

# Einführung in die KI

Prof. Dr. sc. Hans-Dieter Burkhard  
Vorlesung Winter-Semester 2006/07

## 2. Suchverfahren

## 2. Suchverfahren

- 2.1 Allgemeines
- 2.2 Heuristische Suche nach bestem Weg: A\*
- 2.3 Suche in Parameter-Räumen
- 2.4 Problemzerlegung
- 2.5 Suche in Spielbäumen

## 2.1 Allgemeines

Zustandsraumsuche

Komplexität

Expansion

## Suche als Methodik in der Informatik

Suche als Wiederfinden

- Datenbank
- Falldatenbank im fallbasierten Schließen
- Suchmaschine

Suche als Problemlösen

- Existiert eine Lösung?
- Finde eine Lösung.
- Was ist die beste Lösung? (Optimierung)

## Suche als Wiederfinden

### Indexstrukturen

- Suchbäume
- Hashtabellen
- ....

- Datenbanken
- Suchmaschinen
- Textsammlungen
- Knowledge-Management

### Weiterführende Fragen

Suche nach ähnlichen Begriffen

Suche nach Inhalten

Erinnern als Wiederfinden  
Erinnern als Rekonstruktion

## Suche als Problemlösen

### Theoretische Informatik:

- Prinzipielle Lösbarkeit (Berechenbarkeit)
- Komplexitätsabschätzungen

### Praktische Informatik, z.B.

- Compilerbau
- Optimierung
- Theorembeweiser, PROLOG

## Zustandsraum-Suche

Graph  $[Z, Op]$

Zustände  $Z$  (Knoten)

Operatoren  $Op \subseteq Z \times Z$  (Kanten)

Anfangszustand  $z_{initial} \in Z$

Zielzustände  $Z_{final} \subseteq Z$

Kostenfunktion  $c: Z \times Z \rightarrow R^+$

Kosten eines Weges  $w = z_0 z_1 \dots z_n \in Z^*$  :

$$c(z_0 z_1 \dots z_n) := \sum_{i=1, \dots, n} c(z_{i-1}, z_i)$$

Schätzfunktion  $\sigma: Z \rightarrow R$  (Heuristik)

H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

7

## Zustandsraum-Suche

Aufgaben:

- Ist ein Zielzustand  $z \in Z_{final}$  vom Anfangszustand  $z_{initial} \in Z$  erreichbar?
- Finde einen Weg vom Anfangszustand  $z_{initial} \in Z$  zu einem Zielzustand  $z \in Z_{final}$
- Finde einen optimalen Weg vom Anfangszustand  $z_{initial} \in Z$  zu einem Zielzustand  $z \in Z_{final}$

H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

8

## Planung: Modellierung als Graph

Mögliche Aktionen:  $A = \{a_1, \dots, a_n\}$

Zustände (Knoten im Graphen):

$Z$  = durch Aktionen entstehende Situationen

Ausgangssituation: Anfangszustand  $Z_{\text{initial}}$

Situationen, in denen Planungsziel erreicht ist:

Zielzustände  $Z_{\text{final}}$

Zustandsübergänge (Kanten im Graphen):

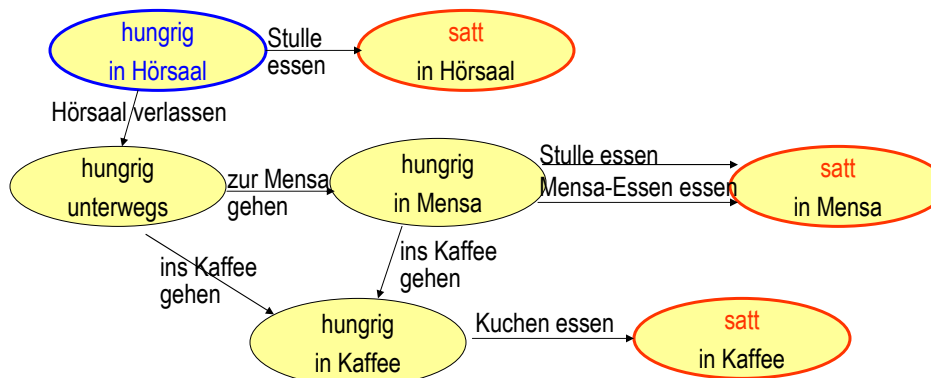
$O_p$  = Übergänge zwischen Situationen durch Aktionen

$G$  ist ein Kanten-beschrifteter Graph mit Mehrfachkanten

## Planung: Modellierung als Graph

Ausgangszustand: *hungrig, im Hörsaal*

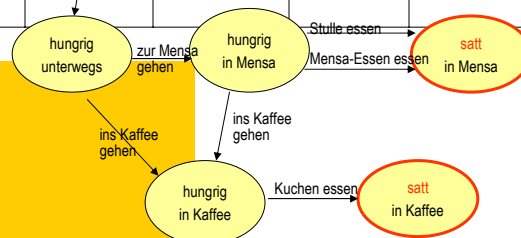
Bedingung an Zielzustände: *satt*



## Übergangsmatrix

	Stulle essen	Hörsaal verlassen	zur Mensa gehen	Mensa-Essen essen	ins Kaffee gehen	Kuchen essen
hungrig, in Hörsaal	s., i.H.	h., u.				
hungrig, unterwegs			h., i.M.			h., i.K.
satt, in Hörsaal						
hungrig, in Mensa	s., i.M.		hungrig in Hörsaal	s., i.M. satt in Hörsaal	h., i.K.	
hungrig, in Kaffee						s., i.K.
satt, in Mensa						
satt, in Kaffee						

- Graph
- Transitionssystem
- Automat
- Akzeptor



11

## Komplexität (Anzahl der Zustände/Knoten)

8-er Puzzle: 9! Zustände

davon  $9!/2 = 181.440$  erreichbar

15-er Puzzle: 16! Zustände

davon  $16!/2$  erreichbar

ungarischer Würfel:  $12 \cdot 4,3 \cdot 10^{19}$  Zustände

1/12 davon erreichbar:  $4,3 \cdot 10^{19}$

Türme von Hanoi:  $3^n$  Zustände für n Scheiben

lösbar in  $(2^n) - 1$  Zügen

Dame: ca  $10^{40}$  Spiele durchschnittlicher Länge

Schach: ca  $10^{120}$  Spiele durchschnittlicher Länge

Go:  $3^{361}$  Stellungen

## Komplexitäts-Probleme

Speicher zu klein für Zustandsraum  
Aufwand für Erkennen von Wiederholungen

Lösungsmethode:

„Expansion des Zustandsraumes“:

Schrittweise Konstruktion und Untersuchung von Zuständen

„konstruieren – testen – vergessen“

## Expansionsstrategien

### – Richtung

- Vorwärts, beginnend mit  $Z_{\text{initial}}$   
(forward chaining, data driven, bottom up)
- Rückwärts, beginnend mit  $Z_{\text{final}}$   
(backward chaining, goal driven, top down)
- Bidirektional

### – Ausdehnung

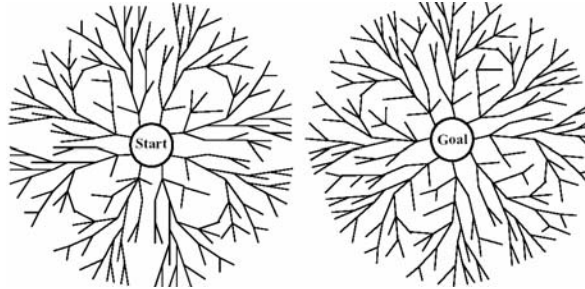
- Tiefe zuerst
- Breite zuerst

### – Zusatzinformation

- blinde Suche („uninformiert“)
- heuristische Suche mit Schätzfunktion  $\sigma$  („informiert“)

## Bidirektionale Breitensuche

Ausgehend von Start und Ziel „parallel“ suchen bis zum Zusammentreffen.



- Aufwand: Von beiden Seiten nur halbe Tiefe

## Expansion

### 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



## 2.2 Zustandsraumsuche

Heuristische Suche nach bestem Weg: A\*  
Weitere Suchverfahren  
Gierige Suche (greedy search)  
Beschneiden des Suchraums (pruning)

## Heuristische Suche nach bestem Weg: A\*

Kosten für Erreichen des Zustandes  $z'$  von  $z$  aus:

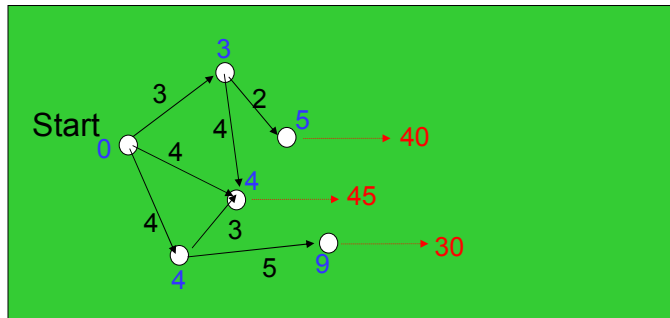
- Falls  $z'$  von  $z$  erreichbar:  
 $g(z, z') := \text{Min}\{ c(s) / s \text{ Weg von } z \text{ nach } z' \},$
- Andernfalls:  $g(z, z') := \infty$

Vorläufigkeit der Kostenberechnung während Expansion:

$G' = [V', E']$  sei (bekannter) Teilgraph von  $G$   
 $g'(z, z', G') := \text{Min} \{ c(s) / s \text{ Weg in } G' \text{ von } z \text{ nach } z' \}$

$$g'(z, z', G') \geq g(z, z')$$

## Heuristische Suche nach bestem Weg



$g'(z_0, z', G')$  : bisher bekannte Kosten zum Erreichen von  $z'$

$\sigma(z')$  : geschätzte Kosten zum Erreichen des Ziels von  $z'$  aus

H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

19

## Algorithmus A\* („weiche Form“)

A\*0: (Start)  $OPEN := [z_0]$ ,  $CLOSED := []$ .

A\*1: (negative Abbruchbedingung)  
Falls  $OPEN = []$ : EXIT(„no“).

A\*2: (positive Abbruchbedingung)  
Sei  $z$  erster Zustand aus  $OPEN$ .  
Falls  $z$  Zielknoten: EXIT(„yes:“  $z$ ).

A\*3: (expandieren)  
 $OPEN := OPEN - \{z\}$ .  $CLOSED := CLOSED \cup \{z\}$ .

$Succ(z)$  := Menge der Nachfolger von  $z$ .

Falls  $Succ(z) = \{\}$ : Goto A\*1.

A\*4: (Organisation von  $OPEN$ )  
–  $OPEN := OPEN \cup Succ(z)$  mit Sortierung nach  
aufsteigendem  $g'(z_0, z', G') + \sigma(z')$

Goto A\*1.

Suchraum wird als  
Baum betrachtet:  
CLOSED wird nicht  
verwendet.

## Algorithmus A\* („weiche Form“)

Definition:  $f(z) := \text{Min} \{ g(z, z_{\text{final}}) \mid z_{\text{final}} \in Z_{\text{final}} \}$   
= tatsächliche minimale Kosten von  $z$  zu Zielzustand  
(  $f(z_0)$  = Kosten des gesuchten optimalen Weges)  
Schätzfunktion  $s$  heisst optimistisch oder Unterschätzung,  
falls  $\sigma(z) \leq f(z)$  für alle  $z \in Z$ .

Satz :

Vor.: Ex.  $\delta > 0$  mit  $c(z, z') > \delta$  für alle  $z, z'$ .

$\sigma$  ist optimistische Schätzfunktion.

Jeder Knoten hat nur endlich viele Nachfolger.

Beh.: Falls Lösung existiert, findet A\* (weiche Form)  
einen optimalen Weg.

## Algorithmus A\* („weiche Form“)

Zum Beweis:

Lemma:

Vor.:  $\sigma$  ist optimistische Schätzfunktion.

Beh.: Für jeden gefundenen Zielzustand ist bei A\* (weiche Form):  
der optimale Weg konstruiert.

Lemma:

Vor.:  $\sigma$  ist optimistische Schätzfunktion.

Beh.: Solange A\* noch nicht gestoppt hat, gilt für jeden optimalen  
Weg  $z_0 \rightarrow z_1 \rightarrow z_2 \rightarrow \dots \rightarrow z_n = z$  zu einem Zielzustand  $z$ :

$$\exists i \in \{1, \dots, n\}: z_i \in \text{OPEN} \wedge g'(z_i, G') + \sigma(z_i) \leq f(z_0)$$

## Güte von Suchalgorithmen

- bzgl. Komplexität des Verfahrens:
  - Zahl der Zustände insgesamt
  - Zahl der erreichbaren Zustände
  - Zahl der untersuchten Zustände
  - Suchtiefe
- bzgl. gefundener Lösungskandidat(en)
  - **Korrektheit:**
    - Antworten sind korrekt
  - **Vollständigkeit:**
    - Algorithmus liefert alle korrekten Antworten  
(schwächer: bei Existenz wird eine Lösung gefunden)
  - **Optimalität:**
    - Algorithmus liefert optimale Lösung(en)

## Spezialfälle

$c \equiv 0$  :

Suche nach irgendeinem Weg mit Heuristik  $\sigma$

$\sigma \equiv 0$  :

Suche nach bestem Weg ohne Heuristik  
(  $\sigma \equiv 0$  ist ebenfalls Unterschätzung )

$c \equiv 1$  ( $g' \equiv$  Suchtiefe) ,  $\sigma \equiv 0$  :

Breitensuche

## Weitere Suchverfahren

Blinde Suche: ohne Heuristik  $\sigma$

Unterschiedliche Expansionsstrategien:

- Tiefe (OPEN als Keller) (siehe Schema S)
- vs. Breite (OPEN als Warteschlange)

Suche nach irgendeinem Weg: ohne Kosten  $c$

	ohne Kosten $c$	mit Kosten $c$
ohne Heuristik $\sigma$	<i>Blinde Suche, irgendein Weg</i>	<i>Bester Weg</i>
mit Heuristik $\sigma$	<i>Heurist. Suche, irgendein Weg</i>	$A^*$

## 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  $OPEN$ )

Bilde neue Liste  $OPEN$  durch Einfügen der Elemente aus  $Succ(z)$ .

Goto S1.

## Blinde Suche nach irgendeinem Weg

Satz:

Für endliche Graphen gilt:

Breite-Zuerst-Suche ist vollständig und korrekt.

Tiefe-Zuerst-Suche ist korrekt.

Speicheraufwand der Liste OPEN:

für  $b$ = Verzweigungszahl(fan-out),  $d$ =Tiefe

- Tiefe-Zuerst: linear  $d \cdot b$
- Breite-Zuerst: exponentiell  $b^d$

(Verzicht auf Speicherung der Liste CLOSED )

## Backtracking

Speicher weiter einschränken:

Statt Liste OPEN nur aktuellen Weg speichern.

Rekursive Variante  
(Prozedurkeller!)

```
success:=false;

function backtrack_search(z);
for all z' in succ(z) do
  if zielknoten(z')
    then success:=true; return;
  backtrack_search(z');
if success then return;
(* sonst: nächstes z' untersuchen *)
```

## 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)“

## Iterative Tiefensuche

DFID bis Tiefe  $d$  bei fan-out  $b$  erfordert insgesamt

$$b^d + 2 \cdot b^{d-1} + 3 \cdot b^{d-2} + \dots + d \cdot b \text{ Schritte}$$

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

$$b^d + b^{d-1} + b^{d-2} + \dots + b \text{ Schritte}$$

DFID hat Speicherbedarf für OPEN wie Tiefe-zuerst  
DFID findet Lösung wie Breite-zuerst

## Einfluss der Heuristik $\sigma$

Trade-off zwischen Güte und Berechnungsaufwand

Kriterium für Expansions-Reihenfolge.

gleiche Reihenfolge wie  $g' + \sigma$  liefern z.B. auch

$a \cdot (g' + \sigma) + b$  für beliebige Konstanten  $a, b$ .

Optimale Reihenfolge bei  $\sigma = f$

$\sigma_2$  *effektiver* als  $\sigma_1$  falls  $\sigma_1 \leq \sigma_2 \leq f$

(Hierarchien für Schätzfunktionen)

## Gierige Suche (greedy search)

Allgemein: die aktuell beste Bewertung bevorzugen

In Suchverfahren:

Expansion bei aktuell besten Werten für  $g'$  und/oder  $\sigma$

Bestensuche: OPEN vollständig sortieren

Bergsteigen:

nur jeweils besten Zustand weiter verfolgen (**local greedy**)

(alternativ: Succ(z) sortiert an Anfang von OPEN)

Probleme bzgl. Vollständigkeit/Korrektheit



## Lokale Suche nach „bestem Zustand“

Bewertung  $v : Z \rightarrow \mathcal{R}$

Gesucht: Zustand mit maximaler Bewertung

(Suchweg uninteressant. Nur Ergebnis, z.B. Stundenplan.)

Nutzensfunktion als „Gebirge“ über dem Zustandsraum betrachten.

(Stetigkeits-)Annahme:

Nachbarn ( $z' \in \text{Succ}(z)$ ) im Zustandsraum haben

zu  $v(z)$  „benachbarte“ Nutzenswerte  $v(z')$

## Lokale Suche nach „bestem Zustand“

### Bergsteigen (Hill Climbing) für lokale Suche nach bestem Zustand

S0: (Start)

$z := z_0$

S1: (expandieren)

$z' \in \text{Succ}(z)$  mit maximalem  $v(z')$  auswählen.

S2: (Abbruchbedingung)

Falls  $v(z') - v(z) < 0$ : EXIT(  $z$  ).

Sonst:  $z := z'$

Goto S1.

## Lokale Suche nach „bestem Zustand“

Verfahren benötigt keinen Speicher für Zustandsmengen

Verfahren liefert lokales Maximum.  
(d.h. oft nicht korrekt und nicht vollständig).

Suboptimale Lösungen akzeptieren?

weitere Versuche mit anderen  $z_0$

## 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

## Typische Probleme lokaler Optimierung

Gegenmaßnahmen:

- Verschlechterungen zulassen.
- Unterschiedliche Ausgangszustände.
- Stochastische Verfahren („ $\epsilon$ -greedy“):
  - Auswahl-Wahrscheinlichkeit  $\sim$  Bewertung  $v(z')$
- Simulated Annealing:
  - Wahrscheinlichkeiten der schlechteren Alternativen nehmen mit der Zeit ab.

## Simulated Annealing

S0: (Start)

$z := z_0$  ,  $T := T_0$  .

S1: (Abbruchbedingungen)

Falls  $T=0$  : EXIT(  $z$  ).

S2: (expandieren)

$T := \tau(T)$

$z'$  zufällig aus  $\text{Succ}(z)$  auswählen.

$\Delta E := v(z') - v(z)$

Falls  $\Delta E > 0$  :  $p := 1$  , andernfalls  $p := \exp(\Delta E/T)$

$z := z'$  mit Wahrscheinlichkeit  $p$

Goto S1.

## Simulated Annealing

$\tau(T)$  ist monoton fallende „Temperaturfunktion“.

Bei geeigneter Temperatur-Funktion (langsames Absinken):  
Vollständig und korrekt (findet Globales Maximum) mit  
Wahrscheinlichkeit nahe 1 .

Analog Bergsteigen für lokale Suche nach bestem Zustand,  
aber:

Wiederholtes Aufsuchen eines Zustandes ermöglicht  
Untersuchung anderer Nachfolger.

## Beschneiden des Suchraums (Pruning)

Beim Expandieren nur Teilmenge von  $\text{succ}(z)$  in OPEN aufnehmen.

- Bergsteigen (je nach Variante)
- Strahlensuche: Die  $k$  am besten bewerteten Zustände werden „parallel“ weiter verfolgt (greedy)
- Wiederholungen vermeiden (Maschen, Zyklen) Zustände aus CLOSED nicht erneut untersuchen)
- Constraints ausnutzen

Verfahren evtl. nicht korrekt/nicht vollständig

## Zyklen, Maschen im Suchraum

Zustände werden mehrmals erreicht und expandiert.

Prolog:

$\text{erreichbar}(X,Y) :- \text{erreichbar}(X,Z), \text{nachbar}(Z,Y).$

$\text{erreichbar}(X,X).$

$\text{symmetrisch}(X,Y) :- \text{symmetrisch}(Y,X).$

Test auf Wiederholungen: Zeit-, Speicher-aufwändig:

Succ(z) mit CLOSED vergleichen (Indizierung verwenden)

Trade-off: Zeit für Wiederholungen vs. Zeit/Speicher für Test

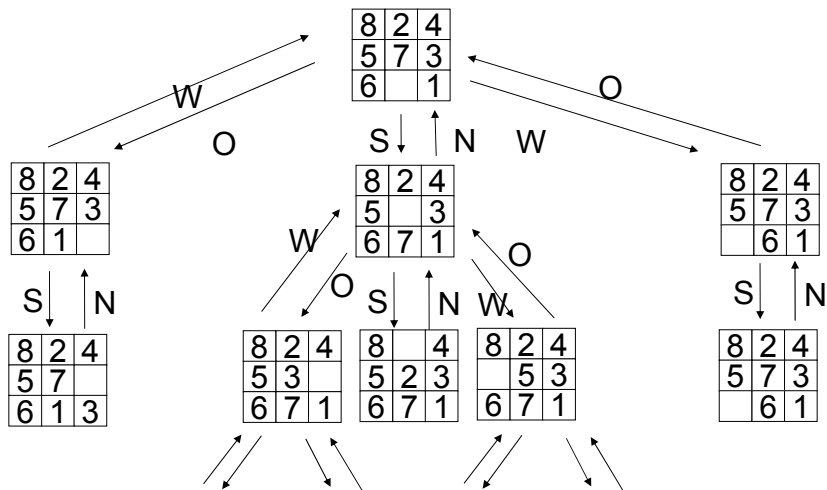
H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

41

## Abwicklung

Ursprünglicher Graph



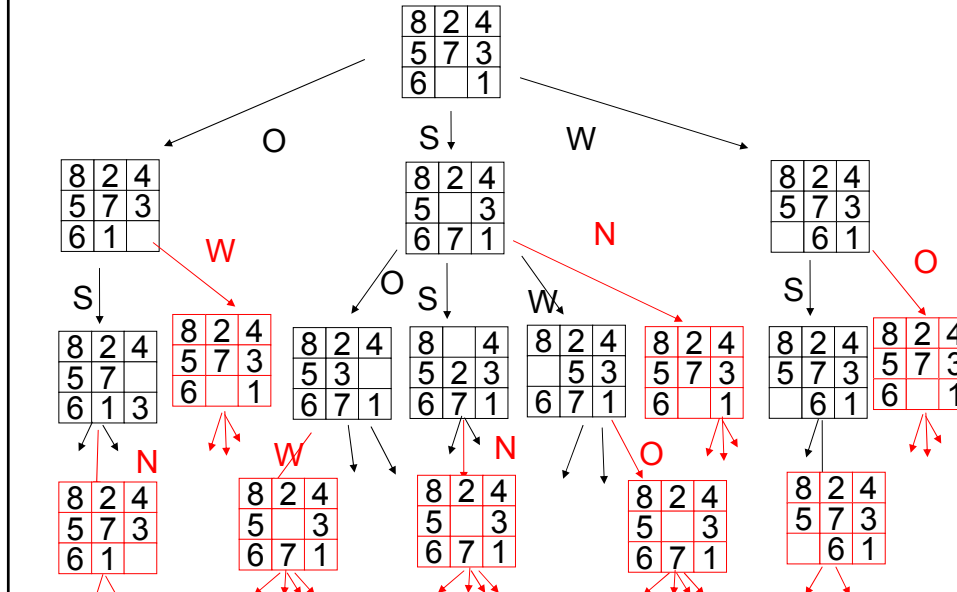
H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

42

## Abwicklung

Durch Abwicklung  
entstandener Baum



## Algorithmus A\* („harte Form“)

A\*0: (Start) **OPEN** :=  $[z_0]$ , **CLOSED** :=  $[\ ]$ .

A\*1: (negative Abbruchbedingung)  
Falls **OPEN** =  $[\ ]$ : EXIT(„no“).

A\*2: (positive Abbruchbedingung)  
Sei **z** erster Zustand aus **OPEN**.  
Falls **z** Zielknoten: EXIT(„yes:“ **z**).

A\*3: (expandieren)  
**OPEN** := **OPEN** -  $\{z\}$ . **CLOSED** := **CLOSED**  $\cup \{z\}$ .  
**Succ(z)** := Menge der Nachfolger von **z**.

Falls **Succ(z)** =  $\{\}$ : Goto A\*1.

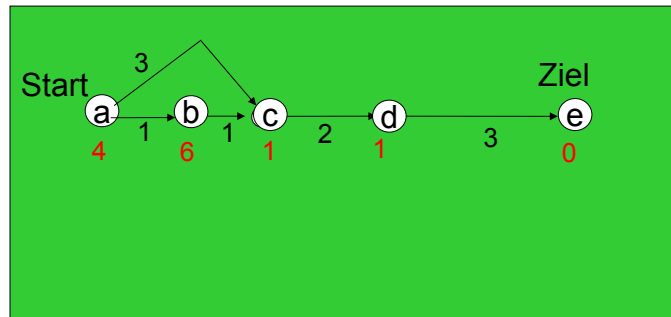
A\*4: (Organisation von **OPEN**)  
**OPEN** := **OPEN**  $\cup$  (**Succ(z)** - **CLOSED**) mit Sortierung  
nach aufstiegender  $g'(z_0, z', G') + \sigma(z')$

Goto A\*1.

## Algorithmus A\* („harte Form“)

Problem:

Für optimistische  $\sigma$  ist die harte Form nicht immer korrekt



H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

45

## Algorithmus A\* („harte Form“)

Definition:

Die Schätzfunktion  $\sigma$  heißt konsistent,  
falls für beliebige Zustände  $z'$ ,  $z''$  gilt:

$$\sigma(z') \leq g(z', z'') + \sigma(z'')$$

Lemma: Wenn  $\sigma$  konsistent ist, so ist  $\sigma$  optimistisch.  
(Umkehrung gilt i.a. nicht)

H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

46

## Algorithmus A\* („harte Form“)

Satz :

Vor.: Ex.  $\delta > 0$  mit  $c(z, z') > \delta$  für alle  $z, z'$  .

$\sigma$  ist konsistente Schätzfunktion

Beh.: Falls Lösung existiert, findet A\* (harte Form)  
einen optimalen Weg.

## Algorithmus A\* („harte Form“)

Zum Beweis:

Lemma:

Vor.:  $\sigma$  ist konsistente Schätzfunktion.

Beh.: Für jedes  $z' \in \text{CLOSED}$  ist bei A\* (harte Form):  
der optimale Weg konstruiert.

Lemma:

Vor.:  $\sigma$  ist konsistente Schätzfunktion.

Beh.: Solange A\* noch nicht gestoppt hat, gilt für jeden optimalen  
Weg  $z_0 \rightarrow z_1 \rightarrow z_2 \rightarrow \dots \rightarrow z_n = z$  zu einem Zielzustand  $z$ :  
 $\exists i \in \{1, \dots, n\}: z_i \in \text{OPEN} \wedge g'(z_i, G') + \sigma(z_i) \leq f(z_0)$



## Algorithmus A\* („harte Form“)

„konsistent“ ist schwerer nachweisbar als „optimistisch“

Benutzung optimistischer Schätzfunktion mit schwächerem Beschneiden des Suchraums:

Nur dann Zustände in OPEN (bzw. Succ(z)) streichen, wenn neue Bewertung schlechter als frühere.

## Algorithmus A\*

Suchkosten ~ Zahl expandierter Zustände,  
~ Berechnungskosten von  $\sigma$  und  $g'$

Exponentieller Platzbedarf

Suchkosten vs. Lösungskosten

(optimale/suboptimale Lösungen)

Maß: „Penetranz“ := Weglänge/expandierte Knoten

## Platzsparende Varianten für Algorithmus A\*

Iterative Deepening A\* ( IDA\* ) analog IDA:

- Tiefe-Zuerst jeweils bis Schranke  $g'(z_0, z', G') + \sigma(z')$  aus vorherigem Iterationsschritt übertroffen wird

SMA\* (simplified memory-bounded A\*)

- wie A\* (mit CLOSED) bis Speichergrenze erreicht.  
Dann :
  - Zustand mit höchstem Wert  $g'(z_0, z', G') + \sigma(z')$  streichen.
  - Update dieses Werts beim Vorgänger in CLOSED.  
(wenn alle Nachfolger gestrichen sind, weiß Vorgänger immer noch, was der beste erreichbare Wert seiner Nachfolger ist).

## 2.3 Suche in Parameter-Räumen

Suche nach optimalen Parametern

Lokale Optimierung/Gradientenverfahren

Evolutionäre Verfahren



## Suche in Parameter-Räumen

Parameter:  $n$  Variable  $x_1, \dots, x_n$  mit Wertebereichen  $W_1, \dots, W_n$

Optimalitätsfunktion  $c(x_1, \dots, x_n)$

Gesucht:

$\mathbf{w} = [w_1, \dots, w_n] \in W_1 \times \dots \times W_n$  mit  $c(\mathbf{w})$  minimal  
(bzw. maximal)

Lösung durch Extremwertbestimmung:

$$\text{grad}(c) = 0$$

## Suche in Parameter-Räumen

Such-Verfahren:

Schrittweises Modifizieren der Parameter

Mit 1 Parametersatz:

- Lokale Optimierung
- Gradienten-Verfahren/Steilster Abstieg (bzw. Anstieg)

Mit mehreren Parametersätzen („Individuen“)

- Evolutionäre/Genetische Algorithmen

## Lokale Optimierung

Vgl. Bergsteigen/Simulated Annealing (s.o.):

Zustände:  $z = [w_1, \dots, w_n]$

Die Nachbarschaft der Zustände ( $\text{succ}(z)$ ) ergibt sich aus der Nachbarschaft der Parameterwerte  $w_i$ .

Bei differenzierbarer Nutzensfunktion:

Gradientenabstieg anwendbar (Suche in Richtung der größten Änderung der Nutzensfunktion)

## Lokale Optimierung

- Vorgebirgsproblem
- Plateau-Problem
- Grat-Problem

## Evolutionäre/Genetische Algorithmen

Idee aus der Natur:

Vermehrung „aussichtsreicher“ Lösungskandidaten

Kombination:

- Kreuzung
- Mutation
- Bewertung:
  - Fitness
- Auslese:
  - gemäß Wahrscheinlichkeit  $\sim$  Fitness

## Bionik

TU Berlin (Ingo Rechenberg)

<http://lautaro.bionik.tu-berlin.de/institut/s2foshow/>



## Evolutionäre/Genetische Algorithmen

Unterschiedliche Bereiche des Suchraums erfassen

Genetische Algorithmen:  
Parameter-Raum =  $\{0,1\}^n$

Evolutionäre Algorithmen:  
Parameter-Raum =  $\mathbb{R}^n$

## Population, Individuum

Population: Menge von Individuen

Individuum: Durch Parametersatz beschrieben.

## Evolutionäre/Genetische Algorithmen

### **Mutation:**

Veränderung von Werten im Individuum  $\mathbf{w} \in \text{Population}$

### **Kombination („cross-over“):**

neues Individuum („Kind“)  $\mathbf{w}'$  aus mehreren Individuen („Eltern“)  $\mathbf{w}_1, \dots, \mathbf{w}_n \in \text{Population}$

**Fitness:** Nähe zu Optimalitätskriterium

**Auswahl:** Wahrscheinlichkeit gemäß Fitness

## Evolutionäre/Genetische Algorithmen

### **Beispiel Kombination („cross-over“):**

Aus  $\mathbf{w}_1 = [w^1_1, \dots, w^1_n]$  und  $\mathbf{w}_2 = [w^2_1, \dots, w^2_n]$

wird  $\mathbf{w}' = [w'_1, \dots, w'_n]$  erzeugt,

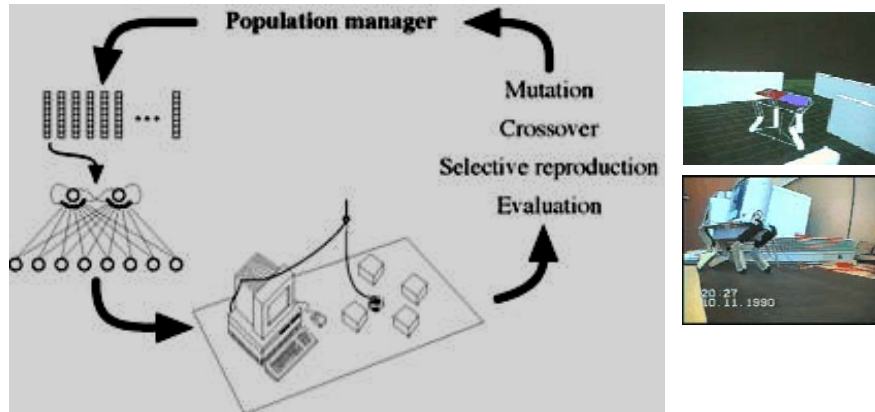
wobei  $w'_i$  mit  $w^1_i$  oder  $w^2_i$  übereinstimmt

alternativ:  $w'_i$  wird aus  $w^1_i$  und  $w^2_i$  berechnet

Statt 2 können k Eltern beteiligt sein (einschließlich k=1).

Mehrere Varianten innerhalb einer Anwendung möglich

## Evolutionäre/Genetische Algorithmen



„Artificial Life“: Golem-Projekt Cornell University

H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

63

## Evolutionäre/Genetische Algorithmen

Grundschemata:

E1: (Start)  $t:=0$ ,  $\text{Population}(0) := \{ \mathbf{w}_1(0), \dots, \mathbf{w}_k(0) \}$   
 $\text{Fitness}(\text{Population}(t)) := \{ \text{Fitness}(\mathbf{w}_1(t)), \dots, \text{Fitness}(\mathbf{w}_k(t)) \}$

E2: (Abbruch)

Falls  $\text{Fitness}(\text{Population}(t))$  „gut“:  $\text{EXIT}(\text{Population}(t))$

E3: (Kombination, Mutation)

$\text{Population}'(t) = \{ \mathbf{w}'_1(t), \dots, \mathbf{w}'_k(t) \}$   
 $:= \text{mutate}(\text{recombine}(\text{Population}(t)))$

E4: (Bewertung)

$\text{Fitness}(\text{Population}'(t)) := \{ \text{Fitness}(\mathbf{w}'_1(t)), \dots, \text{Fitness}(\mathbf{w}'_k(t)) \}$

E5: (Auswahl)

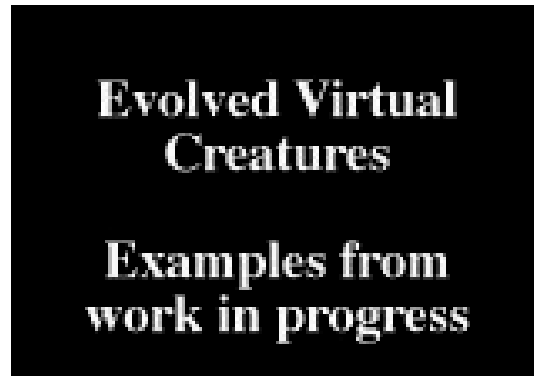
$\text{Population}(t+1) = \{ \mathbf{w}_1(t+1), \dots, \mathbf{w}_k(t+1) \}$   
 $:= \text{select}(\text{Population}'(t), \text{Fitness}(\text{Population}'(t)))$

$t := t+1$  . Goto E2 .



# Evolutionäre/Genetische Algorithmen

Karl Sims -- Virtual Creatures



H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

65

# Evolutionäre/Genetische Algorithmen

Humboldt-Universität:  
RoboCup-Projekt, u.a.  
Omnidirektionales Laufen für AIBO  
Simulierter Roboter: Simloid



H.D.Burkhard, HU Berlin  
Winter-Semester 2006/07

Vorlesung Einführung in die KI  
Suchverfahren

66

## Suche und Umweltmodell

Vollständiges Modell erlaubt offline Berechnungen

- Typisch: Zustandsraumsuche, Gradientenverfahren

Andernfalls:

Lernverfahren, d.h. online-Exploration der Umwelt.

- Evolutionäre Verfahren können auch als Lernverfahren verwendet werden.

## Suche und Sensorinformation

Deterministischer Zustandsraum:

- Auch ohne Sensorinformation kann ein offline berechneter Weg verfolgt werden.

Nichtdeterministischer Zustandsraum (aus Sicht des Agenten)

- mit vollständiger Sensorinformation:
  - Vorher bestimmte Alternativen in Abhängigkeit von Sensorinformation ausführen
- Andernfalls bei beschränkter Sensorinformation:
  - mögliche Umwelt-Zustände als Belief-Zustand des Agenten beschreiben und damit entsprechend arbeiten

## Suchprobleme

Voraussetzung: a-priori-Information über Umwelt

Varianten:

- Zustandsbewertung
- Wegbewertung
- Komplexität
- Struktur des Suchraums
  - fan-in/fan-out, Durchmesser, „zentrale“ Knoten
- Schätzfunktion

Verfahren anpassen