Informatik
Hauptcampus

HOCHSCHULE TRIER

Funktionsprinzipien und Anwendungen von Algorithmen zur Pfadplanung

Bearbeiter 1: Mohammed Salih Mezraoui
Bearbeiter 2: David Gruber
Bearbeiter 3: Marius Müller

Gruppe: WissArb22/Thema 2/Gruppe-7

Ausarbeitung zur Vorlesung Wissenschaftliches Arbeiten

Trier, 15.07.2022

Kurzfassung

In dieser wissenschaftlichen Arbeit werden die wichtigsten Algorithmen zur Pfadplanung dargestellt und analysiert. Es wird anhand der Anwendung in Geoinformationssystemen und bei mobilen Robotern aufgezeigt, wie diese Algorithmen eingesetzt werden und welche Stärken und Schwächen damit einhergehen. Da die Fragestellungen, die mit Algorithmen zur Pfadplanung bearbeitet werden, immer komplexer werden, steigen auch die Anforderungen an Zeit- und Platzkomplexität. Daher werden in dieser Arbeit die wichtigsten Optimierungsstrategien für Algorithmen zur Pfadplanung dargestellt. In einer experimentellen Analyse konnte gezeigt werden, dass die Laufzeit des Algorithmus von Dijkstra durch den Einsatz von Preprocessing und heuristischer Funktionen um mehrere Größenordnungen gesenkt werden kann.

In this scientific work, the most important algorithms for path planning are presented and analysed. The application in geoinformation systems and mobile robots shows how these algorithms are used and what strengths and weaknesses are associated with them. As the issues addressed by path planning algorithms are becoming increasingly complex, the demands on time and space complexity are also increasing. Therefore, this paper presents the most important optimization strategies for path planning algorithms. An experimental analysis showed that the runtime of the Dijkstra algorithm can be reduced by several orders of magnitude by using preprocessing and heuristic functions.

Inhaltsverzeichnis

1

Einleitung und Problemstellung

2

Algorithmen zur Pfadplanung

3

Optimierungsstrategien

In diesem Kapitel wird beschrieben, wie bisher vorgestellten Algorithmen zur Pfadplanung optimiert werden können.

3.1 Bidirektionale Suche

Bei der bidirektionalen Suche wird ein Suchalgorithmus simultan aus zwei Richtungen laufen gelassen – vom Startknoten zum Zielknoten und umgekehrt. Der Suchalgorithmus wird bei diesem Vorgehen so modifiziert, dass die Abbruchbedingung dann eintritt, wenn beide Suchen denselben Knoten expandieren. Somit betrachtet jede der beiden Suchen nur die Hälfte des Graphen, was in einer Reduktion der Zeitkomplexität resultiert [RN10b]. Es ist jedoch zu beachten, dass die Platzkomplexität stark ansteigt, da in beiden Suchen eigene Priority-Queues verwaltet werden müssen. Außerdem ist es für viele Problemstellungen keinesfalls trivial, eine Suche rückwärts durchzuführen, da eine Methode zur Berechnung des Vorgängers eines Knotens gegeben sein muss [RN10b].

Eine Implementierung der Bidirektionalen Suche ist der Bidirektionale Dijkstra Algorithmus von Vaira und Kurasova. [7] Hier wird der in Abschnitt **??** vorgestellte Dijkstra Algorithmus so modifiziert, dass eine bidirektionale Suche von zwei Prozessen auf einem Mehrkernprozessor parallelisiert durchgeführt wird. In mehreren experimentellen Analysen konnte gezeigt werden, dass die Laufzeit des parallelen bidirektionalen Dijkstra Algorithmus bis zu drei Mal kleiner ist als die des Standard Dijkstra Algorithmus.

3.2 Informed Search

Ein Ansatz, um effizienter Lösungen für das Shortest Path Problem zu finden, ist die Informed Search Strategie, bei der problemspezifisches Wissen, das über die Definition des Problems hinausgeht, bei der Lösungsfindung berücksichtigt wird. Der nächste zu expandierende Knoten auf dem Pfad zum Zielknoten wird auf Basis einer Bewertungsfunktion $f(n)$ ausgewählt. Eine Komponente dieser Bewertungsfunktion ist eine heuristische Funktion $h(n)$, die die zu erwartenden Kosten des optimalen Pfades vom Knoten n zum Zielknoten berechnet [RN10a]. Im Falle des

Straßennetzes könnte hierzu die Länge der Luftlinie zwischen dem Knoten n und dem Zielknoten verwendet werden [HNR68].

Die einfachste Umsetzung dieser Strategie ist, nur die heuristische Funktion bei der Bewertung von Knoten heranzuziehen, sodass $f(n) = h(n)$ gilt. Dieses Vorgehen wird auch Greedy Best-First Suche genannt, da in jedem Schritt versucht wird, so nahe wie möglich an den Zielknoten zu gelangen [RN10a]. Auf diese Weise werden die Suchkosten, also die Anzahl der expandierten Knoten zwar minimiert, es kann jedoch nicht garantiert werden, dass die gefundene Lösung optimal ist [HKH17].

Eine elaboriertere Umsetzung der Informed Search Strategie ist der A* Algorithmus zur Berechnung des kürzesten Pfades zwischen zwei Knoten [RN10a]. A* basiert auf Dijkstra's Algorithmus und erweitert diesen um eine heuristische Funktion, um die Laufzeit zu reduzieren [PAG+20]. Die Bewertungsfunktion $f(n)$ für den A*-Algorithmus setzt sich zusammen aus den Kosten des optimalen Pfades vom Startknoten bis zum Knoten n, $g(n)$ und einer heuristischen Funktion $h(n)$, sodass gilt:

$$f(n) = g(n) + h(n) \tag{3.1}$$

Da verschiedene Heuristiken zur Konstruktion von h(n) gewählt werden können, stellt A* streng genommen eine Familie von Algorithmen dar, wobei die Wahl einer Funktion h(n) einen spezifischen Algorithmus der Familie selektiert [HNR68].

Hart, Nilsson und Raphael, die in [HNR68] die A*-Suche 1968 zum ersten Mal beschrieben haben, konnten nachweisen, dass A* vollständig und optimal ist, wenn die gewählte Heuristik zulässig und konsistent ist. Das heißt, dass unter den angegebenen Voraussetzungen für die Heuristik, immer ein Pfad vom Start- zum Zielknoten gefunden wird (sofern dieser existiert) und dass dieser Pfad in jedem Fall optimal ist. Des Weiteren konnte gezeigt werden, dass A* optimal effizient ist – es kann also keinen anderen optimalen Algorithmus geben, der garantiert weniger Knoten expandiert als A* [RN10a].

3.3 Preprocessing

Eine weiter Optimierungsstrategie ist das Preprocessing, also die Vorverarbeitung des Graphen. Dabei wird häufig gefordert, dass die Platzkomplexität der vorverarbeiteten Daten linear in der Größe des zu bearbeitenden Graphen ist, da man in der Realität oft mit sehr großen Graphen arbeitet [GH05].

3.3.1 ALT-Algorithmen

Eine Familie von Algorithmen, die auf Preprocessing basieren, sind die von Goldberg und Harrleson in [GH05] vorgestellten ALT-Algorithmen. ALT ist ein Akronym für A* search, *Landmarks* und *Triangle Inequality*[1].

In der Preprocessing-Phase des Algorithmus, wird eine kleine (konstante) Anzahl von Landmarken im Graphen ausgewählt, von denen aus dem kürzesten Pfad

[1] Dreiecksungleichung

zu allen anderen Knoten bestimmt wird. Die so bestimmte Distanz wird in Kombination mit der Dreiecksungleich als Heuristik für die Bewertungsfunktion der A* Suche verwendet. Die Dreiecksungleich besagt in diesem Fall, dass die Distanz zwischen einem Knoten n und einem Zielknoten s in jedem Fall größer oder gleich der Differenz der Distanz zwischen n und einer Landmarke l und der Distanz zwischen s und der Landmarke l ist.

$$dist(n, s) \geq dist(l, n) - dist(l, s) \tag{3.2}$$

Somit stellt diese Differenz eine untere Schranke für die Distanz zwischen n und s dar und kann somit als Heuristik verwendet werden [GH05].

3.3.2 Reach-Based Pruning

Reach-Based Pruning [2] ist eine von Gutman in [Gut04] vorgestellte Preprocessing-Strategie zur Vereinfachung von Graphen. Der *Reach* r ist hierbei eine Metrik für einen Graphen G, die als Modifikation für den Dijkstra Algorithmus verwendet werden kann.

Sei P ein Pfad in G von einem Startknoten s zu einem Zielknoten t und v ein Knoten auf diesem Pfad P. Sei zudem $dist(v, w, P)$ die Distanz zwischen den Knoten v und w auf dem Pfad P. Dann ist

$$r(v, P) = \min\left(dist(s, v, P), dist(v, t, P)\right) \tag{3.3}$$

der *Reach* von v auf P. Zudem ist der *Reach* von v in G, $r(v, G)$ definiert als das Maximum aller $r(v, Q)$ für alle kürzesten Pfade Q in G.

Gutman konnte beweisen, dass ein Knoten v nur dann vom Dijkstra Algorithmus betrachtet werden muss, wenn

$$r(v, G) \geq \underline{dist(s, v)} \quad \vee \quad r(v, G) \geq \underline{dist(v, t)} \tag{3.4}$$

gilt, wobei $\underline{dist(v, w)}$ eine untere Schranke für die Distanz zwischen zwei Knoten v und w darstellt.

Die einfachste Möglichkeit die *Reaches* aller Knoten zu bestimmen, ist alle kürzesten Pfade eines Graphen zu bestimmen und die Bedingung 3.4 anzuwenden. Effizientere Vorgehen wurden in [Gol07] und [Gut04] beschrieben.

[2] englischer Ausdruck für das Beschneiden von Bäumen. In der Informatik wird *Pruning* oft als Ausdruck für das Vereinfachen von Graphen verwendet [Wik].

3.4 Experimentelle Analyse

Um die Auswirkung der hier vorgestellten Optimierungsstrategien zu veranschaulichen, wurde von Goldberg in [Gol07] die Laufzeit der folgenden Algorithmen verglichen:

- ***B***: Bidirektionaler Dijkstra Algorithmus
- ***ALT***: Algorithmus aus der ALT-Familie
- ***RE***: Implementierung der Reach-Based Pruning Strategie
- ***REAL***: Algorithmus mit zwei Preprocessing-Stufen (ALT und RE)

Als Input wurde das Straßennetz der San Francisco Bay Area verwendet und jeder dieser Algorithmen wurde auf 10.000 zufällig gewählten Paaren von Knoten angewendet. Dabei wurden die in Tabelle 3.1 dargstellten Messwerte[3] bestimmt.

Algorithmus	**Laufzeit Preprocessing**	**Query Laufzeit**
B	-	30,49 ms
ALT	5,7 s	2,91 ms
RE	45,4 s	0,55 ms
REAL	51,1 s	0,28 ms

Tabelle 3.1: Messung der durchschnittlichen Laufzeit der Preprocessing- und Query-Phase der Algorithmen *B*, *ALT*, *RE* und *REAL* auf dem Straßennetz der San Francisco Bay Area

Man kann erkennen, dass die Laufzeit der Algorithmen mit Preprocessing-Phase (*ALT*, *RE*, *REAL*) signifikant besser ist, als die der Algorithmen ohne Preprocessing-Phase (*B*). Es ist jedoch zu beachten, dass die Laufzeit der Preprocessing-Phase um einige Größenordnungen höher ist, als die Laufzeit des eigentlichen Query-Algorithmus.

[3] Die Laufzeitmessungen wurden auf einem Toshiba Tecra 5 Laptop mit 2GB RAM und Dual-Core 2 GHz Processor durchgeführt

4

Anwendungen

5

Zusammenfassung und Ausblick

Literaturverzeichnis

GH05. Goldberg, Andrew V. und Chris Harrelson: *Computing the shortest path: A search meets graph theory.* SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, Seiten 156–165, Januar 2005.

Gol07. Goldberg, Andrew V.: *Point-to-Point Shortest Path Algorithms with Preprocessing.* In: *Lecture Notes in Computer Science*, Seiten 88–102. Springer Berlin Heidelberg, 2007.

Gut04. Gutman, Ron: *Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks.* In: *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics*, New Orleans, LA, USA, Januar 2004.

HKH17. Heusner, M., T. Keller und M. Helmert: *Understanding the Search Behaviour of Greedy Best-First Search.* Tenth Annual Symposium on Combinatorial Search, 8(1), 2017.

HNR68. Hart, Peter, Nils Nilsson und Bertram Raphael: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.

PAG+20. Peralta, Federico, Mario Arzamendia, Derlis Gregor, Daniel G. Reina und Sergio Toral: *A Comparison of Local Path Planning Techniques of Autonomous Surface Vehicles for Monitoring Applications: The Ypacarai Lake Case-study.* Sensors, 20(5):1488, mar 2020.

RN10a. Russel, Stuart J. und Peter Norvig: *Artificial Intelligence - A Modern Approach*, Kapitel 3.5, Seiten 92–102. Pearson Education, Inc, 2010.

RN10b. Russell, Stuart J. und Peter Norvig: *Artificial Intelligence - A Modern Approach*, Kapitel 3.4.6, Seiten 90,91. Pearson Education, Inc, 2010.

Wik. *Wikipedia - Online Lexikon.* `de.wikipedia.org/wiki/Pruning`.

Arbeitsverteilung

Teilnehmer 1: Mohammed Salih Mezraoui
Inhalte:

Teilnehmer 2: David Gruber
Inhalte:

Teilnehmer 3: Marius Müller
Inhalte:

- Kapitel 3: Optimierungsstrategien