Prof. Dr. Gerhard Weiss
Department of Knowledge Engineering
Maastricht University

Einführung in die Künstliche Intelligenz

— Übungsblatt 2 —

Übung 2-1 (Greedy Strategie und heuristische Funktionen)

Im Unterschied zu uninformierten Suchstrategien (wie z.B. Breiten- und Tiefensuche) verwenden informierte Suchstrategien zusätzlich zum "reinen Suchwissen" (Knotenbeschreibungen, Nachfolgerfunktionen, usw.) problemspezifisches Wissen, um die Suche effizienter zu gestalten. Ein prominentes Beispiel für solches Wissen sind Abschätzungen der tatsächlichen Rest-Kosten von einem Knoten im Suchbaum zu einem Zielknoten. In der Literatur wird häufig folgende Abschätzung und Notation verwendet:

$h(n) =$ "geschätzte Kosten des günstigsten Pfades von Knoten *n* zu einem Zielknoten" ,

wobei $h(n)$ als *heuristische Funktion* bezeichnet wird. h dient als Approximation der "tatsächlichen Kostenfunktion" h^*; $h^*(n)$ sind also die tatsächlichen Rest-Kosten des günstigsten Pfades von *n* zu einem Zielknoten. (Wäre h^* bekannt, dann wäre keine Suche erforderlich.) Die Idee hinter der Verwendung von heuristischen Funktion ist offensichtlich, einen Suchbaum in Richtung der geschätzten minimalen Rest-Kosten zu expandieren. Die Suchstrategie, nach der immer derjenige Knoten mit den geringsten zu erwartenden Kosten expandiert wird, ist unter der Bezeichung *greedy search* ("gierige Suche") bekannt.

Zur Veranschaulichung von heuristischen Funktionen betrachten wir das Landkartenproblem und das 8-Puzzle-Problem (siehe Übungsblatt 1). Ein Beispiel für eine heuristische Funktion für das Landkartenproblem ist

$h_{Land}(n) =$ "Luftlinienentfernung (straight-line distance) zwischen *n* und Bucharest" .

zwei Beispiele für heuristische Funktionen für das 8-Puzzle-Problem sind

$h_{Puzzle1}(n) =$ "Anzahl der im Zustand *n* falsch plazierten Ziffern"

und

$h_{Puzzle1}(n) =$ "Summe der Entfernungen aller Ziffern in *n* von ihren Zielpositionen" .

Natürlich können sich heuristische Funktionen ganz erheblich in ihrer Qualität unterscheiden, je nachdem, wie gut sie die tatsächlichen Kosten approximieren und welche sonstige Eigenschaften sie besitzen. Zwei Eigenschaften, die häufig gefordert werden (und mit deren Hilfe verschiedene

Konvergenzbeweise für informierte Suchverfahren geführt wurden), sind *Zulässigkeit* (engl. admissibility) und *Monotonie*. Eine heuristische Funktion h heißt zulässig, falls sie die tatsächlichen Restkosten niemals überschätzt:

$$h(n) \leq h^*(n) \ \forall \ n \quad .$$

Eine heuristische Funktion h heißt monoton, falls für jeden Knoten n und jedem seiner unmittelbaren Nachfolger m gilt:

$$h(n) \leq h(m) \ + \text{“Kosten für Übergang von } n \text{ zu } m\text{”} \quad .$$

2-1-1. Geben Sie einen frei konstruierten (also “erfundenen”) Pfadausschnitt eines Suchbaums an, mit dessen Hilfe illustriert wird, dass es heuristische Funktionen gibt, die zulässig aber nicht monoton sind. Beschriften Sie hierzu jeden Knoten mit seinen h- und h^*-Kosten.

2-1-2. Überprüfen Sie die oben angegebenen heuristischen Funktionen h_{Land}, $h_{Puzzle1}$ und $h_{Puzzle2}$ auf ihre Zulässigkeit und Monotonie. Sind sie zulässig und/oder monoton? Begründen Sie Ihre Antworten knapp.

2-1-3. Betrachten Sie nochmals das Türme-von-Hanoi-Problem mit n Scheiben (Abbildung 2-2 illustriert dieses Problem für $n = 5$). Geben Sie für dieses Problem eine möglichst gute heuristische Funktion an. Achten Sie dabei darauf, dass es im Laufe des Lösungsprozesses erforderlich sein kann, bereits richtig positionierte Scheiben wieder zu entfernen (“lokale Maxima”). Erklären Sie kurz die Ihre heuristische Funktion.

2-1-4. Implementieren Sie die greedy search Strategie und wenden Sie Ihr Programm auf das Landkartenproblem (unter Verwendung von h_{Land}) und das 8-Puzzle-Problem (unter Verwendung von $h_{Puzzle1}$ und $h_{Puzzle2}$).

(a) Wie lauten die gefundenen Lösungen? Welche der beiden Heuristiken $h_{Puzzle1}$ und $h_{Puzzle2}$ liefert die besseren Ergebnisse? Vergleichen Sie die Lösungen mit denen zu den Übungen 1 und 2 (Anzahl der Suchschritte usw.) und stellen Sie die Resultate übersichtlich (in Tabellenform o.ä.) dar.

(b) Überlegen Sie, ob es generell “allgemein gültige Kriterien” gibt, mit deren Hilfe man entscheiden kann, ob bei irgendeinem gegebenen Problem uniform-cost search oder greedy search angewendet werden sollte.

2-1-5. Wenden Sie Ihr “greedy Programm” unter Benutzung der von Ihnen in 1-4-3 vorgeschlagenen Heuristik auf das Türme-von-Hanoi-Problem für $n = 5$ und $n = 10$ an. Listen Sie die gefundene(n) Lösungssequenz(en) auf.

2-1-6. Überlegen Sie alternative Heuristiken für das 8-Puzzle-Problem. Wenden Sie Ihr “greedy Programm” unter Benutzung dieser Heuristiken an und vergleichen Sie die Resultate mit den Resultaten, die Sie mit den beiden angegebenen Heuristiken $h_{Puzzle1}$ und $h_{Puzzle2}$ erzielt haben.

Übung 2-2 (A* Strategie)

Die bekannteste unter den informierten Suchstrategien ist die sogenannte A* Strategie. Dieses Verfahren kombiniert die Vorteile von *uniform-cost search* (d.h. es wird immer der Knoten mit geringsten bisher angefallenen Kosten expandiert) und *greedy search* (d.h. es wird immer der Knoten mit den geringsten noch zu erwartenden Kosten expandiert). Die Beliebtheit von A* rührt daher, dass dieser Algorithmus unter gewissen Bedingungen die optimale Lösung garantiert findet und kein anderer garantiert optimaler Algorithmus weniger Knoten expandiert. Die Kernidee von A* ist es, die Knoten im Suchbaum abhängig von ihren Kosten zu expandieren. Die Kosten $f(n)$ eines Knotens n sind dabei gegeben sind durch

$$f(n) = g(n) + h(n) \quad ,$$

wobei $g(n)$ wie in der Übung 2-1 und $h(n)$ wie in der Übung 1-4 definiert sind und insbesondere h *zulässig* sein muss. (Wegen der Zuässigkeit von h überschätzt f niemals die tatsächlichen *Gesamt*kosten des günstigsten Pfades durch n.) Unter A* wächst der Suchbaum immer an dem Knoten ($\neq$ Zielknoten), der unter allen "Expansionskandidaten" den kleinsten f-Wert besitzt. (Bei gleich kleinen f-Werten kann unter den entsprechenden Knoten der zu expandierende Knoten zufällig ausgewählt werden.) Abbildung 2-1 gibt einen Pseudocode für eine sehr einfache Realisierung des A* Verfahrens an.

2-2-1. Kann es sein, dass A* mit einem suboptimalen Pfad terminiert oder ist garantiert, dass der als Lösung ausgegebene Pfad automatisch der kostengünstigste ist? Formulieren Sie Ihre Überlegungen knapp.

2-2-2. Implementieren Sie die in Abbildung 2-1 angegebene Variante des A* Verfahrens und testen Sie Ihr Programm an den Problemen "Landkarte" und "8-Puzzle" unter Verwendung der in Übung 1 angegebenen heuristischen Funktionen h_{Land} und (wahlweise) $h_{Puzzle1}$ bzw. $h_{Puzzle2}$. (Achtung: auf Blatt 1 wurden fälschlicherweise *beide* "Puzzle"-Heuristiken mit $h_{Puzzle\mathbf{1}}$ bezeichnet.) Vergleichen Sie die Resultate mit denen zur Übung 1 (bezüglich Anzahl der Lösungsschritte und Knotenanzahl).

2-2-3. Der in Abbildung 2-1 angegebene Pseudocode ist recht einfach – und leider auch ineffizient, da nicht überprüft wird, ob neu expandierte Knoten bereits in OPEN oder CLOSED enthalten sind. Dies spart zwar Zeit bei der Knotengenerierung, kann aber dazu führen, dass derselbe Knoten mehrfach im Baum auftritt. Wie müsste der Pseudocode modifiziert bzw. erweitert werden, damit solche Knotenduplikate vermieden werden? Skizzieren Sie knapp Ihre Überlegungen (Grundidee statt Details).

Bearbeitungen in PDF- *und* Source-Format (World, LaTeX, etc.)
bis zur nächsten Vorlesung an: gerhard.weiss@maastrichtuniversity.nl

---

1. Initialisiere eine Liste OPEN mit dem Startknoten n_0. Setze dessen g-Wert auf 0 und dessen h-Wert entsprechend der verwendeten Heuristik. (Für den Startknoten gilt also $f(n_0) = g(n_0)$.)

2. Initialisiere eine leere Liste CLOSED.

3. Wenn OPEN leer ist, stoppe mit Fehlermeldung.

4. Wähle den ersten Knoten aus OPEN und nenne ihn BESTNODE. Entferne ihn aus OPEN und füge ihn in CLOSED ein.

5. Wenn BESTNODE ein Zielknoten ist, dann stoppe mit Erfolgsmeldung und gebe den Pfad vom Startknoten zu BESTNODE aus.

6. Expandiere BESTNODE, d.h. generiere eine Liste SUCCESSOR aller Nachfolgerknoten von BESTNODE.

7. Für jeden Knoten n in SUCCESSOR mache folgendes:

   7.1. setze einen Vermerk – "Rückwärtspointer" – von n auf BESTNODE (diese Rückwärtspointer auf die Elternknoten sind erforderlich, um abschliessend den gesamten Lösungspfad angeben zu können),

   7.2. berechne $g(n) = g(\text{BESTNODE})$ + Kosten für Übergang von BESTNODE zu n,

   7.3. berechne $f(n) = g(n) + h(n)$ und

   7.4. füge n in OPEN ein.

8. Ordne die Knoten in OPEN (in aufsteigender Reihenfolge ihrer f-Werte).

9. Gehe zu Schritt 3.

Abbildung 2-1: Pseudocode für einfache A* Variante.