Beweise aus dem Fach
Grundzüge der Künstlichen Intelligenz

Alexander Pacha
TU Wien - Matr. Nr.: 0828440
alexander.pacha@tuwien.ac.at

1 Begriffserklärungen

Für die folgenden Beweise werden zuerst folgende Begriffe benötigt:

1.1 Uninformierte Suche

Darunter versteht man die Suche in einem Suchbaum, in dem „blind“ gesucht wird, sprich ohne weitere Informationen über das Problem (z.B. bisherige Kosten, Nutzen, ...). Im Gegensatz dazu steht der informierten Suche gewisse Information zur Verfügung, mittels der man andere Suchverfahren (A*-Suche,...) definieren und anwenden kann.

1.2 Asymptotisches Laufzeitverhalten

Dieser Begriff stammt aus der Analyse von Algorithmen. Hierbei beschreiben die Funktionen f(n) allgemeine Schranken, ab einem gewissen n_0, wobei ein konstanter Faktor c vernachlässigt wird. Allgemein beschreibt man Laufzeiten und Speicherkomplexität durch folgende Gleichungen (Theta-Notation und O-Notation):

$$\theta(g(n)) = \{f(n)|(\exists c_1, c_2, n_0 > 0), (\forall n > n_0) : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

stellt eine allgemeine Beschreibung (Ober- und Unterschranke) dar.

$$O(g(n)) = \{f(n)|(\exists c, n_0 > 0), (\forall n > n_0) : 0 \leq f(n) \leq cg(n)\}$$

Beschreibt nur eine obere Schranke (meistens interessanter, vorallem bei aufwendigen und komplexen Problemen ist eine untere Schranke kaum von Interesse).

1.3 Suchbaum

Unter einem Suchbaum versteht man die Expansionsmöglichkeiten eines Problems. Sprich, wenn ich einen Schritt nach dem anderen mache, und verschiedene Alternativen habe, wohin gehe ich. In unseren Beweisführungen gehen wir von einem idealisierten Suchbaum aus, der einen konstanten Teilungsfaktor (Branching) b und eine vorgegebene Tiefe d (Deepness) hat.

Ein typischer Suchbaum sieht also in etwa so aus:

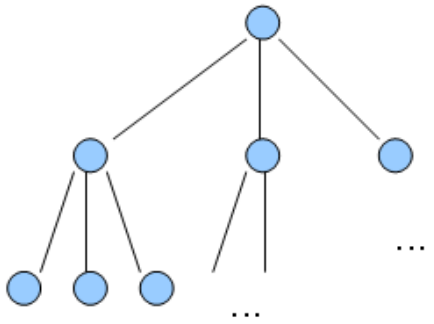

Abbildung 1: Ein einfacher Suchbaum

Dieser Suchbaum hätte b=3, weil jeder Knoten genau 3 Kinder hat und eine Tiefe von 3, da der Baum 3 Ebenen (Wurzel, Kinder der Wurzel, Kinder der Kinder der Wurzel) hat.

1.4 Breitensuche

Breitensuche geht in dem Suchbaum Ebene für Ebene durch. Also im obigen Suchbaum wird zuerst die Wurzel durchsucht (sprich expandiert), dann die 1. Ebene usw. Die Breitensuche findet die optimale Lösung hat aber als großes Problem, dass der Speicheraufwand (die Ressource, von der man nur eine stark beschränkte Menge hat) mit der größe des Baumes zunimmt, da der gesamte bisher durchlaufene Baum gespeichert wird. Allgemein gilt folgendes für die Breitensuche:

- Sie ist vollständig
- hat eine Zeitkomplexität von $O(b^d)$
- hat eine Speicherkomplexität von $O(b^d)$
- und ist optimal

1.5 Iterativ vertiefte Tiefensuche

Die Tiefensuche ist grundsätzlich anders aufgebaut. Hier wird ein Pfad (von der Wurzel zu Blättern) verfolgt und erst später die Nachbarknoten expandiert (und zwar, wenn keine Kinder mehr expandiert werden können). Dadurch wird zwar der Speicherverbrauch eingeschränkt, dafür findet man möglicherweise die optimale Lösung nicht (wenn z.B. Zyklen vorhanden sind, und die Tiefensuche in einer Endlosschleife hängen bleibt).

Daher beschränkt man die Tiefensuche auf eine bestimmte Tiefe x, versucht alle Lösungen mittels Tiefensuche zu finden, und erhöht anschließend x um eins, und startet die Tiefensuche erneut.

Allgemein gilt folgendes für die Iterativ vertiefte Tiefensuche:

- Sie ist vollständig
- hat eine Zeitkomplexität von $O(b^d)$
- hat eine Speicherkomplexität von $O(bd)$
- und ist optimal

2 Breitensuche

Der Beweis soll zeigen, dass die Speicherkomplexität der Breitensuche (breadth-first search) als Oberschranke b^d hat. g(n) bezeichnet den Speicheraufwand des Baumes mit Branching-Faktor b und Tiefe d. (Erklärung: Auf Ebene 1 wird 1 Knoten gespeichert, auf Ebene 2 werden b Knoten gespeichert auf Ebene d werden b^d Knoten gespeichert (sic!). Und die Breitensuche speichert alle diese Knoten, also die Summe dieser Einzelterme.

$$g(n) = O(b^d)$$

$$1 + b^1 + b^2 + b^3 + ... + b^d = c \cdot b^d$$

Nach durchdividieren der linken Seite durch b^d und umordnen erhält man:

$$b^d \cdot (1 + \frac{1}{b} + \frac{1}{b^2} + \frac{1}{b^3} + ... + \frac{1}{b^d}) = c \cdot b^d$$

Kürzen

$$(1 + \frac{1}{b} + \frac{1}{b^2} + \frac{1}{b^3} + ... + \frac{1}{b^d}) = c$$

Jetzt als Substitution $x = \frac{1}{b}$ verwenden.

$$I : (1 + x + x^2 + x^3 + ... + x^d) = c$$

$$II : (x + x^2 + x^3 + ... + x^{d+1}) = c \cdot x$$

Hier jetzt eine Subtraktion der obigen zwei Gleichungen durchführen:

$$c - c \cdot x = 1 - x^{d+1}$$

wobei der letzte Term vernachlässigt werden kann, da dieser für großes d gegen 0 konvergiert.

$$c(1 - x) = 1$$

$$c = \frac{1}{1 - x} = \frac{b}{b - 1}$$

Zuletzt muss noch ein geeignetes c gewählt werden, sodass folgendes gilt:

$$b^d \frac{b}{b - 1} \leq c \cdot b^d$$

Der Wert c=2 erfüllt diese Bedingung, und damit ist der Beweis abgeschlossen. Korrekter Weise müsste man noch ein n_0 angeben. Anmerkung: Im Buch von Russel und Norvig findet sich $O(b^{d+1})$, da diese andere Annahmen bezüglich der Anzahl der Knoten getroffen haben.

3 Iterativ vertiefte Tiefensuche

Der Beweis soll zeigen, dass die Laufzeitkomplexität der Iterativ vertieften Tiefensuche (iterative deepening depth-first search) als Oberschranke b^d hat. Hier werden Knoten teilweise häufiger besucht, da in jedem Iterationsschritt wieder von vorne begonnen wird (quasi als ob man einen neuen Suchbaum hätte). Allgemein gilt

$$n_{dfid}(d) = 1 + bfs(1) + bfs(2) + \ldots + bfs(d)$$

Der erste Knoten wird $(d+1)\cdot b^0$ Mal besucht. Der zweite Knoten wird $(d)\cdot b^1$ Mal besucht, usw. Also:

$$n_{dfid}(d) = (d+1)b^0 + (d)b^1 + (d-1)b^2 + (d-2)b^3 + \ldots + \underbrace{(d-d+1)b^d}_{b^d}$$

Durchdividieren und umordnen:

$$b^d \left[1 + \frac{2}{b} + \frac{3}{b^2} + \frac{4}{b^3} + \ldots + \frac{d+1}{b^d}\right]$$

wir verwenden wieder die Substitution $x = \frac{1}{b}$ und schreiben nun unsere Behauptung an:

$$b^d \left[1 + 2x + 3x^2 + 4x^3 + \ldots + (d+1)x^d\right] = c \cdot b^d$$

Nun verwenden wir wieder den selben Trick wie bei der Breitensuche

$$I : c = 1 + 2x + 3x^2 + 4x^3 + \ldots + (d+1)x^d$$

$$II : c \cdot x = x + 2x^2 + 3x^3 + 4x^4 + \ldots + (d+1)x^{d+1}$$

Und wir subtrahieren wieder die beiden Gleichungen und erhalten (Anm: auch hier wurde der letzte Term der Einfachheit halber wieder weggelassen)

$$c(1-x) = 1 + x + x^2 + x^3 + \ldots$$

Und für den rechten Teil wissen wir bereits was das Ergebnis ist, also

$$c(1-x) = \frac{1}{1-x}$$

$$c = \frac{1}{(1-\frac{1}{b})^2}$$

Wiederum wählen wir einen Wert für c, der unsere Bedingung erfüllt, also für $b \geq 2$ wählen wir c=4 und sind fertig. Abschließende Anmerkung, wenn man die beiden c's vergleicht, so kann man in etwa davon ausgehen, dass Iterativ vertiefte Tiefensuche doppelt so lange dauert wie Breitensuche, allerdings von der Speicherkomplexität einen deutlichen Vorteil bringt!

4 Konsistente Heuristiken

Zu zeigen ist, dass eine konsistente Heuristik zulässig ist. Wobei hier die Begriffe noch genauer zu spezifizieren wären. Eine zulässige Heuristik h(n) ist eine optimistische Schätzung der Kosten, wobei die wahren Kosten nie überschätzt werden dürfen.

Die Kosten der optimalen Lösung bezeichnen wir mit C*. g(n) sind die Kosten zur Erreichung eines Knotens (also die Summe aller Wegkosten, bis zu einem bestimmten Knoten). h(n) sind die Kosten um von dem Knoten zum Ziel zu gelangen. Diese Werte können z.B. die Luftliniendistanz (also die euklidische Distanz) bei einem Streckenproblem sein, da diese auf keinen Fall unterschritten werden können.

Beispielsweise die Greedy-Best-First-Strategie verwendet als Heuristikfunktion $f(n) = h(n)$. Wohingegen die A*-Suche nicht nur den „Restweg“, sondern auch die bisherigen Kosten g(n) berücksichtigt, also $f(n) = h(n) + g(n)$.

Konsistenz ist dann gegeben, wenn $h(n) \leq c(n, a, n') + h(n')$ wobei n' ein Nachfolger von dem Knoten n ist. Wobei $c(n, a, n')$ die Übergangskosten sind, die anfallen, wenn man die Aktion a auf den Zustand n anwendet um zu n' zu gelangen.

Eine zulässige Heuristikfunktion liegt dann vor, falls folgende drei Bedingungen erfüllt sind:

1. $\forall Knoten\ n: \underbrace{h(n)}_{geschaetzte\ Kosten} \leq \underbrace{h^*(n)}_{reale\ Kosten}$ sprich, wenn die Schätzung optimistisch ist.
2. $h(n) \geq 0$
3. $h(Zielknoten) = 0$ (für alle möglichen Ziele)

Bezeichnet $S_1, S_2, S_3, ...$ die Knoten und $a_1, a_2, ...$ die Kosten um von Knoten n zu Knoten n+1 zu kommen, so gilt aufgrund der Konsistenz folgende Beziehung für alle Knoten (erhält man durch ein einfaches Umformen der Konsistenz):

$$h(S_1) - h(S_2) \leq c(S_1, a_1, S_2)$$
$$h(S_2) - h(S_3) \leq c(S_2, a_2, S_3)$$
$$h(S_3) - h(S_4) \leq c(S_3, a_3, S_4)$$
$$\dots$$
$$h(S_{g-1}) - h(S_g) \leq c(S_{g-1}, a_{g-1}, S_g)$$

Bildet man nun eine Summe, erhält man

$$h(S_1) - h(S_g) \leq h^*(S_1) = c(S_1, S_g)$$

und nachdem ein Zielknoten die Kosten 0 hat, ist unser Beweis fertig und es gilt:

$$h(S_1) \leq h^*(S_1)$$

Anmerkung: Wir haben diesen Beweis jetzt nur für einen Knoten S_1 gezeigt, und nicht für alle. Es gilt natürlich analoger Beweis für alle Knoten.