Einführung in die KI

Prof. Dr. sc. Hans-Dieter Burkhard
Vorlesung Winter-Semester 2007/08

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

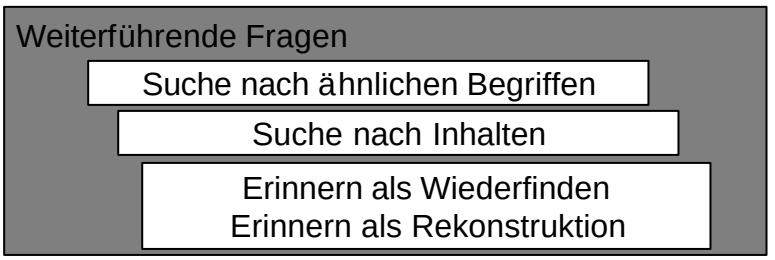
Suche als Problemlösen

- Existiert eine Lösung?
- Finde eine Lösung.
- Finde die beste Lösung. (Optimierung, Lernen)

Suche als Wiederfinden

Indexstrukturen
- – Suchbäume
- – Hashtabellen
- – ....

- Datenbanken
- Suchmaschinen
- Textsammlungen
- Knowledge-Management



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 $w = z_0\ z_1\ ...\ z_n \in Z^*$:

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

Schätzfunktion σ: $Z \rightarrow R$ (Heuristik) für verbleibende Kosten von z zu einem Zielzustand

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 $z \in Z_{final}$

Planung: Modellierung als Graph

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

Zustände (Knoten im Graphen):

Z = durch Aktionen entstehende Situationen

Ausgangsituation: Anfangszustand $z_{initial}$

Situationen, in denen Planungsziel erreicht ist:
Zielzustände Z_{final}

Zustandsübergänge (Kanten im Graphen):

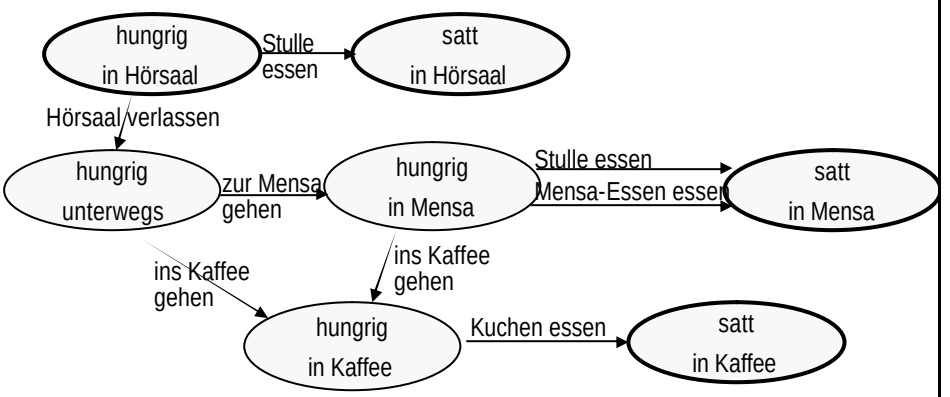
Op = Übergänge zwischen Situationen durch Aktionen

G ist ein Kanten-beschrifteter Graph mit Mehrfachkanten

Planung: Modellierung als Graph

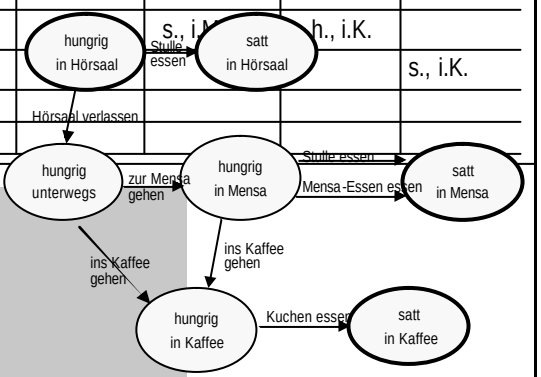
Ausgangszustand: *hungrig, im Hörsaal*

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.			s., i.M.	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

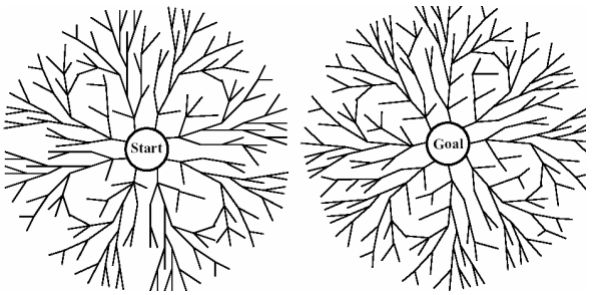
„konstruieren – testen – vergessen“

Expansionsstrategien

- **Richtung**
  - Vorwärts, beginnend mit $z_{initial}$
    (forward chaining, data driven, bottom up)
  - Rückwärts, beginnend mit Z_{final}
    (backward chaining, goal driven, top down)
  - Bidirektional
- **Ausdehnung**
  - Tiefe zuerst
  - Breite zuerst
- **Zusatzinformation**
  - blinde Suche („uninformiert“)
  - heuristische Suche mit Schätzfunktion σ („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: $g(z,z') := ¥$

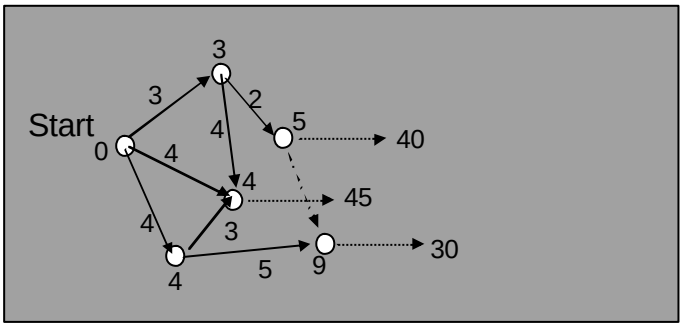
Vorläufigkeit der Kostenberechnung während Expansion:

$G' = [V', E']$ 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

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).

Suchraum wird als Baum betrachtet: CLOSED wird nicht verwendet.

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.

Algorithmus A* („weiche Form“)

Definition: $f(z) := \text{Min} \{ g(z, z_{final}) \mid z_{final} \in Z_{final} \}$
= tatsächliche minimale Kosten von z zu Zielzustand
($f(z_0)$ = Kosten des gesuchten optimalen Weges)
Schätzfunktion σ 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' .
σ 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.: σ ist optimistische Schätzfunktion.
Beh.: Wenn A* (weiche Form) mit einem Zielzustand z stoppt (Schritt2)
so wurde der optimale Weg für z konstruiert.

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

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 = 0 :
Suche nach irgendeinem Weg mit Heuristik σ

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

c = 1 (g´ = Suchtiefe) , $\sigma = 0$:
Breitensuche

Weitere Suchverfahren

Blinde Suche: ohne Heuristik σ

Unterschiedliche Expansionsstrageien:

- Tiefe (OPEN als Keller)
- vs. Breite (OPEN als Warteschlange)

(siehe Schema S)

Suche nach irgendeinem Weg: ohne Kosten c

	ohne Kosten c	mit Kosten c
ohne Heuristik σ	*Blinde Suche, irgendein Weg*	*Bester Weg*
mit Heuristik σ	*Heurist. Suche, irgendein Weg*	*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 ∪{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 für Tiefe-Zuerst

(Liste OPEN im 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 *)
```

Backtracking

Speicher weiter einschränken:
Statt Liste OPEN nur aktuellen Weg speichern.

heuristische rekursive Varianten:

Reihenfolge, Beschränkte Auswahl

```
success:=false;

function backtrack_search(z);
for best 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

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

Einfluss der Heuristik σ

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 $\sigma = f$
σ_2 *effektiver* als σ_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 σ

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 $v : Z \rightarrow \Re$
Gesucht: Zustand mit maximaler Bewertung
(Suchweg uninteressant. Nur Ergebnis, z.B. Stundenplan.)

Nutzensfunktion als „Gebirge“ über dem Zustandsraum betrachten.

(Stetigkeits-)Annahme:
Nachbarn ($z' \in Succ(z)$) 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 := z0
S1: (expandieren)
z‘∈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
  - Auswahl von z‘ mit Wahrscheinlichkeit ~ v(z‘)
- Simulated Annealing:
  - Wahrscheinlichkeiten der schlechteren Alternativen nehmen mit der Zeit ab.

Simulated Annealing

S0: (Start)
z := z_0 , T:= T_0 („Temperatur“).
S1: (Abbruchbedingung)
Falls T=0 : EXIT(z).
S2: (expandieren)
T := τ(T)
z‘ zufällig aus Succ(z) auswählen.
ΔE := v(z‘) - v(z)
Falls ΔE > 0 : p:= 1 , andernfalls p:= exp(ΔE/T)
z:= z‘ mit Wahrscheinlichkeit p
Goto S1.

Simulated Annealing

τ(T) ist monoton fallende „Temperaturfunktion“.

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

Analog zu 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 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:

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

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

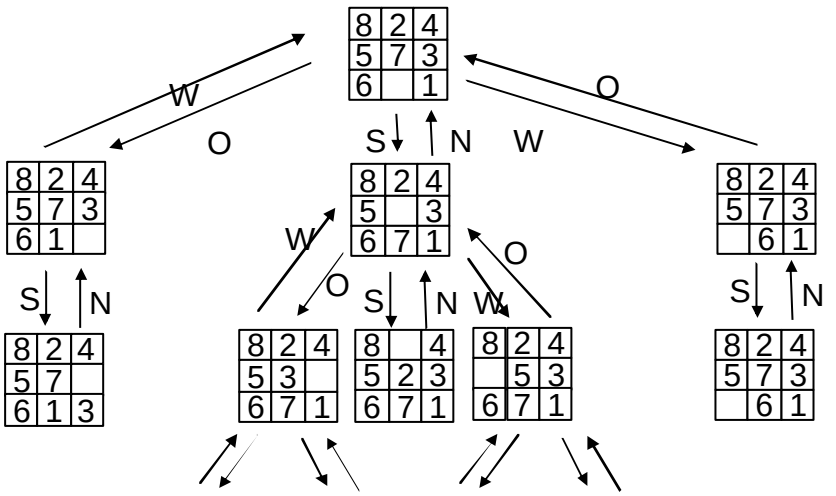
Test auf Wiederholungen: Zeit-, Speicher-aufwändig:

Succ(z) mit CLOSED vergleichen (Indizierung verwenden)

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

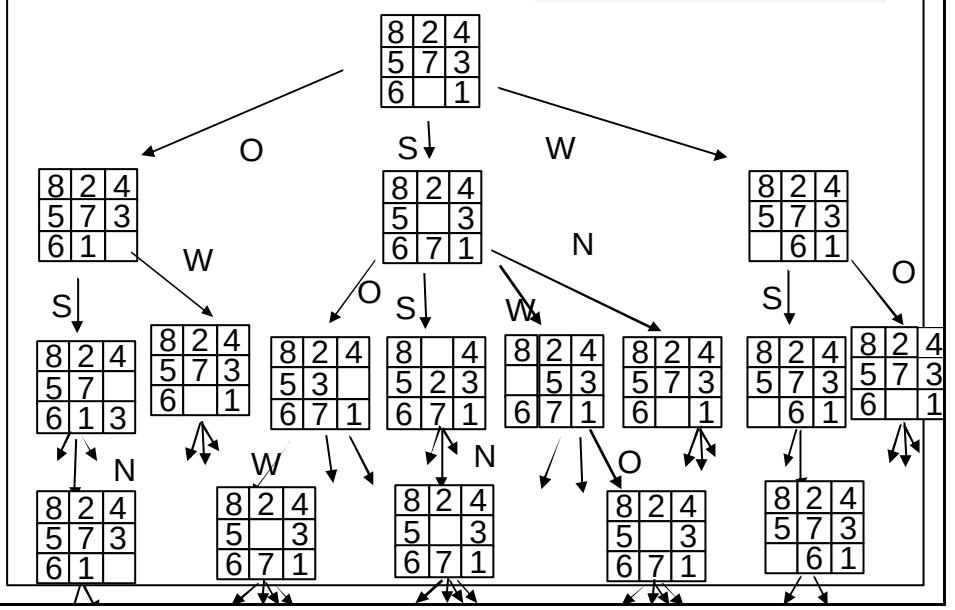
Abwicklung

Ursprünglicher Graph



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)

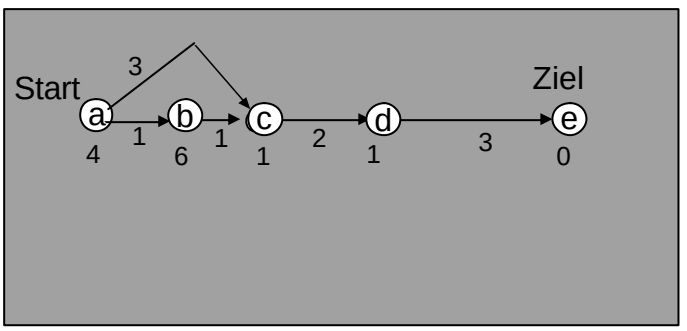
OPEN := OPEN $\cup$ (Succ(z) - CLOSED) mit Sortierung nach aufsteigendem $g'(z_0, z', G') + \sigma(z')$

Goto A*1.

Algorithmus A* („harte Form“)

Problem:

Für optimistische σ ist die harte Form nicht immer korrekt



Algorithmus A* („harte Form“)

Definition:

Die Schätzfunktion σ heißt *konsistent*,
falls für beliebige Zustände z', z'' gilt:

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

Lemma: Wenn σ konsistent ist, so ist σ optimistisch.

(Umkehrung gilt i.a. nicht)

Algorithmus A* („harte Form“)

Satz :

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

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

Algorithmus A* („harte Form“)

Zum Beweis:

Lemma:

Vor.: σ ist konsistente Schätzfunktion.

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

Lemma:

Vor.: σ ist konsistente Schätzfunktion.

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

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

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 σ 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.

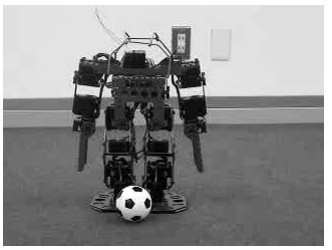
Dann :

- 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 $x_1,...,x_n$ mit Wertebereichen $W_1,...,W_n$

Optimalitätsfunktion $c(x_1,...,x_n)$

Gesucht:
$w = [w_1,...,w_n] \in W_1 \times ... \times W_n$ mit $c(w)$ minimal
(bzw. maximal)

Lösung durch Extremwertbestimmung:
$grad(c) = 0$

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: $z = [w_1,...,w_n]$

Die Nachbarschaft der Zustände (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

Gradienten-Verfahren/Steilster Abstieg

Start: z_0 wählen

Iteration: $z_{t+1} := z_t + \alpha \ \text{grad}(c(z_t))$

mit Schrittlängen $\alpha > 0$

„Simulated Annealing“ bzgl. Schrittlängen:

Schrittlänge α als monoton fallende Funktion der Zeit

Evolutionäre/Genetische Algorithmen

Idee aus der Natur:

Vermehrung „aussichtsreicher“ Lösungskandidaten
Kombination:

- Kreuzung
- Mutation

– Bewertung:

- Fitness

– Auslese:

- gemäß Wahrscheinlichkeit ~ Fitness

Bionik

TU Berlin (Ingo Rechenberg)
http://lautaro.bionik.tu-berlin.de/institut/s2foshow/

Evolutionäre/Genetische Algorithmen

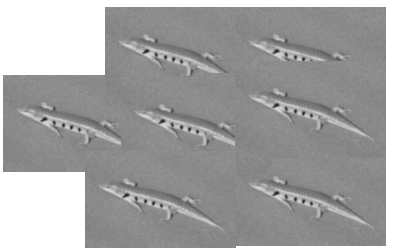
Unterschiedliche Bereiche des Suchraums erfassen

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

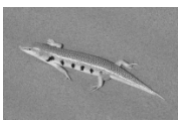
Evolutionäre Algorithmen:
Parameter-Raum = $\mathbf{R}^n$

Population, Individuum

Population: Menge von Individuen



Individuum: Durch Parametersatz beschrieben.



Evolutionäre/Genetische Algorithmen

Mutation:

Veränderung von Werten im Individuum $w \in$ Population

Kombination („cross-over“):

neues Individuum („Kind“) w' aus mehreren Individuen („Eltern“) $w_1,\ldots,w_n \in$ Population

Fitness: Nähe zu Optimalitätskriterium

Auswahl: Wahrscheinlichkeit gemäß Fitness

Evolutionäre/Genetische Algorithmen

Beispiel Kombination („cross-over“):

Aus $w_1 = [w^1_1,\ldots,w^1_n]$ und $w_2 = [w^2_1,\ldots,w^2_n]$
wird $w' = [w'_1,\ldots,w'_n]$ erzeugt,

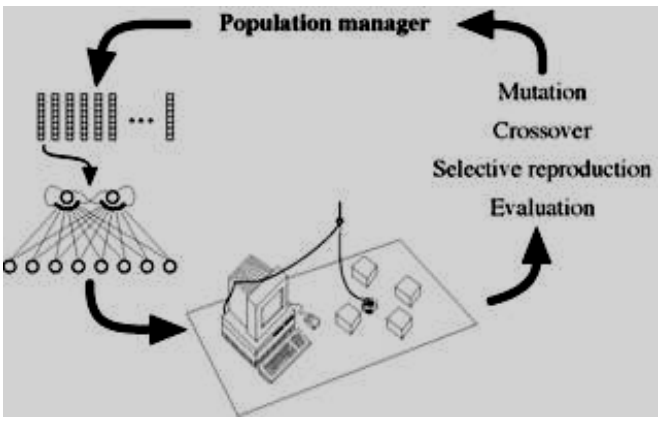
wobei w'_i mit w^1_i oder w^2_i übereinstimmt

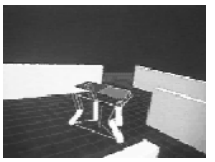
alternativ: w'_i wird aus w^1_i und w^2_i berechnet

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

Evolutionäre/Genetische Algorithmen

Grundschema:

E1: (Start) t:=0, Population(0) := { $w_1(0)$,.., $w_k(0)$ }
Fitness(Population(t)) := { Fitness($w_1(t)$) ,.., Fitness($w_k(t)$) }

E2: (Abbruch)
Falls Fitness(Population(t)) „gut“: EXIT(Population(t))

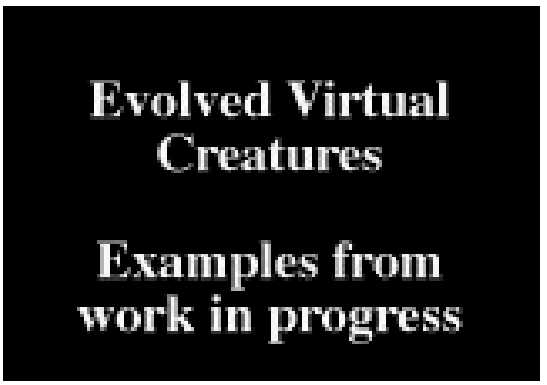
E3: (Kombination, Mutation)
Population‘(t) = {$w'_1(t)$,.., $w'_{k'}(t)$ }
:= mutate (recombine (Population(t))

E4: (Bewertung)
Fitness(Population‘(t)) := { Fitness($w'_1(t)$) ,.., Fitness($w'_{k'}(t)$) }

E5: (Auswahl)
Population(t+1) = {$w_1(t+1)$,.., $w_k(t+1)$ }
:= select(Population‘(t), Fitness(Population‘(t)))
t := t+1 . Goto E2 .

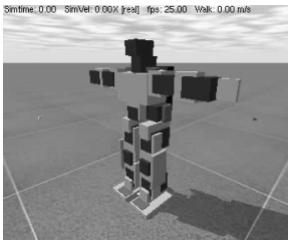
Evolutionäre/Genetische Algorithmen

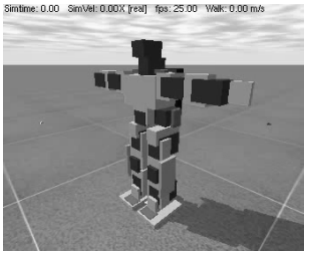
Karl Sims -- Virtual Creatures

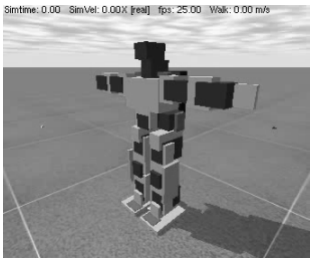


Evolutionäre/Genetische Algorithmen

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







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