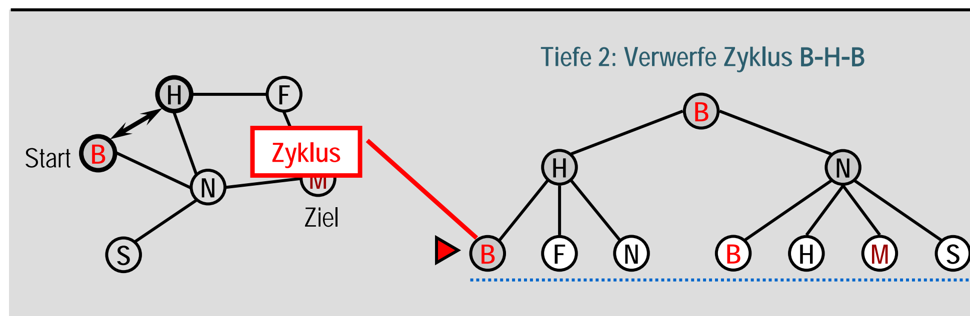
Bemerkungen zum Suchbaum:

1. <u>Der rote Pfeil zeigt auf das aktuelle zu expandierende Blatt.</u>
2. <u>Knoten des aktuellen Teilpfads sind grau gefärbt.</u>
3. <u>Neu erzeugte aber nicht aktuelle Blätter sind weiß gefärbt.</u>
4. Blätter, die nicht weiter expandiert werden können sind schwarz gefärbt

Suchstrategien

Breitensuche

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

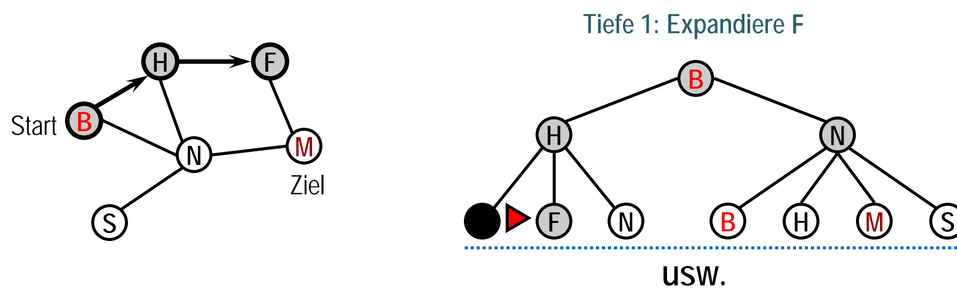


Simulation erklären.

Bemerkungen zum Suchbaum:

1. <u>Der rote Pfeil zeigt auf das aktuelle zu expandierende Blatt.</u>
2. <u>Knoten des aktuellen Teilpfads sind grau gefärbt.</u>
3. <u>Neu erzeugte aber nicht aktuelle Blätter sind weiß gefärbt.</u>
4. Blätter, die nicht weiter expandiert werden können sind schwarz gefärbt

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

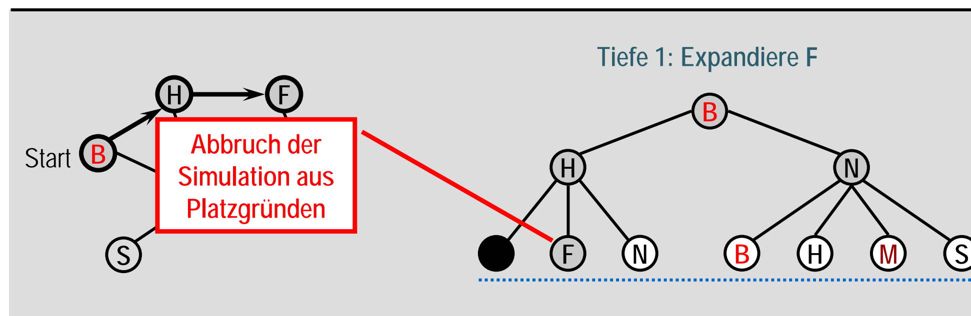


Simulation erklären.

Bemerkungen zum Suchbaum:

1. Der rote Pfeil zeigt auf das aktuelle zu expandierende Blatt.
2. Knoten des aktuellen Teilpfads sind grau gefärbt.
3. Neu erzeugte aber nicht aktuelle Blätter sind weiß gefärbt.
4. Blätter, die nicht weiter expandiert werden können sind schwarz gefärbt

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



Simulation wird an dieser Stelle aus Platzgründen abgebrochen. Es sollte dennoch klar sein, wie die Breitensuche weiter verfahren würde.

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.

A|O|T Grundlagen der Künstlichen Intelligenz © B.J. Jain 32

Wir fassen die Breitensuche nun in algorithmischer Form zusammen.

Algorithmus gemäß Folie beschreiben.

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

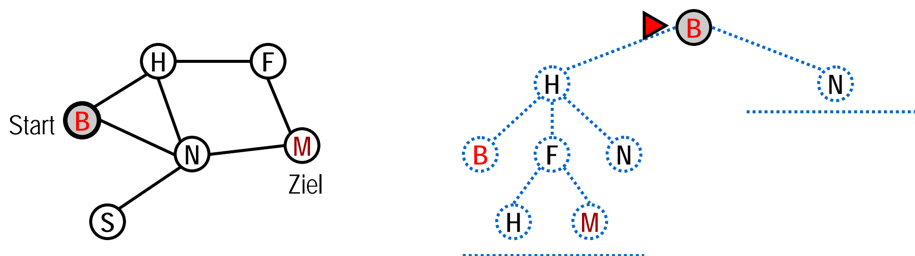
Wie „gut" is Breitensuche? Dazu bewerten die Leistungsfähigkeit der Breitensuche hinsichtlich unser vier Kriterien Vollständigkeit, Optimalität, Zeitkomplexität und Speicherkomplexität.

Formeln an der Tafel vorrechnen und erklären (s. Norvig, S. 74)

Suchstrategien

Tiefensuche

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



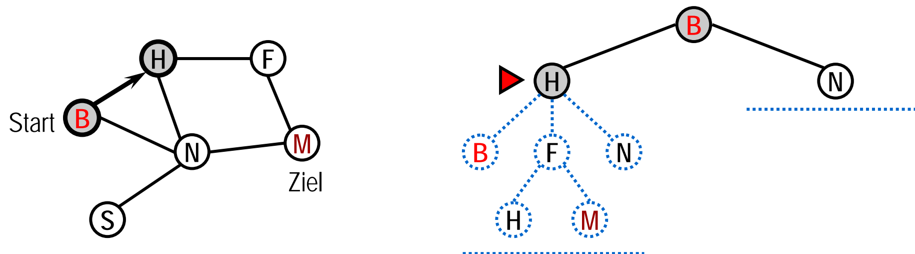
Nun schauen wir uns die Tiefensuche an.

Nach dem gleichen Muster wie die Breitensucher erklären

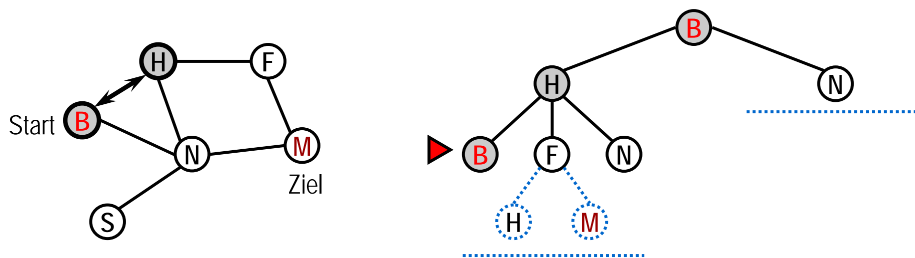
⇨ Idee: Expandiere Blatt mit maximaler Tiefe

⇨ Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden

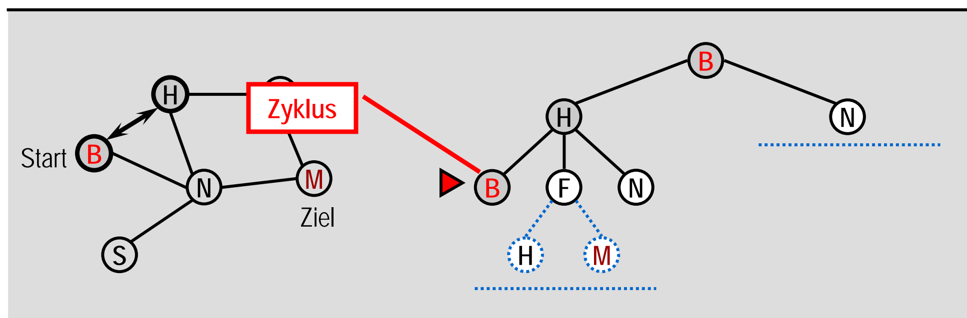
- d.h. Nachfolger an den Anfang der Queue



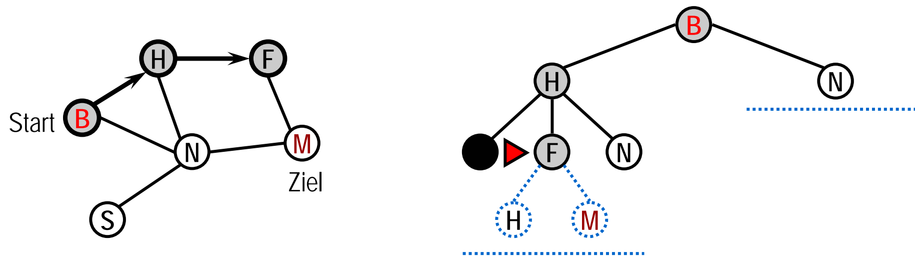
- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



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



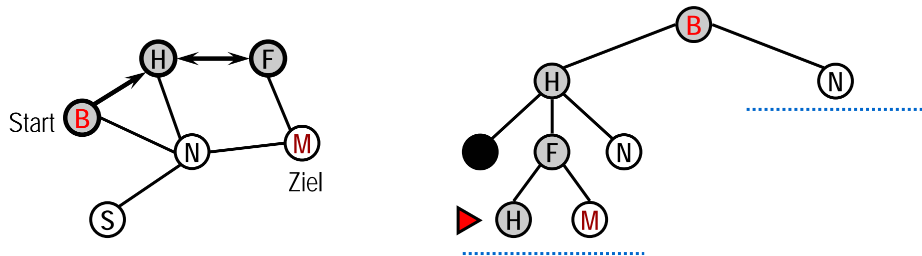
- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



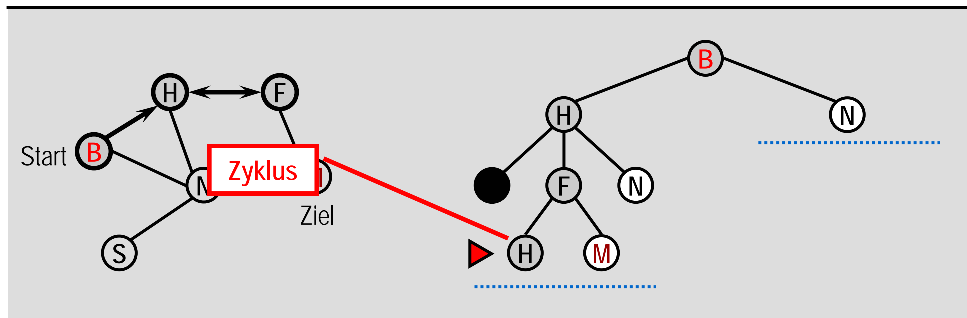
⇨ Idee: Expandiere Blatt mit maximaler Tiefe

⇨ Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden

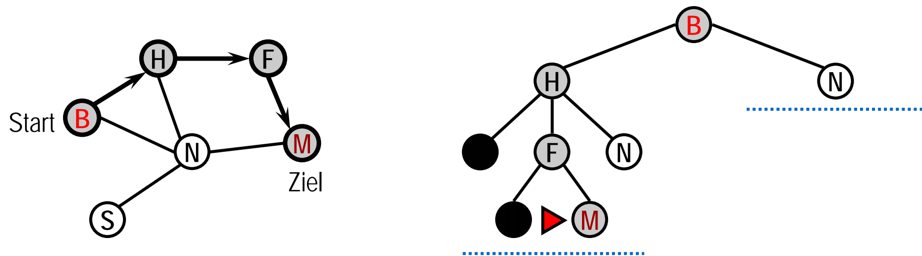
➞ d.h. Nachfolger an den Anfang der Queue



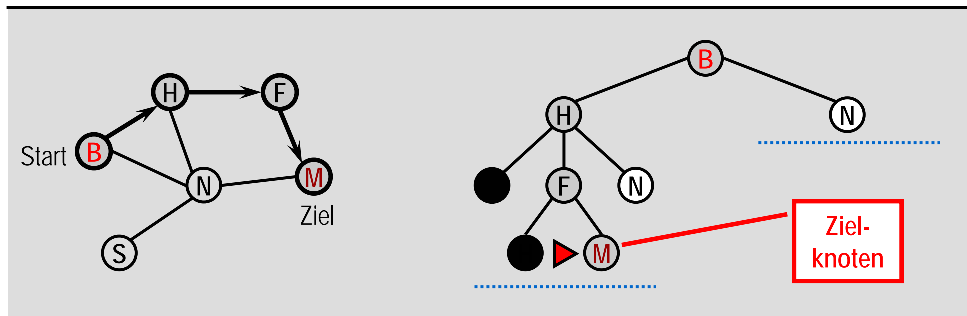
- Idee: Expandiere Blatt mit maximaler Tiefe
- Implementierung: LIFO-Queue (Stack) bestehend aus Teilpfaden
  - d.h. Nachfolger an den Anfang der Queue



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



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



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.

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

Die Tiefensuche hat ein Probleme (s. Folie).

Um dieses Problem zu umgehen betrachten wir die Limitierte Tiefensuche.
Grundidee gemäß Folie beschreiben.

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.

Die limitierte Tiefensuche entspricht also der Tiefensuche, wobei nach Schritt 2a ein Test eingebaut wird, der überprüft ob das Limit erreicht ist (der Test ist rot hervorgehoben).

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.

A|O|T Grundlagen der Künstlichen Intelligenz © B.J. Jain 48

Sowohl Tiefen- als auch die limitierte Tiefensuche sind weder vollständig noch optimal. Ein Ausweg ist die iterative Tiefensuche.

Grundidee der iterativen Tiefensuche gemäß Folie beschreiben

Algorithmus beschreiben

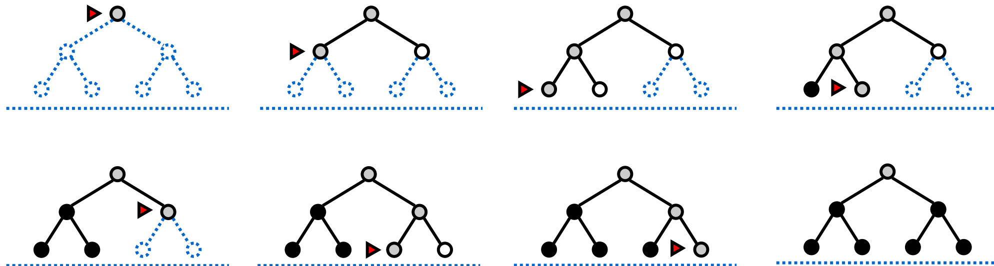
Limit = 0

Limit = 1

Wir illustrieren die iterative Tiefensuche anhand eines fiktiven Beispiels.

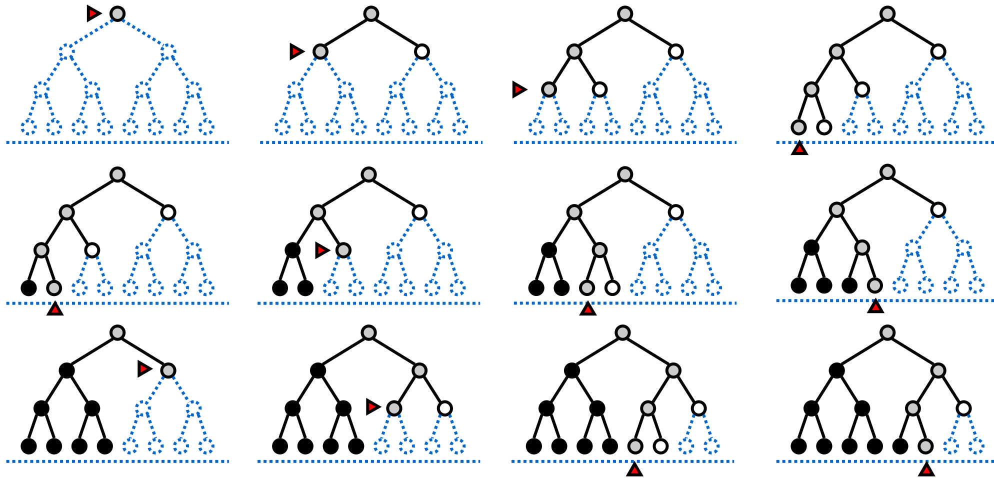
Simulation erklären

Limit = 2



Simulation erklären

Limit = 3



Simulation erklären.

Leistungsmessung der iterativen Tiefensuche

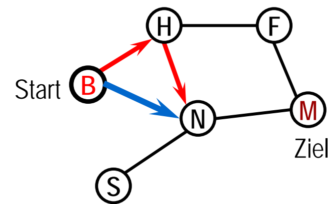
- b = Verzweigungsgrad
- d = Tiefe des Suchbaumes entlang optimaler Lösung

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

Für endlichen Verzweigungsgrad b und identische Kosten

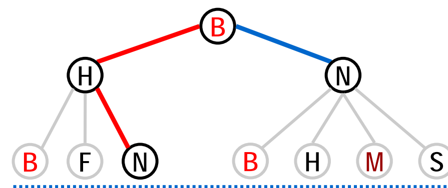
⇨ Problem:

- 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.



Wir haben nun eine Reihe von Verfahren vorgestellt, die folgendes Problem haben (siehe Folie).

Lösung des Problems gemäß Folie & Beispiel beschreiben.

Gliederung

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

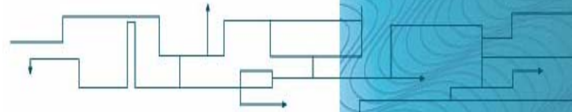
Zusammenfassung

- ⇨ 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.