Suchverfahren (Grobe Klassifizierung)

	Nach irgendeinem Weg	Nach einem besten Weg (mit Gütefunktion)
Ohne Heuristik	z.B. Tiefe-Zuerst	z.B. Branch and Bound
Mit Heuristik (mit Schätzfunktion)	z.B. Bergsteigen	z.B. A*

Aufgaben von Suchalgorithmen

- Probleme:
  - Existiert ein Weg von z_0 zu einem $z_f \in Z_f$
  - Konstruiere einen Weg von z_0 zu einem $z_f \in Z$
  - Konstruiere optimalen Weg von z_0 zu einem $z_f \in Z_f$

Güte von Suchalgorithmen

Bezogen auf Konstruktion von Wegen zum Ziel:

- **Korrektheit:**
  - Algorithmus liefert nur korrekte Wege.
- **Vollständigkeit:**
  - Algorithmus liefert (mindestens) alle korrekten Wege.
- **Optimalität:**
  - Algorithmus liefert optimale Wege.

Vollständigkeit kann auch schwächer gefasst werden (vgl. Existenzproblem): Algorithmus liefert einen korrekten Weg, falls eine Lösung existiert.

Komplexität von Suchalgorithmen

- bzgl. Komplexität des Verfahrens:
  - Zahl der Zustände insgesamt
  - Zahl der erreichbaren Zustände
  - Zahl der untersuchten Zustände
  - Suchtiefe
- bzgl. Gefundener Lösung

Graph: G = [Z,E] mit

–Anfangszustand $z_0 \in Z$

–Zielzuständen $Z_f \subseteq V$

Zyklen, Maschen im Suchraum

Prolog:

```
erreichbar(X,Y) :- erreichbar(X,Z), nachbar(Z,Y).
erreichbar(X,X).

symmetrisch(X,Y):- symmetrisch(Y,X).
```

Test auf Wiederholungen:

Zeit-, Speicher- aufwändig

Beschränkung der Suchtiefe

Suche nach einem Weg

Expansion: Schrittweise Konstruktion des Zustandsraums

Datenstrukturen :

- Liste OPEN:

Ein Zustand (Knoten) heißt "offen" , falls er bereits konstruiert, aber noch nicht expandiert wurde (Nachfolger nicht berechnet)

- Liste CLOSED:

Ein Zustand (Knoten) heißt "abgeschlossen" , falls er bereits vollständig expandiert wurde (Nachfolger alle bekannt)

Zusätzliche Informationen:
z.B. Nachfolger/Vorgänger der Knoten
(für Rekonstruktion gefundener Wege)

Schema S (Suche nach irgendeinem Weg)

S0: (Start) Falls Anfangszustand z_0 ein Zielzustand: EXIT(„yes:“ z_0).
OPEN := [z_0] , CLOSED := [ ] .
S1: (negative Abbruchbedingung) Falls OPEN = [ ] : EXIT(„no“).
S2: (expandieren)
Sei z der erste Zustand aus OPEN.
OPEN := OPEN – {z} . CLOSED := CLOSED $\cup${z} .
Bilde die Menge Succ(z) der Nachfolger von z.
Falls Succ(z) = {} : Goto S1.
S3: (positive Abbruchbedingung)
Falls ein Zustand z_1 aus Succ(z) ein Zielknoten ist: EXIT(„yes:“ z_1).
S4: (Organisation von OPEN)
Reduziere die Menge Succ(z) zu einer Menge NEW(z)
durch Streichen von nicht weiter zu betrachtenden Zuständen.
Bilde neue Liste OPEN durch Einfügen der Elemente aus NEW(z) .
Goto S1.

Variable Komponenten in Schema S :

(Re-)Organisation von OPEN in S4
- V1. Bildung der Menge NEW(z) aus Succ(z) :
  (Auswahl der weiter zu betrachtenden Zustände)
  - alle Zustände aus Succ(z)
  - einige (aussichtsreiche)
  - nur die, die noch nicht in OPEN
  - nur die, die nicht in CLOSED
- V2. Sortierung von OPEN
  (bestimmt den nächsten zu expandierenden Zustand in S2)
  - NEW(z) sortieren
  - NEW(z) einfügen, z.B. an Anfang oder Ende,
  - OPEN (gesamte Liste) neu sortieren
- V3. Weitere Bedingungen
  - Beschränkung der Suchtiefe
  - Reduzierte Menge CLOSED

Blinde Suche mit Test auf Wiederholungen:

(1) Tiefe-Zuerst:
- V1: NEW(z) = Succ(z)-(OPEN ∪CLOSED)
- V2: NEW(z) an den Anfang von OPEN

Keller

(2) Breite-Zuerst:
- V1: NEW(z) = Succ(z)-(OPEN ∪CLOSED)
- V2: NEW(z) an das Ende von OPEN

Warteschlange

Für endliche Graphen:
korrekt und vollständig

Hoher Speicheraufwand
speziell für CLOSED

Vollständig im Sinne:
findet (eine) Lösung im Fall der Existenz

Blinde Suche ohne Test auf Wiederholungen:

Graph als „Abgewickelter Baum“

(1) Tiefe-Zuerst:
- V1: NEW(z) = Succ(z)
- V2: NEW(z) an den Anfang von OPEN

(2) Breite-Zuerst:
- V1: NEW(z) = Succ(z)
- V2: NEW(z) an das Ende von OPEN

Für endliche Graphen:
Tiefe-Zuerst: korrekt, aber nicht immer vollständig
Breite-Zuerst: Korrekt und vollständig

Speicheraufwand für OPEN
- Tiefe-Zuerst: linear $d \cdot b$
- Breite-Zuerst: exponentiell b^d
  (bei b=fan-out, d=Tiefe)

Hoher Zeitaufwand für Wiederholungen

Backtracking

- Implementierung von Tiefe-zuerst-Verfahren
- Spezielle Organisation der Liste OPEN:
  Referenz auf jeweils nächsten zu expandierenden Zustand in jeder Schicht
- Nach Abarbeiten aller Zustände einer Schicht zurücksetzen (backtracking) auf davor liegende Schicht
- Möglichkeit für Zyklenvermeidung mit reduzierter Menge CLOSED (nur für aktuellen Zweig):
  - Beim Backtracking Rücksetzen von CLOSED auf früheren Stand

Iterative Tiefensuche

Stufenweise begrenzte Tiefensuche

- Stufe 1: begrenzte Tiefensuche bis zur Tiefe 1
- Stufe 2: begrenzte Tiefensuche bis zur Tiefe 2
- Stufe 3: begrenzte Tiefensuche bis zur Tiefe 3
- ...

„Depth-first-iterative deepening (DFID)“

DFID bis Tiefe d bei fan-out b erfordert insgesamt

$b^d + 2 \cdot b^{d-1} + 3 \cdot b^{d-2} + ... + i \cdot b$ Schritte

Vergleich mit Tiefe-Zuerst/ Breite-Zuerst bis Tiefe d :

$b^d + b^{d-1} + b^{d-2} + ... + b$ Schritte

DFID hat Speicherbedarf für OPEN wie Tiefe-zuerst

DFID findet Lösung wie Breite-zuerst

Heuristische Suche

Schätzfunktion $\sigma(z)$: geschätzter Konstruktions-Aufwand für Erreichen eines Zielzustandes von z aus

Heuristik: Zustände mit optimaler Schätzung bevorzugen

(3) Bergsteigen/"hill climbing" (ohne Test):

- V1: NEW(z) = Succ(z)
- V2: NEW(z) nach Aufwand sortiert an Anfang von OPEN

„Lokale Optimierung"

(4) Bestensuche (ohne Test):

- V1: NEW(z) = Succ(z)
- V2: OPEN $\cup$ NEW(z) nach Aufwand sortieren

Für endliche Graphen:
korrekt, aber nicht immer vollständig

Typische Probleme lokaler Optimierung

- Vorgebirgsproblem:
  steilster Anstieg führt auf
  lokales Optimum ("Nebengipfel")

- Plateau-Problem:
  keine Unterschiede in der
  Bewertung

- Grat-Problem:
  vorgegebene Richtungen
  erlauben keinen Anstieg

Konsequenz: zwischenzeitliche Verschlechterungen zulassen

Suche nach "bestem Weg“

Bester/optimaler Weg:
Minimale Kosten

Graph: G = [Z,E] mit
–Anfangszustand $z_0 \in Z$
–Zielzuständen $Z_f \subseteq V$

Kosten für Zustandsübergang (Kante)
c: $E \rightarrow R^+$ (Kosten stets positiv!)
mit c (e) = Kosten der Kante $e \in E$
bzw. c(z,z´) = Kosten der Kante e=[z,z´]

Weg-Kosten als Summe von Kosten der Kanten.

Kosten eines Weges $s = e_1 \ldots e_n \in E^*$:

$$c(e_1 \ldots e_n) = \Sigma_{i=1,\ldots,n}\, c(e_i)$$

Kosten eines Weges $s = z_0\, z_1 \ldots z_n \in Z^*$

$$c(z_0\, z_1 \ldots z_n) = \Sigma_{i=1,\ldots,n}\, c(z_{i-1}, z_i)$$

Suche nach "bestem Weg“

Kosten für Erreichen des Zustandes z´ von z aus:

– Falls z´ von z erreichbar:
g(z,z´) := Min{ c(s) / s Weg von z nach z´ },

– Andernfalls: g(z,z´) := ∞

Vorläufigkeit der Kostenberechnung während Expansion:

G´= [Z´, E´] sei (bekannter) Teilgraph von G

g´(z, z´,G´) := Min { c(s) / s Weg in G´ von z nach z´ }

$$g´(z,z´,G´) \geq g(z,z´)$$

Suche nach "bestem Weg“

Verfahren “Generate and Test”:
Alle Wege im Graphen untersuchen.

$L(z_0)$
= Menge der in z_0 beginnenden Wege $p= v_0 ... v_n$

$L(z_0,Z_f)$
= Menge der in z_0 beginnenden Wege $p= v_0 ... v_n$ mit $v_n \in Z_f$

Kürzesten Weg in $L(z_0,Z_f)$ bestimmen.

Suche nach "bestem Weg“

S0: (Start) Falls Anfangszustand z_0 ein Zielzustand: EXIT(„yes:“ z_0).
OPEN := [z_0] , CLOSED := [ ] .

Schema S (Suche nach irgendeinem Weg)

findet eventuell zuerst teure Wege

OPEN := OPEN - {z} . CLOSED := CLOSED ∪{z} .
Bilde die Menge Succ(z) der Nachfolger von z.
Falls Succ(z) = {} : Goto S1.

Lösungsidee:

Abbrechen, wenn alle offenen Wege teurer sind als aktuell gefundene Lösung

Reduziere die Menge Succ(z) zu einer Menge NEW(z)
Durch Streichen von nicht weiter zu betrachtenden Zuständen.
Bilde neue Liste OPEN durch Einfügen der Elemente aus NEW(z) .

dafür:

- Positive Abbruchbedingung von Schema S verändern
- Umstellung der Schritte in Schema S

Schema S‘ für Suche nach "bestem Weg“

S‘0: (Start) Falls Anfangszustand z_0 ein Zielzustand: EXIT(„yes:“ z_0).
OPEN := $[z_0]$, CLOSED := [ ] .
S‘1: (negative Abbruchbedingung) Falls OPEN = [ ] : EXIT(„no“).
S‘2: **(positive Abbruchbedingung)**
Sei z der erste Zustand aus OPEN.
Falls z ein Zielknoten ist: EXIT(„yes:“ z).
S‘3: (expandieren)
OPEN := OPEN - {z} . CLOSED := CLOSED $\cup${z} .
Bilde die Menge Succ(z) der Nachfolger von z.
Falls Succ(z) = {} : Goto S‘1.
S‘4: (Organisation von OPEN)
– g´(z_0 ,z´,G´) für alle z´$\in$Succ(z) berechnen (im aktuellen G´) .
– Neue Liste OPEN durch Einfügen der Elemente aus Succ(z):
Sortieren von OPEN $\cup$ Succ(z) nach aufsteigenden Kosten
Goto S‘1.

Schema S‘ für Suche nach "bestem Weg“

Satz :

Vor.: Es existiert δ >0 mit c(z,z´) > δ für alle Kanten in G

Beh.: Falls Lösung existiert, so findet S‘ einen optimalen Weg

- „Verzweigen und Begrenzen“ (Branch and bound)
- „Dijkstra's Algorithmus“ (1959)

Verbesserungen möglich:
Streichen aus OPEN (bzw. Succ(z)):
– Zustände aus CLOSED
– mehrmaliges Auftreten von Zuständen

- Prinzip der Dynamischen Optimierung/Programmierung

Heuristische Suche nach „bestem Weg“

Problem:

Gleichzeitig **Kostenfunktion** g´(z, z´,G´) *(bisheriger Weg)*

und **Schätzfunktion** $\sigma(z)$ *(zukünftiger Weg)* berücksichtigen

Vorläufigkeit der Kostenberechnung während Expansion:

G´= [Z´, E´] sei (bekannter) Teilgraph von G

g´(z, z´,G´) := Min { c(s) / s Weg in G´ von z nach z´ }

Schätzfunktion $\sigma(z)$: geschätzter Konstruktions-Aufwand für Erreichen eines Zielzustandes von z aus

Heuristik: Zustände mit optimaler Schätzung bevorzugen

Schema S‘‘ für heurist. Suche nach "bestem Weg“

S‘‘0: (Sta … z_0).
OPE

S‘‘1: (ne

S‘‘2: **(positive Abbruchbedingung)**

Sei z d

Falls z ei

S‘‘3: (expandieren)

OPEN := OPEN - {z} . CLOSED := CLOSED $\cup${z} .

Bilde die Menge Succ(z) der Nachfolger von z.

Falls Succ(z) = {} : Goto S‘1.

S‘‘4: (Organisation von OPEN)

– g´(z_0 ,z´,G´) + σ(z) für alle z´$\in$Succ(z) berechnen (im aktuellen G´)

– Neue Liste OPEN durch Einfügen der Elemente aus Succ(z):

– Sortieren von OPEN $\cup$ Succ(z) nach aufsteigendem g´(z_0 ,z´,G´) + σ(z)

Goto S‘‘1.

Modifikation von Schema S‘ in Schritt 4:

g´(z_0 ,z´,G´) + σ(z) anstelle von g´(z_0 ,z´,G´)

Verfälschung des Ergebnisses durch σ(z) möglich.

Heuristische Suche nach „bestem Weg“

Kosten für Erreichen des Zustandes z´ von z aus:

$g(z,z') := \text{Min}\{ c(s) / s \text{ Weg von z nach z´} \}$ (bzw. ∞)

Kosten für Erreichen eines Zielzustandes von z aus:

$g(z,Z_f) := \text{Min} \{ g(z,z_{final}) \mid z_{final} \in Z_{final} \}$

Schätzfunktion $\sigma(z)$: geschätzter Konstruktions-Aufwand für Erreichen eines Zielzustandes von z aus

Heuristik: Zustände mit optimaler Schätzung bevorzugen

Schätzfunktion σ heisst *optimistisch* oder *Unterschätzung*,

falls $\sigma(z) \leq g(z,Z_f)$ für alle $z \in Z$.

Schema S‘‘ für heurist. Suche nach "bestem Weg“

Satz :

Vor.: Es existiert $\delta > 0$ mit $c(z,z') > \delta$ für alle Kanten in G

σ sei eine optimistische Schätzfunktion

Beh.: Falls Lösung existiert, so findet S‘ einen optimalen Weg

Bemerkung:

Streichen von CLOSED-Zuständen aus OPEN kann bei S‘‘ zu Problemen führen. Benötigen schärfere Bedingungen an σ („konsistente Schätzfunktion“, Algorithmus A*)