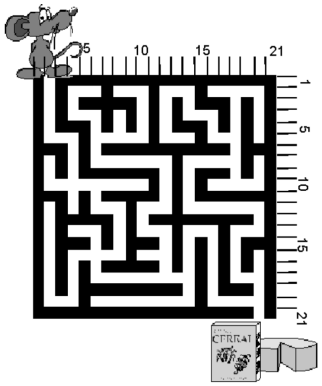
Übungsblatt: Suchverfahren, Modellierung

— LÖSUNG —

1 Aufgabe 1 - Maus im Labyrinth

Im KI-Hörsaal wurde ein Labyrinth aufgestellt. In diesem befindet sich eine Maus. Die Maus möchte nun zu ihrem Futter gelangen, kennt jedoch den Weg (weiße Felder) nicht.



1.1 Suchstrategien

Die Maus hat in der Vorlesung gut aufgepasst und möchte den Weg durch das Labyrinth mit Hilfe eines uninformierten Suchverfahrens, der **Tiefensuche**, bestimmen. Warum ist die Tiefensuche hier eine sinnvolle Suchstrategie im Vergleich zu den anderen Strategien? Bitte begründen Sie Ihre Antwort.

Lösung

- Breitensuche: Praktisch nicht möglich, da sonst viel unnötig hin- und hergelaufen werden muss. Ziel: irgendein Weg, nicht der beste
- Tiefensuche: Normales Vorgehen für ein Labyrinth ("gehe immer an der rechten Wand entlang"). Jeder Weg wird höchstens doppelt gelaufen
- Uniforme Kostensuche: wie Breitensuche

- Tiefenbeschränkte. Suche: Nicht zielführend, da wir nicht wissen wie tief die Lösung ist
- Iterative Tiefensuche: Könnte Sinn machen, wenn wir vermuten, dass $d << m$ Allerdings müssen viele Wege öfter gelaufen werden

1.2 Problemformulierung

Für die Konzeption des von Ihnen ausgewählten Suchverfahrens sind die folgenden Aspekte wichtig:

1. Zustandsraum

   Lösung Mehrere Möglichkeiten! Alle Felder (Z: Maus auf Feld), alle Abzweigungen (Z: Maus auf Abzweigung)

2. Operatoren

   Lösung Mehrere Möglichkeiten! Maus im Labyrinth: gehe links, rechts, geradeaus, Vogelperspektive: gehe Nord, Ost, Süd, West

3. Anfangszustand

   Lösung Bei Zustandsraum aus Feldern: Maus ist auf Feld (2,1) (links oben); Bei Zustandsraum aus Kreuzungen: Maus ist auf Feld (4,10) (erste Kreuzung)

4. Zieltest

   Lösung Bei Zustandsraum aus Feldern: Maus ist auf Feld (20,21 = rechts unten); Bei Zustandsraum aus Kreuzungen: ...Ziel nicht im Zustandsraum! → Zustandsraum erweitern um "Ende des Wegs"

5. Pfadkosten

   Lösung Pfad: Sequenz von Aktionen, die von einem Zustand zu einem anderen führen. Pfadkosten: Kostenfunktion g über Pfaden. Setzt sich normalerweise aus der Summe der Kosten der Aktionen zusammen. Z.B. 0,5 Sekunden pro Kästchen

6. Suchkosten

   Lösung Im Fall der Tiefensuche im Labyrinth sind die Suchkosten gleich den Pfadkosten (jeder "abgesuchte" Weg muss ja abgelaufen werden!). Keine Offline-Suche vor Betreten des Labyrinths ohne Karte möglich!

7. Vollständigkeit

   Lösung (Falls eine Lösung existiert, wird sie auch gefunden.) Tiefensuche: ja, falls keine unendlichen Pfade und keine Zyklen existieren

8. Optimalität

   Lösung Tiefensuche: nein, es könnte sein, dass es einen kürzeren Weg gibt; Breitensuche wäre optimal (aber nicht praktikabel)

9. Worst-Case Komplexität (Zeit und Platz)

   Lösung Tiefensuche: Zeitkomplexität b^m, Platz $b*m$, wobei $b \approx 3$ (schwer abzuschätzen, max. drei Wege pro Kreuzung); m = max. Tiefe des Suchbaums: kommt auf Zustandsraum an Falls Zustandsraum "alle Kästchen" ist m = Länge des längsten Weges; falls Zustandsraum "alle Kreuzungen" ist m = maximale Anzahl an aufeinanderfolgenden Kreuzungen

   Achtung: m und b sind abhängig! Mehr Wege → weniger Kreuzungen bzw. mehr Kreuzungen → kürzere Wege! Zustandsraum "alle Kreuzungen" macht Abschätzung einfacher; z.B. ca. 400 Kästchen insgesamt, pro Kreuzung ca. 50 Kästchen → Daumen: $m = 8$ Kreuzungen, max 3 Wege/-Kreuzung; Worst case: Zeit: 3^8 Wegstrecken laufen, Platz: $3*8$ Kreuzungen „merken"

Beschreiben Sie diese anhand einer Tiefensuche. Gehen Sie bei der Realisierung der Lösung davon aus, dass die Maus 0,5 Sekunden benötigt um von einem Feld des Labyrinths ins nächste zu gelangen.

1.3 Probleme

Welche Probleme treten auf, wenn es mehr als einen möglichen Weg zum Futter gibt? Wie kann die Maus dennoch - in endlicher Zeit - einen Weg finden? Können Sie sicherstellen, den kürzesten Weg zu finden?

Lösung

- Mehr als ein Weg → Zyklen möglich!
- Tiefensuche könnte in eine Endlosschleife geraten!
- Lösung:
  - merken, welche Knoten man bereits expandiert (= besucht) hat
  - falls man wieder auf einen dieser Knoten stößt, kommt: nicht nochmal expandieren, sondern nächsten zu expandierenden Knoten wählen (= umdrehen und zur letzten Kreuzung zurückgehen)
  - Außerdem: Abzweigung als "besucht"/expandiert markieren

2 Tourenplanung

Gegeben sei folgendes Problem: Ihr Unternehmen verfügt über insgesamt n verschiedene Niederlassungen und an jeder Niederlassung ist genau ein LKW stationiert. Mit diesen n LKWs müssen jeden Tag alle m Kunden Ihres Unternehmens beliefert werden. Es ist davon auszugehen, dass die Anzahl der Kunden wesentlich größer ist als die Anzahl der LKWs ($m >> n$). Sie sind nun mit der Aufgabe betraut einen Einsatzplan für ihre LKWs mit minimalen Kosten zu ermitteln. Der Einsatzplan, welcher jeweils immer für einen Tag erstellt wird, beschreibt in welcher Reihenfolge die Kunden durch die n verschiedenen LKWs angefahren

werden. Die Kosten der LKWs (und damit die Kosten für den Einsatzplan) sind direkt abhängig von der Anzahl der gefahrenen Kilometer. Bitte berücksichtigen Sie bei der Lösung, dass jeder LKW pro Tag nicht mehr als k Kunden beliefern kann (wobei jedoch $k * n > m$) und jeder LKW am Ende seiner Tour wieder an seinen Heimatstandort zurückkehren muss.

2.1 Uniforme Kostensuche

Sie möchten für das gegebene Problem eine uniforme Kostensuche entwerfen. Geben Sie für das Problem den Zustandsraum, Suchoperatoren, Aktionskosten und einen Zieltest an.

Lösung

- Zustand / Zustandsraum

  Kunden: $K_1, \ldots, K_m$ Niederlassungen $N_1, \ldots, N_n$

  Vektor L der Länge n, mit l_i = Position von LKW i Gibt an, welcher LKW gerade welchen Kunden beliefert.

  Beispiel:

  $(N_1, K_1, K_5) \rightarrow$ LKW1 ist bei Niederlassung1, LKW2 bei Kunde 1,LKW3 bei Kunde 5

  Zusätzlich: Merken, welche Kunden schon besucht worden sind! Vektor B der Länge m mit $b_i = 1$ falls Kunde bereits besucht

  Beispiel:

  $(0, 1, 0, 0) \rightarrow$ bisher nur Kunde 2 besucht

  Und: Kapazitätsrestriktion beachten!

  Vektor R der Länge n, $k_i = 1$ falls Kapazitätsrestriktion erreicht ist

  Beispiel: $(0, 0, 1) \rightarrow$ LKW 3 ist voll!

  Eine spezielle Belegung von L,B,R ist ein Zustand $\rightarrow$ Kreuzprodukt $L \times B \times R$ ist Zustandsraum

- Operator

  Bewege LKW i (nur ein LKW auf einmal!)

  z.B.

  $(N1, K1, K5) \rightarrow (N1, K2, K5)$

  oder

  $(N1, K1, K5) \rightarrow (K3, K1, K5)$

  Passe Vektor B (besuchte Kunden) und Vektor R (Kapazitätsrestriktion) an. Achtung: manche Operationen sind nicht zulässig (Kapazitätsrestriktion erreicht)

- Pfadkosten Anzahl der Kilometer, die alle LKW bisher zurücklegen mussten $\rightarrow$ addiere die zusätzlichen Kosten bei Expansion eines Knotens

- Suchstkosten

  Aufwand durch das Expandieren von Knoten für das Finden einer Lösung (CPU-Zeit, Speicher)

- Zieltest

  Vektor $L = (N1, N2, ..., Nn)$ (alle LKW daheim) und

  Vektor $B = (1, 1, ..., 1)$ (alle Kunden wurden besucht)

2.2 Optimalität

Kann sichergestellt werden, dass die von Ihnen entwickelte uniforme Kostensuche für das gegebene Problem die optimale Lösung findet?

Lösung

Ja, da $g(succ(n)) > g(n)$ (siehe Skript)

g: Pfadkosten zu Knoten n

$succ(n)$: Nachfolger von Knoten n

Keine negativen Kosten (Strecken) möglich!

Wenn Lösung in Blatt b gefunden: prüfen, ob es noch einen expandierbaren Knoten l gibt, für den gilt $g(l) < g(b)$

3 Das Rucksackproblem

Beim Rucksackproblem sollen aus einer Menge von Objekten, die jeweils ein Gewicht und einen Nutzenwert haben, eine Teilmenge von Objekten so ausgewählt werden, dass das Gesamtgewicht der ausgewählten Objekte eine vorgegebene Gewichtsschranke nicht überschreitet und deren Nutzenwert maximiert wird. Bekanntestes Beispiel für das Rucksackproblem ist das Packen eines Rucksacks derart, dass ein vorgegebenes Maximalgewicht nicht überschritten wird und der Wert der eingepackten Güter maximal ist.

Das Problem lässt sich formal in folgender Weise beschreiben:

Gegeben sei eine Menge U von n Objekten. Jedes Objekt $i, i = 1, \ldots, n$ hat einen Nutzenwert v_i und ein Gewicht w_i . Weiterhin existiert eine Schranke w_{max} (Kapazität des Rucksacks). Gesucht ist eine Teilmenge $K \subseteq U$ mit m Elementen ($m \leq n$), die die Bedingung $\sum_{i \in K} w_i \leq w_{max}$ einhält und $\sum_{i \in K} v_i$ maximiert.

3.1 Suchbaum

Sie wollen einen Binärbaum als Suchbaum für das skizzierte Rucksackproblem entwerfen. Entwickeln Sie hierfür ein Modell. Beschreiben Sie dabei die möglichen Zustände, Suchoperatoren, Aktionskosten und einen Zieltest.

Lösung

- Zustandsraum Binärbaum: Für jeden Gegenstand wird entschieden, ob er eingepackt wird oder nicht.

Ein Zustand ist definiert durch einen Vektor R der Länge n besetzt mit Nullen und Einsen. 1 an Position i bedeutet, Objekt i ist eingepackt.

$(1, 0, 0, 1, \ldots, 0) \rightarrow$ Objekte 1+4 sind eingepackt Tiefe des Suchbaums bestimmt, wie viele Objekte bereits betrachtet wurden.

- Suchoperator

  Entscheide für das nächste Objekt, ob es eingepackt wird oder nicht. Achte dabei auf Kapazitätsrestriktionen.

- Pfadkosten

  Gewicht der bisher eingepackten Gegenstände.

- Zieltest

  Alle Objekte betrachtet oder Kapazitätsschranke erreicht.

3.2 Zeitkomplexität

Wie hoch ist die Zeitkomplexität für die Lösung des Problems bei Verwendung von Breitensuche?

Lösung

Für den Binärbaum: $b^n \rightarrow 2^n$

3.3 Gierige Suche

Erläutern Sie kurz die Funktionsweise von gieriger Suche. Bezüglich welcher Kriterien können Sie die Qualität von unterschiedlichen Verfahren der gierigen Suche beurteilen.

Lösung

Allgemein: gierig: benutzt momentan vorhandene (lokale) Information, trifft kurzfristig beste Entscheidung

heuristische Funktion $h(n)$, schätzt Distanz von Knoten n zu einem Zielknoten, wobei $h(\mathit{Zielknoten}) = 0$; die Qualität der gierigen Suche hängt ab von der Güte der Schätzfunktion ab.

Problem: nicht optimal, manchmal extrem schlecht; Vorteil: schnell und einfach, meistens gut

Beispiele, wo Greedy sogar zum Optimum führt: Kruskal's algorithm, Prim's algorithm (minimale aufspannende Bäume), Dijkstra's algorithm (kürzeste Wege) und andere

3.4 Suchstrategie

Skizzieren Sie eine mögliche Strategie für eine gierige Suche für das Rucksackproblem.

Lösung Sortiere mögliche Knoten und expandiere "besten" Knoten, falls möglich (Kapazität!)

Sortierung, z.B.: nach Wert; nach Wert pro Gewichtseinheit v_i/w_i

3.5 Mathematische Beschreibung

Wandeln Sie die mathematische Beschreibung des Problems so ab, so dass nicht nur eine Gewichtsschranke w_{max} existiert, sondern auch eine Volumenschranke v_{max} . Beeinflusst dies Ihre Lösungen?

Lösung $\sum_i v_i \leq v_{max}$

Keine grundsätzliche Änderung bei Problemformulierung, außer weiterer Kapazitätsrestriktion (Prüfung vor dem Expandieren). Suchbaum wird zusätzlich beschnitten (also kleiner)

Greedy-Strategie muss angepasst werden! Nach was wird jetzt sortiert?

- Wert pro Gewichtseinheit v_i/w_i
- Wert pro Volumeneinheit v_i/vol_i
- Kombination $v_i/(c_1 * w_i + c_2 * v_i)$, mit c_1, c_2 Konstanten