Der A* - Algorithmus

Beim A* - Algorithmus geht es darum,

- einerseits vor dem Erreichen einer Lösung möglichst wenige Möglichkeiten versucht zu haben
- andererseits aber sicher zu sein, dass man nicht in einem lokalen Nebenmaximum (- Minimum) hängen geblieben ist, sondern erwiesenermaßen eine Lösung gefunden hat.

Dazu berechnet man an allen Knoten des Suchraumes zu allen Möglichkeiten eine Schätzfunktion zu den Kosten. Dabei setzt sich diese aus den – nicht zu schätzenden, da schon bekannten – Kosten zusammen, die bis zu diesem Knoten schon aufgelaufen sind und den geschätzten ***Restkosten*** bis zum Ziel der Suche. Dass eine solche Abschätzung der Restkosten nicht immer möglich ist, ist sicher eine triviale Erkenntnis.

Nicht trivial ist dagegen die Eigenschaft dieser Schätzfunktion, dass sie die Restkosten immer ***optimistisch schätzen*** muss. Optimistisch bedeutet in diesem Zusammenhang, dass sie keinesfalls einen Wert angeben darf, der höher als die tatsächlichen Kosten auf irgend einem der von diesem Knoten ausgehenden Wege sein darf. Das führt in einigen Fällen zu scheinbar unsinnig einfachen Schätzfunktionen, hat aber den Grund, dass wir doch sicher sein wollen, auch wirklich die (eine) beste Lösung gefunden zu haben.

Haben wir einen Weg zum Ziel gefunden, dann ist er sicher die (eine) Lösung, wenn

- seine tatsächlichen Kosten – wir sind ja da und können sie ausrechnen –

kleiner sind als

- die optimistisch geschätzten Gesamtkosten aller anderen auf dem Weg zu diesem Knoten betretenen Zwischenknoten.

Hier ist „optimistisch“ von zentraler Bedeutung! Wenn man von den anderen Knoten zu ***keinem*** besseren Wert kommen ***konnte***, muss die gefundene Lösung optimal sein!

Leider ist es keinesfalls trivial diese Restkostenabschätzungen zu finden. Wir werden uns mit einigen Beispielen beschäftigen.

CLP[1] mit dem A - Algorithmus*

Die Bewertungsfunktion

Die Frage nach der Abwägung zwischen komplex und einfach ist nicht eindeutig zu beantworten. Sicher ist eine einfache Modellierung für den Einstieg sinnvoll, bleibt man aber dabei, dann wird man wesentliche Problemaspekte nicht kennen lernen können. Ein wichtiges Problem ist das Problem der Zyklen, das man aber sinnvollerweise an einem Beispiel wie einem Labyrinth untersucht. Ein weiteres Problem ist aber auch die Komplexität des Systems selbst, die wir uns am Beginn der Betrachtungen zum CLP deutlich gemacht haben. Eine wichtige Folge der Komplexität des CLP ist, dass eine Bewertungsfunktion für Lösungen und Teillösungen keinesfalls trivial zu finden ist.

Dazu betrachten wir – wiederum eingeschränkt – eine lineare Modellierung der Container wie beim Rucksackproblem, berücksichtigen nun aber, dass es verschiedene Bewertungsgrößen gibt, nämlich

- die Länge der Stücke
- das Gewicht der Stücke
- der Wert der Stücke
- die Priorität ihres Transportes

Daraus ergeben sich möglicherweise folgende Restriktionen:

- die Länge der Stücke im Container hat eine Obergrenze
- das Gewicht der Stücke im Container hat eine Obergrenze
- der Wert der Stücke im Container hat eine Obergrenze (z.B. aus Versicherungsgründen)
- die Priorität ihres Transportes (z.B. I – III) legt eine Reihenfolge fest

Wenn wir nun die Güte einer Lösung oder Teillösung beurteilen wollen und sie mit Beurteilungen von anderen Lösungen oder Teillösungen vergleichen wollen, dann brauchen wir eine Bewertungsfunktion. Sie ermöglicht es uns, im Suchraum eine Ordnung zu definieren.

Eine einfache Bewertungsfunktion könnte prüfen, ob eine der Restriktionen verletzt ist oder nicht, allein die Priorität könnte ggf. eine Reihenfolge definieren. Die oben angegebenen Bedingungen geben also bis auf die Priorität nur eine entweder-oder – Entscheidung her. Entweder Lösungen oder Teillösungen erfüllen die Bedingungen oder nicht.

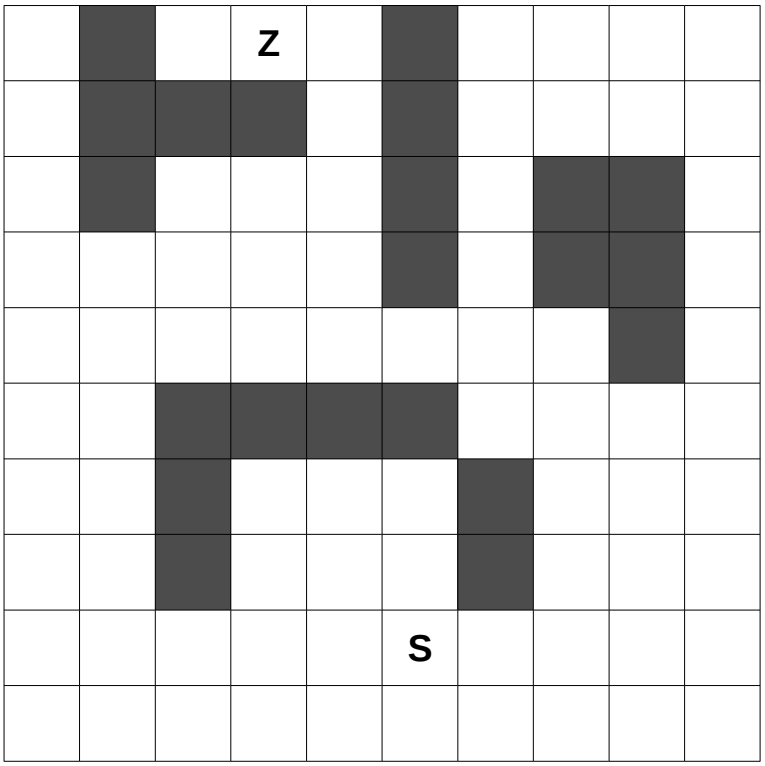
Eine sinnvolle Bewertungsfunktion könnte in dem gewählten Beispiel folgendermaßen aussehen:

- Jedes Stück mit höherer Priorität kommt vor allen anderen
- Jede Verletzung einer Obergrenze führt zu vollständiger Abwertung der Lösung
- Zu jeder der Größen Länge, Gewicht und Wert wird ein relativer Füllungsgrad errechnet, dessen Summenwert die weitere Reihenfolge der Lösungen oder Teillösungen bestimmt.

1 **C**ontainer **L**ade **P**roblem

Der A-Algorithmus in einem anderen Beispiel*

Gesucht ist ein kürzester Weg vom Start zum Ziel. Die Problemumgebung[1] ist schachbrettartig und es wird davon ausgegangen, dass nur horizontale und vertikale Bewegungen möglich sind.



Natürlich ginge es bei dem dargestellten Beispielgraphen mit der Breitensuche: Uniforme Kantenbewertung und ein nicht zu großer Suchraum lassen es annehmen, dass man auf einfache Art eine optimale Lösung in akzeptabler Zeit finden kann. Dieses einfache Beispiel dient daher allein dem Ziel besserer Anschaulichkeit.

Informierte Suchverfahren

Bei diesem Problem erkennt man relativ schnell, dass man mit einem informierten Suchverfahren die Auswahl von Nachfolgeknoten im Suchraum deutlich verbessern kann. Grundlage dafür ist der Abstand des jeweils aktuell erreichten Punktes vom Zielpunkt. Für diesen Abstand kann man natürlich die Luftliniendistanz verwenden, da in diesem Problem aber nur horizontale und vertikale Bewegungen möglich sind, ist es ebenso angemessen – und natürlich jeweils sehr viel einfacher zu berechnen – wenn man statt dessen die Manhattan-Distanz der Felder verwendet.

Zu jedem Ort, den man bei der Suche erreicht hat, lassen sich daher einfach die beiden Werte der aktuell schon aufzuwendenden Kosten, die bisherige Weglänge, und die minimalen, bei Ignorieren der Hindernisse notwendigen Restkosten berechnen.

Aufgabe

Berechnen Sie die geschätzten Gesamtkosten für einige Beispiele.

1 Vorlage von Volker Turau, Algorithmische Graphentheorie

Wenn man die geschätzten Gesamtkosten für einige Beispiele berechnet hat, sollte man erkennen, dass man mit ihrer Hilfe eine Auswahl unter den weiteren Möglichkeiten im Suchraum treffen kann: Verwendet man immer den (*einen der*) nach dem Schätzwert der Gesamtkosten günstigsten Nachfolger weiter, wird man sich erst nachrangig mit den schlechteren Wegen beschäftigen, nämlich dann, wenn man mit den zunächst günstig erscheinenden Wegen in einen Bereich mit höheren geschätzten Gesamtkosten gerät.

Paperwork

Führen Sie diese Suche beispielhaft ohne Computerunterstützung durch. Ziel ist es dabei, sich die notwendigen Datenstrukturen und den Algorithmus zu erarbeiten.

Ergebnisse

Datenstrukturen:

- Felder lassen sich mit Listen von zwei Koordinaten beschreiben, also z.B. (3 4)
- Die Hindernisse werden in einer Liste von Feldern abgelegt, alternativ in einer Assoziationsliste von Zeilennummer mit den Spaltennummern der Hindernisfelder.
- Die Wege werden in einer Prioritätswarteschlange gehalten, um den Zugriff auf die aktuell besten Knoten des Suchraums zu erleichtern.
  [Ob die geschätzten Gesamtkosten dabei in die Wegangaben mit übernommen werden oder jeweils neu berechnet werden, ist allein eine Entwurfsentscheidung.]

Algorithmus

Das Ziel liegt in unserem Beispielfeld bei (4 1). Man beginnt mit einem Weg, der allein den Startort, hier also (6 9) enthält. Seine Schätzkosten bestehen aus Weglänge + x-Distanz zum Ziel + y-Distanz zum Ziel, also

aktuelle Kosten	**+ Restkosten**	**=**	**Gesamtkosten**
0	**+ 2 + 8**	**=**	**10**

Die möglichen Nachfolgeorte sind die Orte (8 6) (10 6) (9 5) (9 7). Dabei erkennt man sehr leicht den Algorithmus zur Berechnung der Nachbarfelder.

Ein erstes Filtern auf die mit Hindernissen belegten Felder hin führt hier nicht zum Ausschluss von Nachfolgern, so dass nun zu den vier Folgefeldern die Wege gebildet und die geschätzten Gesamtkosten der Wege berechnet werden müssen.

- ((6 8) (6 9)) → 1 + 2 + 7 = 10
- ((6 10) (6 9)) → 1 + 2 + 9 = 12
- ((5 9) (6 9)) → 1 + 1 + 8 = 10
- ((7 9) (6 9)) → 1 + 3 + 8 = 12

Das Ergebnis ist bei einem Blick auf das Feld für uns nicht überraschend. Die Warteschlange könnte tabellarisch[1] dargestellt werden, sie muss den ursprünglichen Start nicht mehr enthalten, da er vollständig entwickelt wurde.

10 → ((6 8) (6 9)) ((5 9) (6 9))
12 → ((6 10) (6 9)) ((7 9) (6 9))

1 Eine passende Datenstruktur könnte also auch eine Assoziationsliste sein.

Entscheidungen zur Art des Algorithmus

Wir müssen uns entscheiden, wie stark wir nun die Knoten weiter entwickeln. Eine Möglichkeit ist, jeweils den ersten Knoten des Suchraumes zu entnehmen (er ist dann also nicht mehr enthalten), diesen vollständig zu expandieren und die entstehenden neuen Wege in die Prioritätswarteschlange einzufügen. Das hat den Vorteil, dass man sich mit diesem Knoten dann nicht mehr beschäftigen muss. Eine Alternative wäre, nur einen der Wege weiter zu verfolgen, die nicht zu höheren geschätzten Gesamtkosten führen. Dann hat man das Problem, dass man den bisherigen Weg erst entfernen kann, wenn er später weiter voll expandiert wurde.

Die erste mögliche Alternative hätte auch schon am Beginn verwendet werden können. Gehen wir einmal danach vor, wäre also eine Möglichkeit folgender Ablauf:

((10 ((6 9))) in der Richtung weiter
((10 ((6 8) (6 9)) ((6 9)))
((10 ((6 7) (6 8) (6 9)) ((6 8) (6 9)) (6 9)))

Beim nächsten Schritt trifft man bei gleicher Richtung auf eine Hindernis: (6 6). Da man nun nicht einfach weiter kann, muss man vom bisherigen Weg abweichen. Rechts, bei (7 7) finden wir ein Hindernis, zurück wollen wir nicht (verhindern!), also bleibt nur das Feld (5 7), das zu keinem schlechteren Weg führt.

((10 ((5 7) (6 7) (6 8) (6 9)) ((6 7) (6 8) (6 9)) ((6 8) (6 9)) ((6 9)))

Weiter in derselben Richtung:

((10 ((4 7) (5 7) (6 7) (6 8) (6 9)) ((5 7) (6 7) (6 8) (6 9)) ((6 7) (6 8) (6 9))
((6 8) (6 9)) ((6 9)))

Hier taucht wieder ein Hindernis auf. Man merkt, dass der Ablauf nun langsam unübersichtlich wird. Die zweite Alternative scheint also verständlicher zu sein.

Die zweite Alternative

Hier entwickelt man jeden Knoten voll und entfernt ihn aus dem Suchgraphen. Das führt zu folgendem Algorithmus:

Entferne die (die erste der) im Moment am günstigsten geschätzte Kante und füge die entwickelten Nachfolger in den Suchgraphen ein.

Die Datenstruktur nach dem ersten Schritt:

(10 → ((6 8) (6 9)) ((5 9) (6 9))
12 → ((610) (6 9)) ((79) (6 9)))

Zweiter Schritt:

(10 → ((5 8) (6 8) (6 9)) ((6 7) (6 8) (6 9)) ((5 9) (6 9))
12 → ((6 10) (6 9)) ((7 9) (6 9)))

usw.

Wichtig ist, zu beachten, dass keine aktuellen Orte am Kopf einer Liste mit höheren Kosten auftauchen, die schon in einer Liste mit geringeren Kosten vorhanden sind. Diese in der Prioritätswarteschlange nachfolgenden Wege müssen also entfernt werden.