Der A^*-Algorithmus

Der A^*-Algorithmus löst ein Suchproblem unter Ausnutzung einer Heuristik. Ein Suchproblem ist definiert durch

- eine Menge Z von Zuständen,
- eine (endliche) Menge O von Operationen, die einen Zustand in einen anderen überführen,
- einen Startzustand $z_0 \in Z$ und
- eine Menge $E \subseteq Z$ von Endzuständen.

Die Mengen Z und O legen den Suchraum in Form eines Zustandsgraphen fest: Die Zustände bilden die Knoten, die Operationen die Kanten des Graphen. Dieser Zustandsgraph ist i.a. nur implizit durch den Startzustand und die Operationenmenge gegeben. Gesucht ist eine Folge von Operationen, die den Startzustand in einen Zielzustand überführt und die minimale Kosten verursacht. Durch die Heuristik werden die Kosten einer Operationenfolge zum Erreichen eines Zielzustandes geschätzt. Mit ihr wird gesteuert, wie der Zustandsgraph durchlaufen wird. Die folgende Darstellung des Algorithmus weicht leicht von der in der Vorlesung angegebenen ab.

Gegeben

- Der Startzustand z_0. (Wie ein Zustand repräsentiert wird, ist für den A^*-Algorithmus selbst nicht relevant, sondern nur für die Implementierung der im folgenden aufgeführten Operationen und Funktionen.)
- Eine Menge $O = \{o_1, \ldots, o_n\}$ von Operationen, die zu einem gegebenen Zustand einen Nachfolgerzustand liefern. (I.a. sind nicht alle Operationen auf alle Zustände anwendbar. Wir nehmen daher an, daß eine Operation den speziellen Wert $\perp$ (undefiniert) statt eines neuen Zustandes liefert, wenn sie nicht anwendbar ist.)
- Eine reellwertige Funktion costs, die für jede Operation in O die ihr zugeordneten Kosten liefert. (U.U. hängen die Kosten einer Operation davon ab, auf welchen Zustand sie angewandt wird. In diesem Fall kann die Funktion costs auch zweistellig sein.)
- Eine reellwertige Heuristikfunktion $\hat{h}$, die für einen gegebenen Zustand eine Schätzung der Kosten einer Operationenfolge zum Erreichen des Zielzustandes liefert.
- Eine Funktion goal, mit der sich feststellen läßt, ob ein gegebener Zustand ein Zielzustand ist. (Ist die Menge der Zielzustände explizit gegeben, kann die Funktion goal einfach prüfen, ob der gegebene Zustand in dieser Menge enthalten ist.)

Gesucht

- Eine Folge von Operationen, die den Startzustand in einen Zielzustand überführt und die minimale Kosten verursacht.

Algorithmus

1. Erzeuge einen (gerichteten) Graphen $G = \{V, E\}$, der nur den Startzustand z_0 enthält, d.h. setze $V := \{z_0\}$ und $E := \emptyset$. (Im Graphen G werden der besuchte Teil des Suchraums und die besten bekannten Wege zum Erreichen eines Zustandes dargestellt.)
2. Erzeuge eine Menge `open`, die nur den Startzustand enthält, d.h. setze `open` $:= \{z_0\}$. (Die Menge `open` enthält die erreichten Zustände, deren Nachfolger noch nicht erzeugt wurden.)
3. Erzeuge eine leere Menge `closed`, d.h. setze `closed` $:= \emptyset$. (Die Menge `closed` enthält die erreichten Zustände, deren Nachfolger bereits erzeugt wurden.)

4. Erzeuge eine Abbildung $\hat{g}$ von der Menge der Zustände in die reellen Zahlen, die für den Zustand z_0 den Wert 0 liefert und sonst undefiniert ist. (Die Abbildung $\hat{g}$ ist der sogenannte Tiefenfaktor der Heuristik, der die Kosten der besten gefundenen Operationenfolge zum Erreichen eines Zustandes vom Startzustand aus angibt.)
5. Erzeuge eine Abbildung e von der Menge der Zustände in die Menge der Operationen, die für alle Zustände undefiniert ist. (Die Abbildung e gibt an, durch welche Operation ein Zustand von seinem Vorgängerzustand aus erreicht wird. Mit ihr wird schließlich die Lösung des Problems aufgebaut, die ja eine Operationenfolge ist.)
6. Wähle einen Zustand z aus der Menge `open`, für den die Funktion $\hat{f} = \hat{g} + \hat{h}$ den kleinsten Wert unter allen Elementen der Menge `open` liefert, d.h. wähle ein $z \in \{x \mid \hat{f}(x) = \min_{y \in \texttt{open}} \hat{f}(y)\}$. (Wähle also einen der Zustände, die gemäß der Heuristik am erfolgversprechendsten sind.)
7. Falls goal(z) gilt, d.h. falls z ein Zielzustand ist, ist eine Lösung des Problem gefunden. Der Algorithmus wird in diesem Fall mit Erfolg abgebrochen. Die Lösung wird aus dem erzeugten Graphen G folgendermaßen in umgekehrter Reihenfolge abgelesen:
   (a) Erzeuge eine leere Folge L von Operationen, d.h. setze $L := ()$.
   (b) Falls $z = z_0$ (also falls der Startzustand erreicht ist), so ist L die gesuchte Folge von Operationen, gib also L zurück.
   (c) Füge die Operation $e(z)$, durch die der Zustand z erreicht wird, am Anfang der Folge L hinzu, d.h. mit $L = (l_1, \ldots, l_k)$ setze $L := (e(z), l_1, \ldots, l_k)$.
   (d) Fahre mit dem direkten Vorgänger von z fort, d.h. setze $z := x$ mit $(x, z) \in E$ (man beachte, daß der Vorgänger von z in G eindeutig bestimmt ist), und gehe zurück zu Schritt b.
8. Entferne den Zustand z aus der Menge `open`, d.h. setze $\texttt{open} := \texttt{open} \setminus \{z\}$. (Die Nachfolger des Zustand z werden im folgenden Schritt erzeugt.)
9. Für alle $o \in O$:
   (a) Setze $x := o(z)$ und $c := \hat{g}(z) + \text{costs}(o)$.
   (b) Falls $x = \perp$ (also falls die Operation o nicht auf den Zustand z anwendbar ist), übergehe die folgenden Schritte und fahre mit der nächsten Operation o fort.
   (c) Falls $x \notin \texttt{open} \cup \texttt{closed}$ (also falls x ein neuer, bisher nicht erreichter Zustand ist), setze $\texttt{open} := \texttt{open} \cup \{x\}$ (der Zustand x wurde erreicht, aber seine Nachfolger wurden noch nicht erzeugt), $e(x) := o$ (der Zustand x wurde durch Anwendung der Operation o von seinem Vorgänger z aus erreicht), und $\hat{g}(x) := c$ (die Operationenfolge zum Erreichen von x verursacht die Kosten c). Erweitere außerdem den Graphen G um den Knoten x und die Kante (z, x), d.h. setze $V := V \cup \{x\}$ und $E := E \cup \{(z, x)\}$.
   (d) Falls $x \in \texttt{open} \cup \texttt{closed}$ (also falls der Zustand x schon auf einem anderem Wege erreicht wurde) und $c < \hat{g}(x)$ (also falls die Kosten der Operationenfolge, durch die der Zustand x über den Zustand z erreicht wird, geringer sind als die der Folge über die x vorher erreicht wurde), setze $\hat{g}(x) := c$ (eine Operationenfolge zum Erreichen von x mit geringeren Kosten ist jetzt bekannt) und $e(x) = o$ (der Zustand x wird günstiger durch Anwendung der Operation o auf den Zustand z erreicht). Mache außerdem den Zustand z zum neuen Vorgänger von x in G, d.h. setze $E := (E \setminus \{(a, b) \mid b = x\}) \cup \{(z, x)\}$.

      Falls $x \in \texttt{closed}$ (also falls x bereits Nachfolger besitzt), prüfe alle Zustände, die sich durch Anwendung von Operationen aus O von x aus erreichen lassen (nicht nur die Nachfolger von x im Graphen G!). Für jeden Zustand, der sich nun über den Zustand x günstiger erreichen läßt, ist der Tiefenfaktor $\hat{g}$ anzupassen und die Kante von seinem bisherigen Vorgänger durch eine Kante von dem Zustand x aus zu ersetzen. Für alle so geänderten Zustände, die selbst in der Menge `closed` enthalten sind, sind diese Anpassungen rekursiv auszuführen, d.h. die Zustände, die sich von ihnen aus durch Operationen in O erreichen lassen, sind zu prüfen und ggf. anzupassen usf.

10. Füge den Zustand z der Menge `closed` hinzu, d.h. setze `closed` := `closed`$\cup\{z\}$. (Die Nachfolger des Zustands z wurden im vorhergehenden Schritt erzeugt.)
11. Falls die Menge `open` leer ist, so ist das Problem unlösbar. Der Algorithmus wird daher ohne Erfolg abgebrochen. Anderenfalls gehe zu Schritt 6.

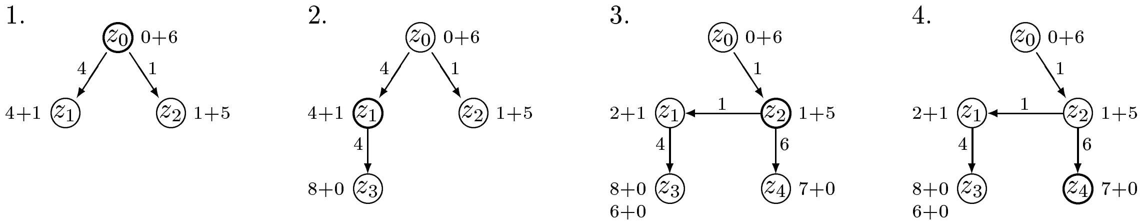
Bemerkungen

Man kann beweisen, daß der A^*-Algorithmus korrekt ist (d.h. sicher die Operationenfolge mit den geringstmöglichen Kosten zum Erreichen eines Zielzustandes liefert), wenn folgende Bedingungen erfüllt sind:

1. Jeder Zustand hat nur endlich viele Nachfolgezustände. (Diese Bedingung ist in der obigen Formulierung des A^*-Algorithmus durch die Forderung einer *endlichen* Menge von Operationen ausgedrückt.)
2. Die Kosten aller Operationen sind größer als ein bestimmter positiver Wert ϵ.
3. Die Heuristikfunktion $\hat{h}$ überschätzt für *keinen* Zustand z die Kosten einer Operationenfolge zum Erreichen eines Zielzustandes (eine solche Funktion $\hat{h}$ heißt auch *optimistischer Schätzer*).

In Schritt 9(d) werden u.U. mehrere Kanten des Graphen G geändert und der Tiefenfaktor $\hat{g}$ für mehrere Zustände angepaßt. Diese Änderungen verringern zwar den Aufwand der folgenden Suche, können jedoch selbst sehr aufwendig sein (im schlechtesten Fall exponentiell, da ggf. rekursiv alle Zustände, die sich durch Operationen von einem geänderten Zustand aus erreichen lassen, bearbeitet werden müssen). Diese Anpassungen werden daher oft nicht implementiert. Leider kann man jedoch von dem so vereinfachten Algorithmus nicht mehr zeigen, daß er korrekt ist. Das sei an zwei Beispielen erläutert.

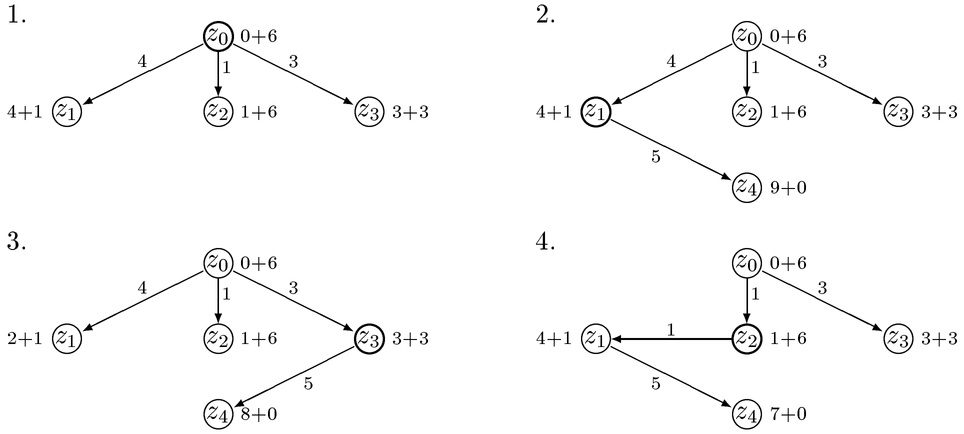
Warum es notwendig ist, den Tiefenfaktor $\hat{g}$ auch für die Nachfolger eines Zustandes zu aktualisieren, wenn der Tiefenfaktor dieses Zustandes geändert wird, zeigt das folgende Beispiel.



In der obigen Abbildung sind vier Schritte des Aufbaus des Graphen G für ein nicht näher spezifiziertes Suchproblem durch den A^*-Algorithmus gezeigt, ausgehend vom Startzustand z_0. Die Zustände z_3 und z_4 seien Zielzustände. Der in dem jeweiligen Schritt erweiterte Zustand ist durch einen fettgedruckten Kreis markiert. Die Zahlen an den Kanten geben die Kosten der zugehörigen Operationen an, die Summen neben den Zuständen die Werte der Heuristikfunktion $\hat{f} = \hat{g} + \hat{h}$.

Im Schritt 3 wird durch die Erweiterung des Zustandes z_2 eine günstigere Operationenfolge zum Erreichen des Zustandes z_1 gefunden, wodurch sich der Tiefenfaktor $\hat{g}$ für den Zustand z_1 von 4 auf 2 ändert. Wird nun nicht auch die Tiefenfaktor des Nachfolgerzustandes z_3 entsprechend von 8 auf 6 geändert, so wird nicht die beste Lösung gefunden. Denn mit einem Tiefenfaktor von 8 für den Zustand z_3 wird der Zustand z_4 für die nächste Erweiterung vorgezogen. Da Zustand z_4 ein Zielzustand ist, wird die Suche mit dem suboptimalen Ergebnis $z_0 \xrightarrow{1} z_2 \xrightarrow{6} z_4$, also einer Operationenfolge mit den Gesamtkosten 7, beendet. Wird dagegen im Schritt 3 auch der Tiefenfaktor des Zustandes z_3 korrekt auf 6 geändert, so wird im Schritt 4 statt des Zustandes z_4 der Zustand z_3 erweitert und damit die beste Lösung $z_0 \xrightarrow{1} z_2 \xrightarrow{1} z_1 \xrightarrow{4} z_3$, also eine Operationenfolge mit den Gesamtkosten 6, gefunden. Man beachte, daß in diesem Beispiel die Zustände z_3 und z_4 auch identisch sein können. Dann wird zwar der richtige Endzustand erreicht, aber mit einer suboptimalen Operationenfolge.

Warum es notwendig ist, alle Zustände zu prüfen und gegebenenfalls (rekursiv) zu bearbeiten, die sich durch Operationen von einem Zustand aus erreichen lassen, dessen Tiefenfaktor $\hat{g}$ geändert wurde, auch wenn diese Zustände keine Nachfolger des geänderten Zustands im Graphen G sind, zeigt das folgende Beispiel.



In der obigen Abbildung sind vier Schritte des Aufbaus des Graphen G für ein nicht näher spezifiziertes Suchproblem durch den A^*-Algorithmus gezeigt, ausgehend vom Startzustand z_0. Der Zustand z_4 sei ein Zielzustand. Der in dem jeweiligen Schritt erweiterte Zustand ist wieder durch einen fettgedruckten Kreis markiert. Die Zahlen an den Kanten geben die Kosten der zugehörigen Operationen an, die Summen neben den Zuständen die Werte der Heuristikfunktion $\hat{f} = \hat{g} + \hat{h}$.

Im Schritt 3 wird durch die Erweiterung des Zustands z_3 ein günstigerer Weg zum Knoten z_4 gefunden. Daher wird die Kante $z_1 \rightarrow z_4$ entfernt und stattdessen die Kante $z_3 \rightarrow z_4$ eingefügt. Danach ist der Zustand z_4 kein Nachfolger des Zustands z_1 mehr. Im Schritt 4 wird nun aber ein günstigerer Weg zum Zustand z_1, dem ehemaligen Vorgänger des Zustandes z_4 gefunden, was den Tiefenfaktor des Zustandes z_1 von 4 auf 2 ändert. Damit ist es aber wieder günstiger den Zustand z_4 über den Zustand z_1 statt über den Zustand z_3 zu erreichen. Folglich muß die Kante $z_3 \rightarrow z_4$ entfernt und die Kante $z_1 \rightarrow z_4$ wieder eingefügt werden. Geschieht dies nicht, wird nicht die optimale Lösung $z_0 \xrightarrow{1} z_2 \xrightarrow{1} z_1 \xrightarrow{5} z_4$ mit den Gesamtkosten 7 sondern nur die schlechtere Lösung $z_0 \xrightarrow{3} z_3 \xrightarrow{5} z_4$ mit den Gesamtkosten 8 gefunden.

Statt rekursiv alle Zustände zu prüfen und ggf. anzupassen, die sich von einem Zustand in der Menge `closed` erreichen lassen, dessen Tiefenfaktor geändert wurde, kann man allerdings auch den geänderten Zustand aus der Menge `closed` entfernen und ihn wieder in die Menge `open` einfügen. Dann wird die eventuell notwendige Anpassung anderer Zustände durch das normale Vorgehen des Algorithmus erledigt. Dies ist meist die einfachste Möglichkeit der Implementierung. Außerdem werden auf diese Weise nur die Anpassungen durchgeführt, die notwendig sind, um ein korrektes Ergebnis sicherzustellen.