

Künstliche Intelligenz

Teil II: Schwache Problemlösemethoden¹

Robert Jäschke
Gerhard Gossen

FG Wissensbasierte Systeme/Forschungszentrum L3S
Leibniz Universität Hannover

Sommersemester 2015

¹Dieser Foliensatz basiert auf Material von Mirjam Minor,
Humboldt-Universität Berlin, WS 2000/01

2 Suche

- Einführung
- Blinde Suche
- Heuristische Suche
- Problemzerlegung
- Suche in Spielbäumen

2 Suche

- **Einführung**
- Blinde Suche
- Heuristische Suche
- Problemzerlegung
- Suche in Spielbäumen

- Auswahl: Rucksackproblem
- Planung: z.B. Klötzchen-Welt, Routen, TSP, Agenten, Roboter
- Problemlöser: Beweise/Ableitungen – Logik/Prolog; VLSI Layout
- intelligente Spielerprogramme – Gewinnstrategien
- Informationsbeschaffung/Recherche – Diagnose

Mögliche Ziele einer Suche

- finde eine Lösung
- finde alle Lösungen
- finde die kürzeste Lösung
- finde eine optimale Lösung

In allen genannten Fällen kann zudem der Lösungsweg von Interesse sein.

Bevor ein Suchverfahren eingesetzt werden kann, muss das Suchproblem zunächst auf adäquate Weise beschrieben werden. Eine Möglichkeit ist die *Zustandsraumrepräsentation*. Sie enthält:

Zustände: Beschreibungen der verschiedenen möglichen Situationen

Zustandsübergänge: Operatoren die einen Zustand in einen anderen überführen

Es gibt zwei Mengen ausgewählter Zustände: *Startzustände* und *Zielzustände*. Die Zielzustände können implizit (durch Kriterium) oder explizit (durch Aufzählung) gegeben sein.

Die Angabe einer expliziten Zielmenge kann evtl. genauso aufwendig sein wie die Suche selbst!

Interpretation des Zustandsraums als Graph/Baum

Beispiele für Zustandsraumbeschreibungen

Beweis eines Theorems

Zustände: Faktenmenge (Lemmata, Zwischenresultate)

Operatoren: Inferenzregeln (Faktenmenge wird erweitert)

Anfangszustand: Axiome

Zielzustände: Faktenmengen, die das Theorem enthalten

Rubik's Cube („Zauber-Würfel“)

Zustände: $8! \times 12! \times 3^8 \times 2^{12}$ Stellungen

Operatoren: 6×3 Drehungen

Anfangszustand: Anfangsstellung

Zielzustände: geordnete Stellung



Beispiele für Zustandsraumbeschreibungen

Prolog

Zustände: Datenbasis + Abarbeitungszustand (offene Subgoals; alternative Klauseln)

Operatoren: Übergang zu einem Subgoal

Anfangszustand: initiales Programm + Anfrage

Zielzustände: Programmziel

Schiebepuzzle (8-er Puzzle/15-er Puzzle)

Zustände: Positionen aller Steine

Operatoren: mögliche Bewegungen des leeren Feldes

Anfangszustand: aktuelle Stellung

Zielzustände: geordnete Stellung



Beispiele für Zustandsraumbeschreibungen

Prolog

Zustände: Datenbasis + Abarbeitungszustand (offene Subgoals; alternative Klauseln)

Operatoren: Übergang zu einem Subgoal

Anfangszustand: initiales Programm + Anfrage

Zielzustände: Programmziel

Schiebepuzzle (8-er Puzzle/15-er Puzzle)

Zustände: Positionen aller Steine

Operatoren: mögliche Bewegungen des leeren Feldes

Anfangszustand: aktuelle Stellung

Zielzustände: geordnete Stellung



Warum bewegen wir das Leerfeld und nicht die anderen Teile?

Das wäre komplexer zu modellieren und es wäre mehr zu überprüfen.

Fazit: Die Modellierung beeinflusst, wie kompliziert die Suche wird.

Generate and Test?

= bei den Anfangszuständen beginnend sukzessive Nachfolgezustände generieren und auf Zielbedingung testen.

Problem: kombinatorische Explosion

8-er Puzzle: $9!$ Zustände (davon $9!/2$ erreichbar)
bei durchschnittlich 3 Nachfolgern in jedem Zustand und
einer typischen Lösungsweglänge von 20: $3^{20} \approx 3,5$ Mrd.
Knoten

Rubik's Cube: rund $4,3 \times 10^{19}$ erreichbare Zustände
kürzeste Lösungsfolge max. 20 Züge (Rokicki, Davidson,
Dethridge und Kociemba, 2010)

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

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

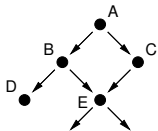
Go: $\leq 3^{361}$ Stellungen

Fazit: Heuristik wird benötigt.

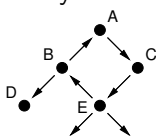
Suchen in Graphen und Bäumen

Probleme:

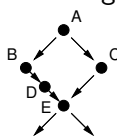
Schlaufen



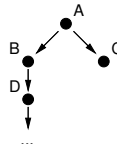
Zyklen



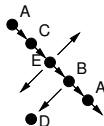
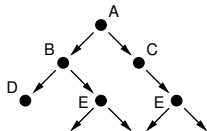
Abkürzungen



unendliche Pfade



Überführung eines Graphen in einen Baum durch Abwicklung:



Schema S zur Suche nach irgendeinem Weg

- S0** : Sei z_0 Anfangszustand. Falls dieser bereits Zielzustand ist: EXIT(yes)
OPEN := $\{z_0\}$, CLOSED := $\{ \}$
- S1** : Falls OPEN = $\{ \}$: EXIT(no)
- S2** : Sei z der erste Zustand in OPEN
OPEN := OPEN $\setminus \{z\}$, CLOSED := CLOSED $\cup \{z\}$
Bilde Nachfolgermenge SUCC(z); Falls SUCC(z) = $\{ \}$: GOTO S1
- S3** : Falls SUCC(z) einen Zielzustand enthält: EXIT(yes)
- S4** : Reduziere SUCC(z) auf NEW(z) durch Streichen nicht weiter zu betrachtender Elemente; Füge NEW(z) in OPEN ein, GOTO S1

Schema S zur Suche nach irgendeinem Weg (Forts.)

- OPEN enthält generierte, noch nicht expandierte Knoten
- CLOSED enthält die expandierten Knoten

S0: Start

S1: negative Abbruchbedingung

S2: Expandieren

S3: positive Abbruchbedingung

S4: Organisation von OPEN

Die Organisation von OPEN ist noch nicht spezifiziert, deswegen „Schema“.

- Expansionsrichtung: vorwärts, rückwärts, bidirektional
- Verwendung von Zusatzinformationen: blinde/heuristische Suche
- Entfernung der Knoten vom Start beachten
- Verwendung aller/einiger Nachfolger
- Modifikationen bzgl. der Elemente in OPEN und CLOSED

- Problem tritt bei Baum als Suchraum nicht auf.
- Wiederholte Zustände sind u.a. unvermeidbar, wenn Aktionen umkehrbar sind (bspw. Routensuche oder Schiebepuzzle). Der Suchbaum ist dann unendlich, kann aber durch Stutzen auf endliche Größe gebracht werden.
- „Algorithmen, die ihren Verlauf vergessen, neigen dazu, ihn zu wiederholen.“ D.h., man muss zwischen Speicher- und Zeitaufwand abwägen.

- Vermeidung wird in Schema S durch Ignorieren der Knoten in CLOSED erreicht.
- Wenn nach einem kürzesten Pfad gesucht wird, muss i.A. für Knoten in CLOSED, die erneut erreicht werden, überprüft werden, ob der neue Weg kürzer ist als der alte (siehe schwacher A*-Algorithmus).

Bewertungskriterien für die Qualität eines Suchverfahrens

Vollständigkeit: Wird eine existierende Lösung immer gefunden?

Optimalität: Wird die beste Lösung gefunden?

Platzbedarf: Wieviele Knoten müssen gleichzeitig gespeichert werden?

Zeitbedarf: Wieviele Knoten müssen expandiert werden? (im besten Fall, im schlimmsten Fall, im Durchschnitt)

Die vorgestellten Suchverfahren setzen voraus, dass das Problem *statisch*, *beobachtbar*, *diskret* und *deterministisch* ist.

2 Suche

- Einführung
- **Blinde Suche**
- Heuristische Suche
- Problemzerlegung
- Suche in Spielbäumen

Breitensuche (BFS)

- in S4 alle Nachfolger ans *Ende* von OPEN stellen
- Platzbedarf b^d ; Zeitbedarf $O(b^d)$; vollständig; Lösung mit min. Anzahl von Zustandsübergängen wird zuerst gefunden

Tiefensuche (DFS)

- in S4 alle Nachfolger an den *Anfang* von OPEN stellen
- Platzbedarf $b \times m = O(m)$; Zeitbedarf $O(b^d)$; nicht optimal, bei unendlichen Pfaden im Suchraum nicht vollständig;

b – Verzweigungsgrad

d – Tiefe der Lösung

m – maximale Tiefe des Suchbaumes

Iterative Tiefensuche (IDS)

- In S4 nur die Nachfolger am Anfang einfügen, wenn maximale Suchtiefe c noch nicht überschritten.
- c sukzessive erhöhen.
- Platzbedarf $b \times d$; Zeitbedarf $O(b^d)$; vollständig; optimal
- Zeitverhalten in Relation zur Tiefensuche:
$$1 \leq \frac{\text{Zeitbedarf}(\text{IDS})}{\text{Zeitbedarf}(\text{DFS})} = \frac{b+1}{b-1} \leq 3$$

Iterative Tiefensuche (IDS)

- In S4 nur die Nachfolger am Anfang einfügen, wenn maximale Suchtiefe c noch nicht überschritten.

- c sukzessive erhöhen.

- Platzbedarf $b \times d$; Zeitbedarf $O(b^d)$; vollständig; optimal

- Zeitverhalten in Relation zur Tiefensuche:

$$1 \leq \frac{\text{Zeitbedarf}(\text{IDS})}{\text{Zeitbedarf}(\text{DFS})} = \frac{b+1}{b-1} \leq 3$$

-
- Knoten in niedrigen Suchtiefen werden wiederholt erzeugt.
 - Hiervon gibt es (bei ungefähr gleichbleibendem Verzweigungsgrad) jedoch relativ wenige.
 - IDS ist bevorzugte uninformierte Such-Methode bei großem Suchraum, wenn die Tiefe der Lösung nicht bekannt ist.

Bidirektionale Suche

- Suche gleichzeitig von Start- und Endzustand; Stop, wenn beide Suchprozesse sich treffen
- Motivation: Der besuchte Bereich wird sehr viel kleiner, da $b^{\frac{d}{2}} + b^{\frac{d}{2}} \ll b^d$
- Abgleich ob Überlappung gefunden mit Hash-Tabelle in konstanter Zeit.

Bidirektionale Suche

- Suche gleichzeitig von Start- und Endzustand; Stop, wenn beide Suchprozesse sich treffen
 - Motivation: Der besuchte Bereich wird sehr viel kleiner, da $b^{\frac{d}{2}} + b^{\frac{d}{2}} \ll b^d$
 - Abgleich ob Überlappung gefunden mit Hash-Tabelle in konstanter Zeit.
-
- Mindestens einer der beiden Suchbäume muss im Speicher gehalten werden, d.h. Speicherbedarf $O(b^{\frac{d}{2}})$. Größte Schwäche dieses Ansatzes.
 - Die Rückwärtssuche ist nur effizient durchführbar, wenn die Vorgänger eines Knotens effizient berechnet werden können.
 - Ansatz ist schwierig, wenn die Zielzustände nur implizit gegeben sind. Bsp.: Schach-Matt.

- Jede Zustandsüberführung verursacht Kosten.
Verlängert sich der Weg, erhöhen sich die Kosten.
Die Kosten sind nicht negativ und können nicht beliebig klein werden.
- Jedem Operator werden Kosten $c(n \rightarrow n') \geq \varepsilon > 0$ zugeordnet.
- Kosten eines Pfades:

$$g(n_0 n_1 \dots n_k) = \sum_{i=0}^{k-1} c(n_i \rightarrow n_{i+1})$$

- Kosten zum Erreichen von z :

$$g^*(z) = \min\{g(s) \mid s \text{ ist Weg von } z_0 \text{ nach } z\}$$

Gleiche-Kosten-Suche (UCS – Uniform Cost Search)

- In S4 alle Nachfolger einfügen und OPEN nach Kosten zum Erreichen der Zustände (Wegkosten) sortieren.
- Ermittelte Kosten $g(n)$ stellen eine obere Schranke für die tatsächlichen Kosten $g^*(n)$ dar: $g^*(n) \leq g(n)$
- Platzbedarf $O(b^d)$; Zeitbedarf $O(b^d)$; vollständig; optimal
- BFS ist ein Spezialfall von UCS ist ein Spezialfall von A*

Gleiche-Kosten-Suche (UCS – Uniform Cost Search)

- In S4 alle Nachfolger einfügen und OPEN nach Kosten zum Erreichen der Zustände (Wegkosten) sortieren.
- Ermittelte Kosten $g(n)$ stellen eine obere Schranke für die tatsächlichen Kosten $g^*(n)$ dar: $g^*(n) \leq g(n)$
- Platzbedarf $O(b^d)$; Zeitbedarf $O(b^d)$; vollständig; optimal
- BFS ist ein Spezialfall von UCS ist ein Spezialfall von A*

Fragen:

- Welches Problem gibt es, wenn keine untere Schranke ε existiert?
- Wieso ist Breitensuche ein Spezialfall von Gleiche-Kosten-Suche?

- Alle vorgestellten blinden Suchverfahren haben Zeitverhalten von $O(b^d)$.
- Hopcroft/Ullman 1979 „Introduction to Automata Theory“: das gilt für jedes blinde Suchverhalten.
- Verzweigungsgrad b und Tiefe der Lösung d werden empirisch abgeschätzt, sie sind domänenabhängig.

2 Suche

- Einführung
- Blinde Suche
- **Heuristische Suche**
- Problemzerlegung
- Suche in Spielbäumen

Verwendung einer heuristischen Schätzfunktion (Heuristik)

- Um ein besseres Zeitverhalten zu erreichen, soll Wissen über das konkrete Suchproblem genutzt werden. Zustände, die nahe am Ziel liegen, werden bei der Expansion bevorzugt.
- Ideal: Funktion $h^*(n)$, die die *tatsächlichen Kosten* des kürzesten Weges von einem Zustand n zu einem Zielzustand angibt. Die Ermittlung von h^* ist leider oft zu aufwendig.
- $h(n)$ liefert eine *Abschätzung der Kosten* für den kürzesten Pfad zum Ziel.
- Die Heuristik ist jeweils problemspezifisch.

- die Heuristik soll stets nicht-negative Werte liefern: $h(n) \geq 0$
- Falls n Zielzustand ist, soll $h(n) = 0$ gelten.
- Bei tatsächlicher Annäherung an einen Zielzustand wird kleinere Distanz signalisiert.
- h soll mit möglichst geringem Aufwand zu errechnen sein.

Seien h_1 und h_2 Heuristiken. Dann heißt h_2 *besser informiert* als h_1 , wenn gilt: $\forall n : h_1(n) \leq h_2(n)$

- die am schlechtesten informierte Heuristik: $\forall n : h(n) = 0$
- für Routenplanung: Luftlinie
- für Schiebepuzzle: Anzahl der falsch liegenden Steine oder Manhattan-Distanz (Summe der horizontalen und vertikalen Entfernungen aller Steine von ihren Zielpositionen)

Frage: Welche Informationen über das Suchproblem werden jeweils verwendet?

Heuristische Suche mit schrittweiser lokaler Verbesserung

Bergsteigen (Hill Climbing with Backtracking – BTHC)

- In S_4 alle Elemente aus $SUCC(z)$ gemäß ihrer Bewertung durch h ordnen und an den Anfang von OPEN stellen.
- nicht vollständig in unendlichen Räumen

optimistisches Bergsteigen (Strict Hill Climbing – SHC)

- In S_4 nur das am besten bewertete Element aus $SUCC(z)$ an den Anfang von OPEN stellen.
- nicht vollständig

Strahlensuche (Beam Search – BS)

- In S_4 die m am besten bewerteten Elemente aus $SUCC(z)$ gemäß ihrer Bewertung ordnen und an den Anfang von OPEN stellen
- nicht vollständig

- Der Suchraum wird als Landschaft interpretiert, h gibt Höhe an.
- Man läuft nach unten (Abstieg gehört ja auch zum Bergsteigen): Finden (lokaler) Minima.
- diskrete Version eines Gradienten-Abstiegs (je nachdem ob man die Heuristik als Kosten- oder Qualitätsfunktion ansieht); Optimierung

- SHC „abwärts“ ist analog zu der Pfadfinderregel: Wenn man sich verlaufen hat, immer bergab laufen. Irgendwann findet man Tal oder Gewässer (= bevorzugte Siedlungsplätze).
- Raum-/Zeitbedarf sind abhängig von der Heuristik
- Übung: Expansion der Suchbäume bei verschiedenen Verfahren zeigen

Vorteile:

- geringer Speicherbedarf (häufig konstant),
- finden Lösungen auch in unendlichen Suchräumen (z.B. bei stetigen Problemen).

Nachteile:

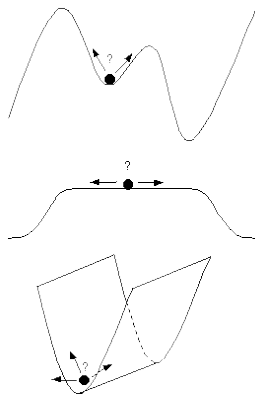
- Sie merken sich nicht den Pfad zur Lösung, sind also nur sinnvoll einzusetzen, wenn es allein auf die Endkonfiguration ankommt (z.B. Damenproblem).
- Keine Nutzung globaler Merkmale der Heuristik (siehe nächste Folie).

Probleme beim Bergsteigen

lokale Minima – vorübergehend keine Verbesserung möglich

Plateaus – keine Unterschiede in der Bewertung

enge Talsohlen – vorhandene Operatoren erfassen den Abstieg nicht



Um die genannten Probleme zu umgehen, werden die Verfahren modifiziert.

Randomisiertes Verfahren (Random Restart Hill Climbing – RRHC)

- Wenn kein Fortschritt mehr erzielt werden kann, wird das Verfahren in einem zufällig gewählten Zustand neu gestartet. Der Zustand mit der bislang besten Bewertung wird gespeichert.
- Globales Minimum wird gefunden, wenn genug Neustarts. Anzahl der notwendigen Neustarts abhängig von Anzahl der lokalen Minima.

Simulated Annealing (Simuliertes Abkühlen)

- Modelliert das Erstarrungsverhalten von Flüssigkeiten. Erfahrung vom Stahlkochen: erst stark erhitzen, dann langsam abkühlen ergibt guten Stahl
- Mit bestimmter Wahrscheinlichkeit dürfen auch Schritte in Richtungen ausgeführt werden, die nicht zu einer weiteren Minimierung führen. Mit zunehmender Dauer des Verfahrens nimmt diese Wahrscheinlichkeit ab.
- Auswahl: Folgezustand zufällig wählen. Wenn er Abstieg bedeutet, wird er auf alle Fälle gewählt, sonst nur mit vorgegebener Wahrscheinlichkeit.

Weiterer Ansatz: bisherige Richtung zu einem gewissen Teil beibehalten (kontinuierlicher Raum); Schrittweite verändern

Greedy Search (GS)

- Idee: Um schneller zum Ziel zu kommen und lokale Minima zu vermeiden, werden alle Zustände in OPEN zur Bewertung herangezogen.
- Gierig (greedy), weil ohne Rücksicht auf die Kosten jeweils mit dem vielversprechendsten Zustand weitergemacht wird
- In S4 alle Elemente aus $SUCC(z)$ in OPEN übernehmen und die gesamte Liste OPEN gemäß der Bewertung durch h sortieren.
- Unvollständig; nicht optimal; Zeit- und Raumbedarf $O(b^m)$; kann durch gute Heuristik deutlich verbessert werden.

- Eine Heuristik h heißt *fair*, falls es zu einem beliebigen $k \geq 0$ nur endlich viele Zustände n gibt mit $h(n) \leq k$.
- eine Heuristik h heißt *zulässig* oder *optimistisch*, falls $\forall n : h(n) \leq h^*(n)$.
- eine Heuristik h heißt *konsistent*, falls für alle n gilt:
 $h(n) \leq c(n \rightarrow n') + h(n')$.
Diese Eigenschaft wird auch als *Monotonie-Beschränkung* bezeichnet.

- Fairness + Zyklenvermeidung durch Abgleich mit CLOSE → Greedy wird vollständig
- bei endlichen Suchräumen ist Fairness automatisch erfüllt
- optimistisch: keine Überschätzung der Kosten
- Konsistenz: geschätzte Distanz zum Ziel schrumpft langsamer, als die Kosten durch diesen Schritt wachsen.
- Heuristiken „Anzahl falsche Steine“ und „Manhattandistanz“ im Schiebepuzzle sind optimistisch

Heuristische Bestensuche – Algorithmus A

- A0 : Sei z_0 Anfangszustand. Falls dies bereits Zielzustand ist: EXIT(yes)
OPEN := $\{z_0\}$, CLOSED := $\{ \}$
- A1 : Falls OPEN = $\{ \}$: EXIT(no)
- A2 : Sei z der erste Zustand in OPEN
Falls z Zielzustand ist: EXIT(yes)
- A3 : OPEN := OPEN $\setminus \{z\}$, CLOSED := CLOSED $\cup \{z\}$
Bilde Nachfolgermenge SUCC(z); Falls SUCC(z) = $\{ \}$: GOTO A1
- A4 : Für alle $z' \in \text{SUCC}(z)$: $g(z')$ gemäß aktuellem Suchbaum neu berechnen.
- A5 : NEW(z) := SUCC(z) $\setminus \{z' \in \text{CLOSED} \mid g(z') \geq g_{alt}(z')\}$
- A6 : OPEN := OPEN \cup NEW(z), OPEN nach aufsteigendem $g(z') + h(z')$ sortieren, Wiederholungen in OPEN streichen
GOTO A1

Heuristische Bestensuche – Algorithmus A*

- Ohne genauere Angaben zur Heuristik lässt sich wenig über das Verhalten von A sagen.
- Wenn die Heuristik *zulässig* ist, lässt sich beweisen, dass der Algorithmus optimal ist.
- Wir nennen den Algorithmus dann *weiche Form* des *A*-Algorithmus*.
- Die *harte Form* von A* verlangt eine *konsistente* Heuristik. Dafür kann auf die Tests bereits expandierter Knoten verzichtet werden, denn es gilt: Der zuerst entdeckte Pfad zu einem Knoten ist auch der kürzeste.

A*4 : Für alle $z' \in \text{SUCC}(z) \setminus \text{CLOSED}$: $g(z')$ gemäß aktuellem Suchbaum neu berechnen.

A*5 : $\text{NEW}(z) := \text{SUCC}(z) \setminus \text{CLOSED}$

- A* entspricht A mit zusätzlichen Anforderungen an die Heuristik
- Beweise in Nilsson, 1982: Principles of Artificial Intelligence
- Zeitbedarf ist – ausser bei sehr strengen Anforderungen an h , die in der Praxis meist nicht zu erfüllen sind – immer noch exponentiell in der Länge der Lösung. D.h. dass man optimale Lösungen oft nicht erwarten darf.
- Da A* alle Zwischenergebnisse speichert, ist häufig bereits der Speicher der Engpass.

Eigenschaften und Verbesserungen von A*

- vollständig, optimal, Speicherbedarf schlimmstenfalls $O(b^d)$, Zeitbedarf im allgemeinen (z.B. bei $h(n) = 0$ für alle n) auch $O(b^d)$
- Bessere Eigenschaften lassen sich für den allgemeinen Fall nur bei sehr hohen Anforderungen an h zeigen, die häufig nicht zu erfüllen sind. Für ein konkretes Suchproblem und eine gute Heuristik wird A* oft wesentlich besser sein.
- Iterativ Deepening A* (IDA*) führt eine iterative Tiefensuche durch, wobei als Abbruchkriterium gilt: $g(n) + h(n) > C$. Die Schranke C wird bei jedem Durchlauf erhöht. Speicherbedarf $O(n)$ bei leicht erhöhtem Zeitaufwand. Pro Durchlauf werden nicht mehr Knoten als durch A* expandiert. Das Verfahren ist ebenfalls vollständig und optimal.

2 Suche

- Einführung
- Blinde Suche
- Heuristische Suche
- **Problemzerlegung**
- Suche in Spielbäumen

Ein weiteres Modell zur Formulierung von Suchproblemen ist die Problemzerlegung. Sie arbeitet auf Mengen von (Teil-)Problemen. Der Übergang zwischen zwei Problem-mengen kann auf unterschiedliche Weisen erfolgen:

- Die *Zerlegung in Teilprobleme*, von denen alle zu lösen sind.
- Die *Auswahl von Alternativen*, von denen eine zu lösen ist.

Teilprobleme, die sich nicht weiter zerlegen lassen, sind entweder direkt lösbar (*primitiv/terminal*) oder unlösbar (*nicht-terminal*).

- Anfangsknoten: stellt das Ausgangsproblem dar
- Und-Verzweigung: repräsentiert Problemzerlegung
- Oder-Verzweigung: repräsentiert Alternativen
- Knoten ohne Nachfolger (Endknoten): entweder primitive oder unlösbare Probleme

Häufig wird die Normierung vorgenommen, dass sich die Verzweigungsarten abwechseln müssen.

Ein zyklensfreier, endlicher Teilgraph, der folgende Bedingungen erfüllt:

- ① der Anfangsknoten ist enthalten
- ② alle Endknoten sind primitiv
- ③ für alle anderen Knoten des Lösungsgraphen gilt:
 - bei einer Und-Verzweigung sind alle Nachfolger enthalten
 - bei einer Oder-Verzweigung ist genau ein Nachfolger enthalten

Beispiele für Problemzerlegungen

Prolog

Teilprobleme: Subgoals einer Klausel

Problemzerlegung: Anwendung einer Klausel

Alternativen: Klauseln einer Prozedur

primitive Probleme: Fakten

symbolische Integration

Teilprobleme: Teilintegrale

Problemzerlegung: Teilintegrationen (Summen, Partielle Integration, Substitution) und Umformungen

Alternativen: unterschiedliche Zerlegungen

primitive Probleme: bekannte Standardformen (aus Tabellen)

Lösung ist nicht nur true/false, sondern Ableitung bzw. Vorgehensweise bei Integration

SAINT – Symbolic Automatic INTEgrator, Slagle 1963

- Dauer 11 Minuten (Rechner von Anfang der 60er Jahre; LISP-Interpreter); heute bei gleichem Programm im Sekundenbereich
- Vorgehen:
 - ① Integrale mittels Tabelle von Standardformen lösen
 - ② algorithmenartige Umformungen versuchen (Faktoren herausziehen; in Summe zerlegen)
 - ③ heuristische Umformungen (Umformung des Integranten, Substitution, partielle Integration) – Analyse des Integranten auf bestimmte Eigenschaften (Zuhilfenahme problemspezifischen Wissens) – z.B. Schwierigkeitsabschätzung mittels maximaler Tiefe der Funktionsverschachtelung

Beispiele für Problemzerlegungen

Spiele

Teilprobleme: mögliche Spielzustände

Problemzerlegung: gegnerische Züge

Alternativen: eigene Züge

primitive Probleme: Endzustände des Spiels mit Gewinn

Rubik's Cube (Zauber-Würfel)

Teilprobleme: zu erledigende Aufgaben

Problemzerlegung: 1., 2., 3. Schicht ordnen; Ecken ordnen, Kanten ordnen

Alternativen: Auswahl bestimmter Schichten und Steine

primitive Probleme: durch bekannte Zugfolge zu lösen

Äquivalenz zwischen Zustandsraumbeschreibung und Problemzerlegung

„ \Rightarrow “ Und-Knoten auf den Kanten einfügen (trivial)

„ \Leftarrow “

Zustände: Menge offener Teilprobleme

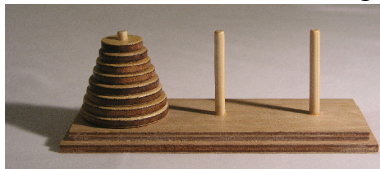
Operatoren: Ersetzung eines Problems in der Menge durch seine Teilprobleme (Problemzerlegung);
Alternative Zerlegungen als Verzweigung;
Primitive Probleme werden aus der Menge gestrichen

Anfangszustand: Menge mit Ausgangsproblem

Zielzustand: Leere Menge

Modellierung der Türme von Hanoi

t_i sind Türme; s_i sind Scheiben; Die Scheiben liegen geordnet auf dem ersten Turm und sollen geordnet auf den letzten Turm gebracht werden. Es darf nie eine größere auf einer kleineren Scheibe liegen.



als Zustandsraum

Zustände: alle Funktionen $f : \{s_1, \dots, s_n\} \rightarrow \{t_1, t_2, t_3\}$
(3^n Zustände)

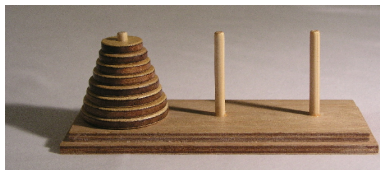
Operatoren: $t_i \mapsto t_j$ für $i, j \in \{1, 2, 3\}$ und $i \neq j$ – oberste Scheibe von t_i nach t_j legen (pro Zustand max. 3 Operatoren anwendbar)

Anfangszustand: $f(s_i) = t_1$ für $i = 1, \dots, n$

Endzustand: $f(s_i) = t_3$ für $i = 1, \dots, n$

Modellierung der Türme von Hanoi

t_i sind Türme; s_i sind Scheiben; Die Scheiben liegen geordnet auf dem ersten Turm und sollen geordnet auf den letzten Turm gebracht werden. Es darf nie eine größere auf einer kleineren Scheibe liegen.



als Problemzerlegung

Ausgangsproblem: n Scheiben von t_1 nach t_3

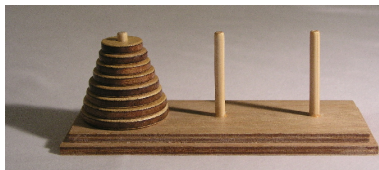
Problemzerlegung: m Scheiben von t_i nach t_j

- $(m - 1)$ Scheiben von t_i nach t_k
- 1 Scheibe von t_i nach t_j
- $(m - 1)$ Scheiben von t_k nach t_j

Primitive Probleme: 1 Scheibe von t_i nach t_j

Modellierung der Türme von Hanoi

t_i sind Türme; s_i sind Scheiben; Die Scheiben liegen geordnet auf dem ersten Turm und sollen geordnet auf den letzten Turm gebracht werden. Es darf nie eine größere auf einer kleineren Scheibe liegen.



- Zustandsraumbeschreibung: Funktion ordnet den Scheiben ihre Plätze zu; von 6 möglichen Operatoren max. 3 anwendbar, wegen Bedingung, dass nur kleine auf großen Scheiben liegen dürfen
- Problemzerlegung: keine Alternativen → Lösungsbaum wird direkt konstruiert

lösbare Knoten

- terminale Knoten
- Knoten mit Und-Verzweigungen: alle Nachfolger lösbar
- Knoten mit Oder-Verzweigungen: mindestens ein Nachfolger lösbar

unlösbare Knoten

- nicht-terminale Knoten
- Knoten mit Und-Verzweigungen: mindestens ein Nachfolger unlösbar
- Knoten mit Oder-Verzweigungen: alle Nachfolger unlösbar

Für endliche Bäume ergibt sich bei Festlegung für die Bewertung der Endknoten eine eindeutige Zerlegung in lösbare und unlösbare Knoten.

Top-Down Verfahren zur Lösungsbaumsuche (1)

- ① $OPEN := \{\text{Startknoten}\}$, $CLOSED := \{ \}$
Falls Startknoten terminal: EXIT(yes)
- ① Sei k der erste Knoten aus OPEN
 $OPEN := OPEN \setminus \{k\}$, $CLOSED := CLOSED \cup \{k\}$,
 $SUCC(k) := \text{Nachfolger von } k$
 $SUCC(k)$ an den Anfang von OPEN (Tiefensuche) bzw.
 $SUCC(k)$ an das Ende von OPEN (Breitensuche)
- ② Falls $SUCC(k) = \{ \}$: GOTO(6) /* Markiere k als unlösbar */
- ③ Falls in $SUCC(k)$ keine terminalen Knoten sind: GOTO(1)
- ④ Markiere die terminalen Knoten in $SUCC(k)$ als lösbar.
Falls dadurch k und weitere Vorgänger von k lösbar werden, so
markiere auch diese als lösbar

Top-Down Verfahren zur Lösungsbaumsuche (2)

- 5 Falls Startknoten als lösbar markiert: EXIT(yes)
Entferne alle Knoten aus OPEN, die einen mit lösbar markierten Vorgänger besitzen
GOTO(1)
- 6 Markiere k als unlösbar. Falls dadurch weitere Vorgänger von k unlösbar werden, so markiere auch diese als unlösbar.
- 7 Falls Startknoten als unlösbar markiert: EXIT(no)
Entferne alle Knoten aus OPEN, die einen mit unlösbar markierten Vorgänger besitzen
GOTO(1)

- Zeitkomplexität $O(b^m)$
- Speicherkomplexität $O(b^m)$ (Breitensuche) oder $O(m)$ (Tiefensuche).
- Zeitkomplexität ist impraktikabel für Anwendungen → Heuristiken nötig.
- Kann angepasst werden, so dass anstelle von lösbar/nicht lösbar Bewertungen der Spielsituationen bestimmt werden.

2 Suche

- Einführung
- Blinde Suche
- Heuristische Suche
- Problemzerlegung
- Suche in Spielbäumen

Wir betrachten ein 2-Personenspiel. Die Spieler A und B ziehen abwechselnd. Am Ende wird der Gewinn bzw. Verlust von A und B numerisch bewertet.

Beispiele für Bewertungen

- Schach: 1/-1 (A gewinnt), 0/0 (Remis), -1/1 (B gewinnt)
- Backgammon: Werte zwischen -192 und +192
- nicht unbedingt antagonistisch: 10/2 3/-8 4/6 (Absprachen, Koalitionen möglich)
- Gefangenendilemma: 2/2 (beide leugnen) 5/5 (beide gestehen) 0/10 bzw. 10/0 (einer von beiden gesteht)

Warum Spiele?

- Abstrakter Wettbewerb, gut repräsentierbar, oft vollständige Informationen, klare Regeln, begrenzte Anzahl von Zügen/Handlungsweisen
- Trotzdem komplex / nicht überschaubar

Erste Schachprogramme Anfang der 50er Jahre (Shannon/Turing):

- Beweis(?) für die Intelligenz(?) eines Programms: „Es kann Schach spielen!“

Suchproblem: günstige Spielzüge finden

Suche in Spielbäumen

Notwendige Komponenten für die Suche

- Anfangszustand
- Operatoren, die die zulässigen Züge beschreiben
- Test ob Endzustand erreicht
- Resultatfunktion (payoff) für die Endknoten, die den Gewinn bzw. Verlust von A und B als numerischen Wert wiedergibt

Darstellung als Baum bzw. Graph möglich

Suche in Spielbäumen

Probleme beim Suchen nach einer Strategie

- Unsicherheit bezüglich gegnerischer Aktionen
- Unüberschaubarkeit durch Komplexität
 - Beispiel Schach: durchschnittlich 35 mögliche Züge, pro Partie ca. 50 Züge (Halbzüge) pro Spieler $\rightarrow 35^{100}$ Knoten (bei ca. 10^{40} unterschiedlichen zulässigen Stellungen)
- begrenzte Zeit für Entscheidungsfindung
- Zufallselemente

Suche in Spielbäumen

Vereinbarungen für die folgenden Verfahren

Es wird verlangt:

- vollständige Informationen
- Nullsummenspiel ($\text{resultat}(B) = -\text{resultat}(A)$),
d.h. Angabe für A reicht
- Spielbaum aus der Sicht von A

Ausgangspunkt: A (*Maximierer*) will maximal mögliches Ergebnis erzielen, B (*Minimierer*) wird versuchen, das Ergebnis von A möglichst gering zu halten (optimale Spielweise von B vorausgesetzt).

Ziel: Strategie, die A in jeder Situation sagt, welcher nächste Zug der beste ist, unter Berücksichtigung aller möglichen Züge von B.

Suche in Spielbäumen

Interpretation als Und-Oder-Baum

Zug von A: Oder-Verzweigung (Alternativen)

Zug von B: Und-Verzweigung (man will allen Reaktionen von B begegnen)

Bei Spielen, die nur die Bewertung 1,-1 (Gewinn,Verlust) haben,
Lösungssuche analog zur Problemzerlegung:

lösbare Knoten: A gewinnt

unlösbare Knoten: B gewinnt

Strategie: im Lösungsbaum Zug zu einem lösbaren Knoten wählen

Theorem

Jedes Spiel mit endlicher Baumstruktur und nur Gewinn/Verlust besitzt entweder eine Gewinnstrategie für A oder eine Gewinnstrategie für B.

Suche in Spielbäumen

Die Minimax-Strategie

Voraussetzung: vollständig entwickelter Baum

S0 : Endknoten mit $\text{resultat}(A)$ bewerten

S1 : Wähle einen Knoten k , dessen Nachfolger alle bewertet sind

- Wenn bei k eine Und-Verzweigung beginnt, erhält k die minimale Bewertung seiner Nachfolger
- Wenn bei k eine Oder-Verzweigung beginnt, erhält k die maximale Bewertung seiner Nachfolger

S2 : Falls Wurzel bewertet: Stop
GOTO S1

Strategie: A wählt in k den Zug zum maximal bewerteten Nachfolger.

Zeitbedarf für vollständige Expansion des Spielbaumes: $O(b^m)$

Möglichkeiten zum Einsparen:

- Die Expansion des Spielbaumes wird vorzeitig abgebrochen; die Qualität des erreichten Zustandes wird abgeschätzt.
- Teilbäume, die aufgrund der bisher ermittelten Bewertungen ohnehin nicht in Frage kommen, werden nicht entwickelt.

Eine Kombination beider Verfahren ist möglich.

Suche in Spielbäumen

Entscheidung ohne vollständige Entwicklung des Baumes

Folgende Ersetzungen bei den Komponenten werden notwendig:

- Test auf Endzustand →
Test ob Abbruch der Suche sinnvoll (cutoff)
- Resultatfunktion →
heuristische Bewertung (utility) der Nützlichkeit/Qualität beliebiger Spielzustände

Bewertungen werden wie bisher per Minimax nach oben propagiert.

Sie soll in den Endzuständen in ihrer Wertung mit der Resultatfunktion übereinstimmen, sie soll die Gewinnchancen von einem Zustand aus widerspiegeln und sie darf nicht zu aufwendig sein.

Häufig werden gewichtete lineare Funktionen verwendet: Es müssen also Kriterien und ihr Einfluss auf die Nützlichkeit der Stellung bestimmt werden. Alternativen: z.B. Neuronale Netze

Beispiel: Schach

- Materialwerte: Bauer (1), Springer u. Läufer (3), Turm (5), Dame (9)
- gute Verteilung der Bauern und Sicherheit des Königs (0,5)
- Werte für beide Spieler aufsummieren und gegeneinander verrechnen.

Die Heuristik hilft alleine noch nicht viel

Beispiel: Schach

Annahmen:

- man kann pro Sekunde 1000 Stellungen durchsuchen (1995)
- im Turnierschach 150s Zeit zum Überlegen
- durchschnittlicher Verzweigungsgrad 35

Die Heuristik hilft alleine noch nicht viel

Beispiel: Schach

Annahmen:

- man kann pro Sekunde 1000 Stellungen durchsuchen (1995)
- im Turnierschach 150s Zeit zum Überlegen
- durchschnittlicher Verzweigungsgrad 35

Ergebnis:

- Suchbaum hat Tiefe von 3 (42 875 Stellungen) bis 4 (ca. 1,5 Mio Stellungen) Halbzügen
- entspricht der Spielstärke eines Anfängers (durchschnittl. menschliche Schachspieler: 6-8 Halbzüge)

Suche in Spielbäumen

Probleme bei unvollständiger Suche

Probleme bei unvollständiger Suche:

Statische Bewertung: Verdichtung aller Bewertungskriterien auf eine Zahl

Horizont-Effekt: entscheidender Spielzug wird wegen Abbruch der Suche nicht betrachtet

Mögliche Lösungen:

- Einsatz von Lernverfahren um gute Gewichtung zu finden
- dynamisches Abbruchkriterium: in unruhigen Situationen (starke Schwankungen in der Bewertung) weiter suchen, bei eindeutig schlechten Situationen vorher abbrechen

A und B werten in jedem Schritt zunächst ein Zufallsereignis (z.B. Würfeln bei Backgammon) aus. Das Ergebnis beeinflusst die Zugmöglichkeiten. Im Spielbaum werden zusätzlich Zufallsknoten eingefügt.

Sei c ein Zufallsknoten. Mit Wahrscheinlichkeit $P(d_i)$ tritt das Resultat d_i ein, welches zum Zustand s_i führt. Der Knoten c wird bewertet mit $value(c) = \sum_i P(d_i) value(s_i)$. Die Bewertung der A-(Maximierer) bzw. B-Knoten (Minimierer) erfolgt wie bisher.

Komplexität $O(b^d n^d)$ – dabei ist n Anzahl der möglichen unterschiedlichen Zufallsergebnisse.

Cubestormer III

17. März 2014: Weltrekord in 3.253 Sekunden

