**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

Inhaltsverzeichnis

[1 Einleitung und Problemstellung – David 3](#_Toc107057518)

[2 Allgemeines 4](#_Toc107057519)

[3 Verbesserte Algorithmen zur Pfadplanung 5](#_Toc107057520)

[4 Optimierung 6](#_Toc107057521)

[4.1 Bidirektionale Suche 6](#_Toc107057522)

[4.2 Informed Search Strategie 6](#_Toc107057523)

[4.3 A\* Suche 6](#_Toc107057524)

[4.4 ALT Algorithmen 7](#_Toc107057525)

[4.5 Reach-Based Pruning 7](#_Toc107057526)

[4.6 Experimentelle Analyse 8](#_Toc107057527)

[5 Zusammenfassung und Ausblick 9](#_Toc107057528)

# Einleitung und Problemstellung – David

Das Thema Funktionsprinzipien und Anwendungen von Algorithmen zur Pfadplanung hat damals, wie heute einen wichtigen Stellenwert. Ob im Bereich der Netzwerkroutenplanung oder bei KI-Spielern in Computerspielen Pfadsuchalgorithmen sind so relevant wie nie. Weitere Beispiele für die Anwendung von Pfadsuchalgroithmen sind Robotik (z.B. Pakete im Logistikbereich), Planung von öffentlichen Verkehrsmitteln und Routenplanung von Navigationssystemen. Viele Breiche im Alltag verwenden im Hintergrund Pfadsuchalgorithmen um den (kosten)günstigsten Weg zu finden, dabei ist es wichtig, dass diese effizient, akkurat und schnell sein müssen, damit die Hauptsysteme noch genügen Ressourcen übrig haben um gewünscht zu funktionieren.

Wir werden die Funktionsweise dieser Algorithmen, angefangen mit den „einfachen“ uninformierten Suchalgorithmen wie die Breitensuche, bis hin zu durch Heuristiken optimierte und informierte Pfadsuchalgorithmen wie Dijsktra oder A\*, veranschaulichen und jeweils Code und Anwendungsbeispiele geben.

# Allgemeines – David

**Breitensuche:**

Die Breitensuche gehört zu den uninformierten Suchalgorithmen, diese werden auch „blind“ genannt, weil bei ihrer Suche auf keine zusätzlichen Informationen (wie z.B. Wichtungen) zurückgegriffen wird.

Bei der Breitensuche wird zunächst vom Wurzelknoten aus betrachtet alle verbunden Knoten ersten Grades besucht und dies Ebene für Ebene im Baum wiederholt bis alle Knoten besucht wurden.

Die Breitensuche findet weitestgehend in der Graphentheorie seine Anwendung.  *(Aritificial Intelligence: A modern Approach | S.81)*

**Tiefensuche:**

Die Tiefensuche gehört ebenfalls zu den uninformierten Suchalgorithmen.

Im Gegensatz zu der Breitensuche werden nicht die Ebenen nacheinander abgesucht sondern je Nachfolger angefangen beim Wurzelknoten werden bis sie keine weiteren Nachfolger mehr haben besucht. Erst dann wird der nächste Nachbar in der ersten Ebene besucht bis keine unbesuchten Knoten mehr vorhanden sind.  Die Tiefensuche ist indirekt an vielen komplexeren Algorithmen beteiligt. Unter anderem kann die Tiefensuche auch für das Ermitteln von Zusammenhangskomponenten oder für das Erzeugen eines Irrgartens verwendet werden.

*(Aritificial Intelligence: A modern Approach | S.85)*

# Verbesserte Algorithmen zur Pfadplanung

Here ist meine Arbeit

# Optimierung

## Bidirektionale Suche

Bei der bidirektionalen Suche wird ein Suchalgorithmus simultan aus zwei Richtungen laufen lassen – 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. [6]

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. [6]

Eine Implementierung der Bidirektionalen Suche ist der Bidirektionale Dijkstra Algorithmus von Vaira und Kurasova. [7] Hier wird der in Abschnitt XY 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.

## Informed Search Strategie

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 ausgewählt. Eine Komponente dieser Bewertungsfunktion ist eine heuristische Funktion , die die zu erwartenden Kosten des optimalen Pfades vom Knoten zum Zielknoten berechnet.[1] Im Falle des Straßennetzes könnte hierzu die Länge der Luftlinie zwischen dem Knoten und dem Zielknoten verwendet werden. [2]

Die einfachste Umsetzung dieser Strategie ist, nur die heuristische Funktion bei der Bewertung von Knoten heranzuziehen, sodass 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. [1] 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. [3]

## A\* Suche

Der A\*-Algorithmus zur Berechnung des kürzesten Pfades zwischen zwei Knoten ist eine weitere Umsetzung der Informed Search Strategie. [1] A\* basiert auf dem in Abschnitt XY vorgestellten Dijkstra’s Algorithmus und erweitert diesen um eine heuristische Funktion, um die Laufzeit zu reduzieren. [4] Die Bewertungsfunktion für den A\*-Algorithmus setzt sich zusammen aus , den Kosten des optimalen Pfades vom Startknoten bis zum Knoten und der heuristischen Funktion , sodass gilt:

[2]

Da verschiedene Heuristiken zur Konstruktion von gewählt werden können, stellt A\* streng genommen eine Familie von Algorithmen dar, wobei die Wahl einer Funktion einen spezifischen Algorithmus der Familie selektiert. [2]

Hart, Nilsson und Raphael, die 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.[2] 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\*. [1]

## ALT Algorithmen

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. [8] Eine Familie von Algorithmen, die auf Preprocessing basieren, sind die von Goldberg und Harrleson in [8] vorgestellten ALT-Algorithmen. ALT ist ein Akronym für A\* search, landmarks und triangle inequality (Dreiecksungleichung). In der Preprocessing-Phase des Algorithmus, wird eine kleine (konstante) Anzahl von Landmarken im Graphen ausgewählt, von denen aus dem kürzesten Pfad zu allen anderen Knoten bestimmt wird.

Für die Bewertungsfunktion der A\* Suche wird die so bestimmte Distanz in Kombination mit der Dreiecksungleichung als Heuristik verwendet. [8] Die Dreiecksungleich besagt in diesem Fall, dass die Distanz zwischen einem Knoten und einem Zielknoten in jedem Fall größer oder gleich der Differenz der Distanz zwischen und einer Landmarke und der Distanz zwischen und der Landmarke ist.

Somit stellt diese Differenz eine untere Schranke für die Distanz zwischen und dar und kann somit als Heuristik verwendet werden.[8]

## Reach-Based Pruning

Die Reach-Based Pruning ist eine von Gutman [9] vorgestellte Preprocessing-Strategie zur Vereinfachung von Graphen. Der Reach ist hierbei eine Metrik für einen Graphen , die als Modifikation für den Dijkstra Algorithmus verwendet werden kann.

Sei ein Pfad in von einem Startknoten zu einem Zielknoten und ein Knoten auf . Sei zudem die Distanz zwischen den Knoten und auf dem Pfad . Dann ist

der Reach von auf . Zudem ist der Reach von in definiert als das Maximum aller für alle kürzesten Pfade in .

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

gilt, wobei eine untere Schranke für die Distanz zwischen zwei Knoten und darstellt.

Die einfachste Möglichkeit die Reaches aller Knoten zu bestimmen, ist alle kürzesten Pfade eines Graphen zu bestimmen und die Definition XY anzuwenden. Effizientere Vorgehen sind in [5],[9] beschrieben.

## Experimentelle Analyse

Um die Auswirkung der hier vorgestellten Optimierungsstrategien zu veranschaulichen, wurde von Goldberg in [5] die Laufzeit der folgenden Algorithmen verglichen:

1. B: der bidirektionale Dijkstra Algorithmus
2. ALT: Algorithmus aus der ALT-Familie
3. RE: eine Implementierung der Reach-Based Pruning Strategie
4. 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. Gemessen wurde die Laufzeit der Preprocessing Phase und die des eigentlichen Suchalgorithmus, die Anzahl der expandierten Knoten und der Speicherplatzbedarf.

Folgende Messwerte wurden von Goldberg bestimmt:

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Abbildung : Random Grid

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Abbildung 2: Toshiba Tecra 5 laptop mit 2GB RAM und dual-core 2 GHz processor

# Zusammenfassung und Ausblick