Kapitel 2: Suche

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

Künstliche Intelligenz und Suche

- Auswahl: Rucksackproblem
- Planung: z.B. Blocksworld, 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.

Zustandsraumsuche

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 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

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:
Zustände: Positionen aller Steine
Operatoren: mögliche Bewegungen des leeren Feldes
Anfangszustand: aktuelle Stellung
Zielzustände: geordnete Stellung

– warum Bewegung des Leerfeldes und nicht der anderen Teile ?

– schwieriger + mehr zu überprüfen

– Modellierung beeinflusst, wie kompliziert die Suche wird

– Feste Abarbeitungsreihenfolge (Inferenz): Tiefe zuerst, Klauselreihenfolge, links-rechts

Generate and Test ?

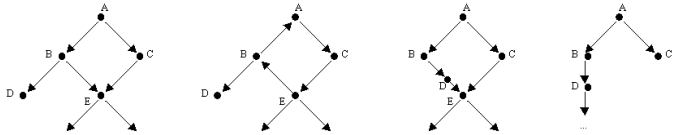
bei den Anfangszuständen beginnend sukzessive Nachfolgezustände generieren und auf Zielbedingung testen.

Problem: kombinatorische Explosion

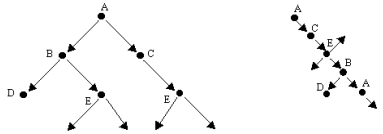
8-er Puzzle: 9! Zustände (davon 9!/2 erreichbar)
bei durchschnittlich 3 Nachfolgern in jedem Zustand und einer typischen Lösungsweglänge von 20 $\Rightarrow 3^{20} \approx$ 3,5 Mrd. Knoten
ungarischer Würfel: rund $4,3 * 10^{19}$ erreichbare Zustände
kürzeste Lösungsfolge max. $40 - 50$ Züge (52 gesichert; 43 vermutet)
Dame: ca. 10^{40} Spiele durchschnittlicher Länge
Schach: ca. 10^{120} Spiele durchschnittlicher Länge
Go: $\leq 3^{361}$ Stellungen

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 - {z }, CLOSED := CLOSED + {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

– warum Schema ?

– 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

Strategien zur Expansion des Suchraumes

- 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

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)

Blinde Suche – 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

Blinde Suche – Tiefensuche (DFS)

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

b – Verzweigungsgrad; d – Tiefe der Lösung; m – maximale Tiefe des Suchbaumes

Verwendung einer Kostenfunktion

- 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 $c(n \rightarrow \acute{n}) \geq \epsilon > 0$ zugeordnet.
- Kosten eines Pfades: $g(n_0 n_1 ... n_k) = \sum_{i=0}^{k-1} c(n_i \rightarrow n_{i+1})$
- Kosten zum Erreichen von z: $g(z) = Min\{g(s) | s$ ist Weg von z_0 nach $z\}$

Blinde Suche – 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 USC ist ein Spezialfall von A*

Fragen:

– Welches Problem gibt es, wenn keine untere Schranke existiert?

– Wieso ist Breitensuche ein Spezialfall von Gleiche Kostensuche?

Iterative Tiefensuche (IDS) 1/2

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

Iterative Tiefensuche (IDS) 2/2

- 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.

Aufwand

– 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

Bidirektionale Suche I

- 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 II

- 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.

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.

Anforderungen an eine Heuristik h

- 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. h_2 heißt **besser informiert** als h_1, wenn gilt: $\forall n : h_1(n) \leq h_2(n)$

Beispiele für Heuristiken

- die am schlechtesten informierte Heuristik: $\forall n : h(n) = 0$
- für Routenplanung: Luftlinie
- für Schiebepuzzle: Anzahl der falsch liegenden Steine
- für Schiebepuzzle: 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 S4 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 S4 nur das am besten bewertete Element aus SUCC(z) an den Anfang von OPEN stellen.
– nicht vollständig

Strahlensuche (Beam Search - BS)
– In S4 die m am besten bewerteten Elemente aus SUCC(z) gemäß ihrer Bewertung ordnen und an den Anfang von OPEN stellen
– nicht vollständig

- Warum Bergsteiger-Metapher?
  - Suchraum als Landschaft interpretieren, h gibt Höhe an
- Warum läuft man nicht nach unten (Abstieg gehört ja auch zu Bergsteigen)?
  - 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 Verlaufen, immer bergab laufen. Irgendwann findet man Tal oder Gewässer (= bevorzugte Siedlungsplätze).

Vorteile lokaler Verfahren

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

- Raum-/Zeitbedarf sind abhängig von der Heuristik
- Übung: Expansion der Suchbäume bei verschiedenen Verfahren zeigen

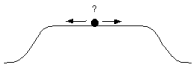
Vorteile lokaler Verfahren

- 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).

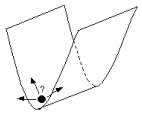
Probleme beim Bergsteigen

lokale Minima - vorübergehend keine Verbesserung möglich

Plateaus - keine Unterschiede in der Bewertung



enge Talsohlen - vorhandene Operatoren erfassen den Abstieg nicht



Heuristische Suche mit schrittweiser lokaler Verbesserung I

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 abh. von Anzahl der lokalen Minima.

Heuristische Suche mit schrittweiser lokaler Verbesserung II

Simulated Annealing (Simuliertes Abkühlen)
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.

- Modelliert das Erstarrungsverhalten von Flüssigkeiten

- Erfahrungen vom Stahlkochen: erst stark erhitzen, dann langsam abkühlen ergibt kristalline Struktur, d.h. harten Stahl

- 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 lassen (kontinuierlicher Raum); Schrittweite verändern z.B. bei Lernverfahren

Heuristische Suche - 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.

Weitere mögliche Eigenschaften einer Heuristik

- 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: $h(n) - h(n') \leq c(n \rightarrow n')$. 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 - {z }, CLOSED := CLOSED ∪ {z }
Bilde Nachfolgermenge SUCC(z), Falls SUCC(z) = { }: GOTO A1

A4 : für alle $z' \in$ SUCC(z) $g(z')$ gemäß aktuellem Suchbaum neu berechnen
NEW(z) := SUCC(z) - CLOSED
OPEN := OPEN ∪ NEW(z), OPEN nach aufsteigendem $g(z')+h(z')$ sortieren,
Wiederholungen in OPEN streichen
GOTO A1

– Änderungen gegenüber dem bisherigen Schema: Schritt 2 und 3 getauscht und modifiziert; Schritt 4 komplexer

– ohne genauere Angaben zur Heuristik lässt sich wenig über das Verhalten von A sagen

Heuristische Bestensuche - Algorithmus A*

In der **weichen Form** von A* wird verlangt, daß die Heuristik **zulässig** ist. A* entsteht aus A durch folgende Modifikation:

A*4 : für alle $z' \in$ SUCC(z) $g(z')$ gemäß aktuellem Suchbaum neu berechnen:
NEW(z) := SUCC(z) - $\{z' \in CLOSED | g(z) \geq g_{alt}(z')\}$
OPEN := OPEN ∪ NEW(z), OPEN nach aufsteigendem $g(z') + h(z')$ sortieren, Wiederholungen in OPEN streichen
GOTO A1

Heuristische Bestensuche - Algorithmus A*

Die **harte Form** von A* verlangt eine **konsistente** Heuristik. Dafür kann auf die Tests bereits expandierter Knoten verzichtet werden, denn es gilt: Der A*–Algorithmus in der harten Form untersucht den selben Knoten nicht mehrfach.

A*4 : für alle $z' \in$ SUCC(z) $g(z')$ gemäß aktuellem Suchbaum neu berechnen
NEW(z) := SUCC(z) - CLOSED
OPEN := OPEN ∪ NEW(z), OPEN nach aufsteigendem $g(z') + h(z')$ sortieren, Wiederholungen in OPEN streichen
GOTO A1

– 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.

Problemzerlegung

Ein weiteres Modell zur Formulierung von Suchproblemen ist die Problemzerlegung. Sie arbeitet auf Mengen von (Teil-)Problemen. Der Übergang zwischen zwei Problemmengen 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**).

Interpretation als Und-Oder-Baum bzw. -Graph

- 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.

Lösung eines Problems im Und-Oder-Graphen

Ein zyklenfreier, 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

– SAINT - Symbolic Automatic INTegrator, Slagle 1963

– Dauer 11 Min (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 Intgration) – Analyse des Integranten auf bestimmte Eigenschaften (Zuhilfenahme problemspezifischen Wissens) – z.B. Schwierigkeitsabschätzung mittels maximaler Tiefe der Funktionsverschachtelung

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

Beispiele für Problemzerlegungen

Spiele:
Teilprobleme: mögliche Spielzustände
Problemzerlegung: gegnerische Züge
Alternativen: eigene Züge
primitive Probleme: Endzustände des Spiels mit Gewinn

Zauberwürfel:
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

Umformung 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

– 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
– Eine bessere Formulierung des Problems würde den Aufwand verringern

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 $f : \{s_1, ..., s_n\} \rightarrow \{t_1, t_2, t_3\}$ (3^n Zustände)
  Operatoren: $t_i \rightarrow t_j$ 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 $i = 1, \ldots, n$
  Endzustand: $f(s_i) = t_3$ für $i = 1, \ldots, n$

- 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

Bewertung der Knoten

lösbare Knoten

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

unlösbare Knoten

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

(0) OPEN := {$Startknoten$}, CLOSED := {}
Falls $Startknoten$ terminal: EXIT(yes)

(1) Sei k der erste Knoten aus OPEN
OPEN := OPEN - {k}, CLOSED := CLOSED $\cup$ {k}, SUCC(k) := 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)

Komplexität

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

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

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

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

Vereinbarungen für die folgenden Verfahren

Es wird verlangt:
– vollständige Informationen
– Nullsummenspiel (resultat(A) = -resulat(B)) → 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.

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

Jedes Spiel mit endlicher Baumstruktur und nur Gewinn/Verlust besitzt entweder eine Gewinnstrategie für A oder eine Gewinnstrategie für B. (Bsp.: Tic-Tac-Toe)

Die Minimax-Strategie

Voraussetzung: vollständig entwickelter Baum

S0 Endknoten mit 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.

Komplexität von Minimax

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.

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.

Wahl der Heuristik

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 und 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

Ergebnis:

- Suchbaum hat Tiefe von 3 (42875 Stellungen) bis 4 (ca. 1,5 Mio Stellungen) Halbzügen
- entspricht der Spielstärke eines Anfängers (durchschnittl. menschliche Schachspieler: 6–8 Halbzüge)

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

Spiele mit Zufallselementen

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.

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