**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 3](#_Toc107392513)

[2 Algorithmen zur Pfadplanung 4](#_Toc107392514)

[2.1 Was ist Pfadplanung Pfadplannung 4](#_Toc107392515)

[2.2 Uninformierter Ansatz 4](#_Toc107392516)

[2.3 Dijkstra Algorithmus 5](#_Toc107392517)

[2.4 Bellman-Ford-Algorithmus 7](#_Toc107392518)

[3 Mögliche Optimierungsstrategien 9](#_Toc107392519)

[3.1 Bidirektionale Suche 9](#_Toc107392520)

[3.2 Informed Search Strategie 9](#_Toc107392521)

[3.3 A\* Suche 9](#_Toc107392522)

[3.4 ALT Algorithmen 10](#_Toc107392523)

[3.5 Reach-Based Pruning 10](#_Toc107392524)

[4 Anwendungen 12](#_Toc107392525)

[4.1 Experimentelle Analyse 12](#_Toc107392526)

[4.2 GIS(Geoinformationssystem) 12](#_Toc107392527)

[5 Zusammenfassung und Ausblick 15](#_Toc107392528)

# Einleitung und Problemstellung

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.

Path Planning Algorithms

In diesem Teil des Papers werden wir über Pfadplanungsalgorithmen wie Dijkstra-Algorithmus und seinen Varianten, die häufig in Anwendungen wie Google Maps und anderen Verkehrsleitsystemen verwendet werden.

Um die Rechenintensität des Dijkstra-Algorithmus bei Blindsuchen zu überwinden, werden A\* und seine Varianten als Stand der Technik-Algorithmen für den Einsatz in statischen Umgebungen vorgestellt[2].

Dann werden wir über den Algorithmus von Bellman und Ford sprechen, der das Problem des kürzesten Weges für eine einzige Quelle löst, bei dem die Kantengewichte negativ sein können. Der Bellman und Ford-Algorithmus für kürzeste Wege ist fast völlig intuitiv und liefert einen booleschen Wert, der angibt, ob es einen Zyklus mit negativem Gewicht gibt, der von der Quelle aus erreichbar ist oder nicht [4].

TO-DO:

* Salahs Teil (grün) in Einleitung verarbeiten
* Deinen Teil einfügen

# Algorithmen zur Pfadplanung

## Was ist Pfadplanung

Die Pfadplanung ist ein nichtdeterministisches, polynomialzeitliches ("NP") schweres Problem mit der Aufgabe, einen kontinuierlichen Pfad zu finden, der ein System von einer Ausgangs- zu einer Endkonfiguration verbindet.

Die Komplexität des Problems steigt mit zunehmender Anzahl der Freiheitsgrade des Systems. Der zu verfolgende Pfad (der optimale Pfad) wird auf der Grundlage von Einschränkungen und Bedingungen bestimmt, z. B. im Bereich mobiler Roboter unter Berücksichtigung des kürzesten Weges zwischen den Endpunkten oder der minimalen Fahrzeit ohne Kollisionen. Manchmal werden Einschränkungen und Ziele gemischt, z. B. um den Energieverbrauch zu minimieren, ohne dass die Fahrzeit einen bestimmten Schwellenwert überschreitet [1].

## Uninformierter Ansatz

**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)*

## Dijkstra Algorithmus

Definition und Prinzip:

Der Dijkstra-Algorithmus (benannt und konzipiert von dem Informatiker Edsger W. Dijkstra) löst das Problem, den kürzesten Weg von einem Punkt in einem Graphen (der Quelle) zu einem Ziel zu finden.

Es stellt sich heraus, dass man die kürzesten Pfade von einer gegebenen Quelle zu allen Punkten in einem Graphen gleichzeitig finden kann, daher wird dieses Problem manchmal als Problem der kürzesten Pfade einer einzelnen Quelle bezeichnet.[3]

Der Dijkstra-Algorithmus kann den optimalen Pfad auswählen, der die Bedingungen z. B. aus einer Topologie-Roadmap erfüllt. Der traditionelle Dijkstra-Algorithmus unterteilt die Knoten im topologischen Netz in drei Teile: erstens, alle anfänglichen Knoten in den Algorithmus Netzwerk sind unmarkierte Knoten, und die Knoten, die zu passieren und zu verbinden in den Prozess der Pfadauswahl und optimalen Pfad Auswahl Für die temporären Knoten, wählt jeder Screening-Zyklus im Prozess der optimalen Pfadauswahl den Knoten mit der kürzesten Pfadlänge vom temporären Markierungsknoten als permanenten Markierungsknoten aus, und der Dijkstra-Algorithmus wird fortgesetzt, bis der Zielknoten oder alle Knoten erreicht sind. Der Knoten endet erst, wenn er zu einem permanenten Markierungsknoten wird [5].

Dijkstras Pseudo-Code:

Die Grundidee des Dijkstra-Algorithmus besteht darin, einige anfängliche Entfernungswerte zuzuweisen und zu versuchen, diese schrittweise zu verbessern. In der folgenden Abbildung ist der Pseudocode des Algorithmus dargestellt. [6]

Text

Description automatically generated

Wie der Pseudo-Code zeigt, funktioniert der Dijkstra-Algorithmus, indem er das Teilproblem k löst, das den kürzesten Weg von der Quelle zu den nächstgelegenen Knoten der Quelle berechnet, von der Quelle zum Ziel.

Damit der Dijkstra-Algorithmus funktioniert, sollte es sich um einen gerichteten, gewichteten Graphen handeln und die Kanten sollten, wie bereits erwähnt, nicht negativ sein. Wenn die Kanten negativ sind, kann der tatsächlich kürzeste Weg nicht ermittelt werden. Daher wird die kürzeste Entfernung in der Sequenz S gespeichert. [7]

Dijkstras Vorteile:

Der Dijkstra-Algorithmus kann alle optimalen Pfade finden, und die Trefferquote dieser optimalen Pfade liegt bei 100 % [8].

Dijkstras Nachteile:

Der größte Nachteil des Algorithmus ist die Tatsache, dass er eine blinde Suche durchführt und dadurch viel Zeit vergeudet, d. h. er ist zeitaufwendig. Ein weiterer Nachteil ist, dass der Algorithmus nicht mit negativen Kanten umgehen kann, was zu azyklischen Graphen führt, und dass er oft nicht den richtigen kürzesten Weg findet [9].

## Bellman-Ford-Algorithmus

Definition und Prinzip:

Der Bellman-Ford-Algorithmus verwendet Entspannung, um kürzeste Pfade auf gerichteten Graphen zu finden, die nur eine Quelle haben.

Der Algorithmus erkennt auch, ob es negative Gewichtszyklen gibt (so dass es keine Lösung gibt). Wenn es um Entfernungen auf einer Karte geht, gibt es keine negativen Entfernungen. Die Grundstruktur des Bellman-Ford-Algorithmus ist ähnlich wie die des Dijkstra-Algorithmus. Er entspannt alle Kanten und tut dies |V| - 1 Mal, wobei |V| die Anzahl der Knoten im Graphen ist [10].

Bellman-Ford Pseudo-Code:

Text

Description automatically generated

Der Bellman-Ford-Algorithmus wird wie in der Abbildung oben dargestellt ausgeführt.

Schritt 1: Setzen Sie den Abstand des Quellknotens s auf den Wert Null (distance[s] = 0) und weisen Sie den anderen Knoten einen Abstand von INFINITY zu.

Schritt 2: entspannt jede Kante (n - 1) Mal, wenn n die Anzahl der Knoten ist. Das Entspannen einer Kante bedeutet zu prüfen ob es möglich ist, den Weg zu dem Knoten, auf den die Kante zeigt, zu verkürzen, und, wenn ja, den Weg zu den Knoten durch die gefundene Route.

Schritt 3: Prüfen, ob der Graph einen negativen Zyklus hat, mit Ausführung der N-ten Schleife. Algorithmus 2 zeigt den Bellman-Ford-Algorithmus [11].

# Mögliche Optimierungsstrategien

## 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

Eine elaboriertere Umsetzung der der Informed Search Strategie ist A\*-Algorithmus zur Berechnung des kürzesten Pfades zwischen zwei Knoten. [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.

# Anwendungen

## 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 1: 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

## GIS(Geoinformationssystem)

Ein geografisches Informationssystem (GIS) ist ein computergestütztes Werkzeug. Mit diesen Werkzeugen können wir räumliche Informationen erstellen, manipulieren, analysieren, speichern und anzeigen. Räumliche Informationen sind Informationen über Objekte, die sich auf der Erde befinden, wie z. B. Städte, Eisenbahnstrecken, Flüsse usw. [12].

Geoinformationssystem mit Dijkstra-Algorithmus:

Implementierung eines geographischen Informationssystems mit Dijkstra Algorithmus auf Basis einer mobilen Anwendung soll Informationen über die nächstgelegene Evakuierungsroute. Benutzer können die Anwendung über eine mobile Anwendung nutzen und die nächstgelegene die nächstgelegene Evakuierungsroute empfehlen und mit Google Maps angezeigt.

Routen-Finder: Der Route Finder findet die optimierte Route durch die Kommunikation mit dem GIS Server-Network Analysis Service, mit Parametern, die den Standort des Leiters und den zugewiesenen den Standort des Leiters und der zugewiesenen Unterkunft, um die kürzeste verfügbare zwischen den beiden Punkten mit detaillierter Routenbeschreibung. Anschließend wird die Route auf einer GIS-Karte an den Anführer zurück [13].

Mobile Roboter mit verbessertem Dijkstra-Algorithmus:

Der Dijkstra-Algorithmus kann außer den bereits durchlaufenen Knoten keine Daten speichern. Um diesen Nachteil zu überwinden, wird ein Speicherschema eingeführt, das ein mehrschichtiges Wörterbuch implementiert, das aus zwei Wörterbüchern und einer Liste von Datenstrukturen besteht, die in hierarchischer Reihenfolge organisiert sind. Das erste Wörterbuch bildet jeden einzelnen Knoten auf seine Nachbarknoten ab. Das zweite Wörterbuch speichert die Pfadinformationen jedes benachbarten Pfades.

Ein mehrschichtiges Wörterbuch bietet eine umfassende Datenstruktur für den Dijkstra-Algorithmus in einer Innenraumanwendung, bei der die Koordinaten des globalen Navigationssatellitensystems und die Kompassorientierung nicht zuverlässig sind. Die Pfadinformationen in der Datenstruktur helfen dabei, den Grad des Drehwinkels zu bestimmen, den der Roboter an jedem Knoten oder jeder Kreuzung ausführen muss. Der vorgeschlagene Algorithmus liefert den kürzesten Pfad in Bezug auf die Länge und gleichzeitig den navigierbarsten Pfad in Bezug auf den niedrigsten erforderlichen Gesamtdrehwinkel in Grad, der mit dem traditionellen Dijkstra-Algorithmus nicht berechnet werden kann [14].

Diagram

Description automatically generated

**Abbildung 2.** Flussdiagramm des verbesserten Dijkstra-Algorithmus für die Pfadplanung.

# Zusammenfassung und Ausblick

*Literatur:* *(Aritificial Intelligence: A modern Approach | 108-109)*

In diesem Paper wurden Methoden vorgestellt, die ein Akteur zur Auswahl von Aktionen in Umgebungen wie z.B Graphen verwenden kann, die deterministisch, beobachtbar, statisch und vollständig bekannt sind. Mit diesen Methoden kann der Akteur durch eine Sequenz von Aktionen sein Ziel erreichen. Diesen Prozess nennt man **Suche**.

* Bevor ein Akteur mit der Suche nach Lösungen beginnen kann, muss ein Ziel identifiziert und ein Problem genau definiert Problem werden.
* Ein Problem besteht aus fünf Teilen:  - dem Ausgangszustand,  - einer Reihe von Aktionen,  - einem Übergangsmodell, das die Ergebnisse dieser Aktionen beschreibt,  - einer Zielüberprüfungsfunktion  - und einer Pfadkostenfunktion.   Das Problem wird durch eine Menge an Zuständen dargestellt. Ein Pfad der aus der Zustandsmenge besteht und vom Start zum Ziel führt ist eine Lösung.
* Suchalgorithmen behandeln Zustände und Aktionen atomar: Sie berücksichtigen keine interne Struktur, die sie besitzen könnten.
* Ein allgemeiner TREE-SEARCH-Algorithmus berücksichtigt alle möglichen Wege, um eine Lösung zu finden, während ein GRAPH-SEARCH-Algorithmus redundante Wege vermeidet.
* Suchalgorithmen werden auf der Grundlage von Vollständigkeit, Optimalität, Zeit- und Raumkomplexität beurteilt. Die Komplexität hängt von b, dem Verzweigungsfaktor im Zustandsraum, und d, der Tiefe der tiefsten Lösung, ab.
* Uninformierte Suchmethoden haben nur Zugriff auf die Problemdefinition. Die grundlegenden Algorithmen sind wie folgt:
  + Die Breadth-first-Suche expandiert zuerst die flachsten Knoten; sie ist vollständig, optimal für einheitliche Pfadkosten, hat aber eine exponentielle Raumkomplexität.
  + Die Uniform-Cost-Suche expandiert den Knoten mit den niedrigsten Pfadkosten, g(n), und ist optimal für allgemeine Pfadkosten.
  + Die Tiefensuche expandiert zuerst den tiefsten nicht expandierten Knoten. Sie ist weder vollständig noch optimal, hat aber eine lineare Raumkomplexität.
  + Die iterative Vertiefungssuche ist eine Wiederholung der Tiefensuche mit zunehmender Tiefenbegrenzung, bis ein Ziel gefunden wird. Sie ist vollständig, optimal für die Kosten pro Schritt, hat eine vergleichbare Zeitkomplexität wie die Breitensuche und eine lineare Raumkomplexität.
  + Die bidirektionale Suche kann die Zeitkomplexität enorm reduzieren, ist aber nicht immer anwendbar und kann zu viel Speicherplatz beanspruchen.
* Informierte Suchmethoden können auf eine heuristische Funktion h(n) zurückgreifen, die die Kosten einer Lösung aus n schätzt.
  + Der generische Best-First-Suchalgorithmus wählt einen Knoten für die Expansion gemäß einer Bewertungsfunktion aus.
  + Der Greedy best-first search expandiert Knoten mit minimalem h(n). Er ist nicht optimal, aber oft effizient.

* + A∗-Suche expandiert Knoten mit minimalem f (n) = g(n) + h(n). A∗ ist vollständig und optimal, vorausgesetzt, h(n) ist zulässig (für TREE-SEARCH) oder konsistent (für GRAPH-SEARCH).
  + RBFS (rekursive Best-First-Suche) und SMA∗ (vereinfachtes speicherbegrenztes A∗) sind robuste, optimale Suchalgorithmen, die nur begrenzte Mengen an Speicher verwenden; mit genügend Zeit können sie Probleme lösen, die A∗ nicht lösen kann, weil ihm der Speicher ausgeht.
* Die Leistung von heuristischen Suchalgorithmen hängt von der Qualität der heuristischen Funktion ab. Manchmal kann man gute Heuristiken konstruieren, indem man die Problemdefinition lockert und vorberechnete Lösungskosten für Teilprobleme in einer Musterdatenbank speichert.

# 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.