

## Funktionsprinzipien und Anwendungen von Algorithmen zur Pfadplanung

Bearbeiter 1: Mohammed Salih Mezraoui

Bearbeiter 2: David Gruber

Bearbeiter 3: Marius Müller

Gruppe: 7

Ausarbeitung zur Vorlesung Wissenschaftliches Arbeiten

Trier, 17.07.2022

---

## Kurzfassung

In dieser wissenschaftlichen Arbeit werden Algorithmen zur Pfadplanung dargestellt und analysiert. Es wird anhand der Anwendung in Geoinformationssystemen und bei mobilen Robotern aufgezeigt, wie der Algorithmus von Dijkstra eingesetzt wird 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.

---

# Inhaltsverzeichnis

<b>1 Einleitung und Problemstellung</b> .....	1
<b>2 Algorithmen zur Pfadplanung</b> .....	2
2.1 Was ist Pfadplanung? .....	2
2.2 Uninformierter Ansatz .....	2
2.2.1 Breitensuche .....	2
2.2.2 Tiefensuche .....	3
2.3 Dijkstra Algorithmus .....	3
2.3.1 Definition .....	3
2.3.2 Prinzip .....	3
2.3.3 Dijkstras Vorteile .....	4
2.3.4 Dijkstras Nachteile .....	4
2.3.5 Dijkstras Pseudo-Code .....	4
2.4 Bellman-Ford-Algorithmus .....	6
2.4.1 Definition .....	6
2.4.2 Prinzip .....	6
2.4.3 Bellman-Ford Pseudo-Code .....	6
<b>3 Optimierungsstrategien</b> .....	8
3.1 Bidirektionale Suche .....	8
3.2 Informed Search .....	8
3.3 Preprocessing .....	9
3.3.1 ALT-Algorithmen .....	9
3.3.2 Reach-Based Pruning .....	10
3.4 Experimentelle Analyse .....	11
<b>4 Anwendungen</b> .....	12
4.1 GIS(Geoinformationssystem) .....	12
4.1.1 Definition .....	12
4.1.2 Geoinformationssystem mit Dijkstra-Algorithmus .....	12
4.2 Mobile Roboter mit verbessertem Dijkstra-Algorithmus .....	14
4.2.1 traditioneller Dijkstra vs besserer Dijkstra .....	14
4.2.2 Verbesserter Dijkstra .....	15

4.3 Autonome Navigation .....	16
<b>5 Zusammenfassung und Ausblick .....</b>	<b>17</b>
<b>Literaturverzeichnis .....</b>	<b>19</b>

# 1

---

## Einleitung und Problemstellung

In diesem Paper werden wir aufeinander aufbauend Algorithmen vorstellen, die in einem Programm mit bestimmten Eingaben und Umgebungen wie z.B Graphen verwendet werden können um von einem gegebenen Start ein oder mehrere Ziele zu finden und dabei durch verschiedene Bewertungskriterien den besten Weg zu bestimmen [Esr22]. Mit diesen Algorithmen kann ein Programm durch eine Sequenz von Aktionen sein Ziel erreichen. Diesen Prozess nennt man Suche [RN10, 150-156].

In diesem Paper werden wir verschiedene Pfadplanungsalgorithmen wie den Dijkstra-Algorithmus und der A\* Algorithmus, die im Verkehrleitsystem von Google Maps verwendet werden, vorstellen [MKL19].

Das Thema *Funktionsprinzipien und Anwendungen von Algorithmen zur Pfadplanung* hat, damals wie heute einen wichtigen Stellenwert. Ob im Bereich der Netzwerkrouteplanung oder bei KI-Spielern in Computerspielen: Pfadsuchalgorithmen sind so relevant wie nie [FGK<sup>+</sup>21]. Weitere Beispiele für die Anwendung von Pfadsuchalgorithmen sind Planung von öffentlichen Verkehrsmitteln und Robotik. Es werden zum Beispiel Pakete in Logistikzentren durch Roboter organisiert und Routen über verschiedene Logistikzentren durch Pfadsuchalgorithmen bestimmt. Viele Bereiche 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 genügend Ressourcen übrig haben um gewünscht zu funktionieren [FGK<sup>+</sup>21].

Wir werden uns in diesem Paper die Funktionsweise der wichtigsten Pfadsuchalgorithmen, angefangen mit den uninformierten Suchalgorithmen wie der Breiten- oder Tiefensuche, welche einen ersten Einblick in die Thematik geben sollen und die Rahmenbedingungen und Problemumgebungen veranschaulichen sollen; Über den Bellman-Ford-Algorithmus, der nahezu intuitiv den kürzesten Weg liefert und bei dem auch negative Kantenlängen möglich sind[MS20a]. Bis hin zu durch Heuristiken optimierte und informierte Pfadsuchalgorithmen wie Dijkstra oder A\*, und wie sie in heutiger Zeit sonst eingesetzt werden können, veranschaulichen und jeweils (Pseudo-)Code- und Anwendungsbeispiele geben[RN10, 64].

## 2

---

# Algorithmen zur Pfadplanung

In diesem Kapitel werden zwei der am häufigsten verwendeten Algorithmen (Dijkstra und Bellman) für die Planung von Pfaden beschrieben.

## 2.1 Was ist Pfadplanung?

Karthik Karur, Nitin Sharma, Chinmay Dharmatti, und Joshua E. Siegel haben Pfadplanung definiert als „ein nicht-deterministisches Polynomialzeit ("NP") schweres Problem definiert, dessen Aufgabe es ist, einen kontinuierlichen Pfad zu finden, der ein System von einer Anfangs- zu einer Endkonfiguration verbindet“<sup>1</sup>.

Die Komplexität des Problems wächst mit der Zunahme der Freiheitsgrade des Systems. Die zu wählende Route (die beste Route) wird durch Einschränkungen und Begrenzungen bestimmt, wie z. B. die kürzeste Entfernung zwischen den Endpunkten oder die kürzeste Zeit für eine kollisionsfreie Fahrt. Gelegentlich werden Beschränkungen und Ziele kombiniert, z. B. der Versuch, den Energieverbrauch zu begrenzen und gleichzeitig einen bestimmten Schwellenwert für die Fahrzeit nicht zu überschreiten.[KSDS21a].

## 2.2 Uninformierter Ansatz

### 2.2.1 Breitensuche

Die Breitensuche gehört zu den uninformierten Suchalgorithmen. Diese werden auch „blind“ genannt, weil bei ihrer Suche auf keine zusätzlichen Informationen (wie z.B. Gewichtungen) zurückgegriffen wird [RN10, 81]. Der erste Schritt der Breitensuche ist es alle verbundenen Knoten ersten Grades des Wurzelknotens zu besuchen. Dies wird Ebene für Ebene im Baum wiederholt, bis alle Knoten besucht wurden. Die Breitensuche findet weitestgehend in der Graphentheorie ihre Anwendung [RN10, 81].

---

<sup>1</sup> Definition der Pfadplanung [KSDS21a]

### 2.2.2 Tiefensuche

Die Tiefensuche gehört ebenfalls zu den uninformierten Suchalgorithmen. Das Vorgehen bei der Tiefensuche ist es zuerst alle Unterknoten eines direkten Nachbarn des Wurzelknotens zu besuchen bis dieser keine Unterknoten mehr hat. Erst dann wird der nächste Nachbar in der ersten Ebene besucht bis keine unbesuchten Knoten mehr vorhanden sind [RN10, 85,86]. Die Tiefensuche ist indirekt an vielen komplexeren Algorithmen beteiligt. So nutzt die iterative Vertiefungssuche die klassische Tiefensuche in Wiederholung in Verbindung mit einer inkrementellen Tiefenbegrenzung bis ein Ziel gefunden wird [RN10, 108,109]. Außerdem kann die Tiefensuche auch für das Ermitteln von Zusammenhangskomponenten oder für das Erzeugen eines Irrgartens verwendet werden [RN10, 85,86].

## 2.3 Dijkstra Algorithmus

### 2.3.1 Definition

Adeel Javaid beschrieb, dass „Der Dijkstra-Algorithmus (benannt nach seinem Entdecker E.W. Dijkstra) das Problem löst, den kürzesten Weg von einem Punkt in einem Diagramm (der Quelle) zu einem Ziel zu finden“<sup>2</sup>.

### 2.3.2 Prinzip

Da nachgewiesen wurde, dass die kürzesten Wege von einem Quellknoten zu allen Orten in einem Diagramm tatsächlich in derselben Zeit gefunden werden können, wird dieses Problem manchmal als Problem der kürzesten Wege für eine einzige Quelle bezeichnet[Jav19].

Der Dijkstra-Algorithmus kann die beste Route wählen, die die Voraussetzungen für die Topologie-Roadmap erfüllt. Der klassische Dijkstra-Algorithmus unterteilt die Knoten im topologischen System in drei Gruppen: zunächst alle vorläufigen Knoten im System des Algorithmus, die nicht gekennzeichnet sind, und die Knoten, die sich während der Routenauswahl und der effizienten Routenauswahl kreuzen und verbinden sollen. Jede Screening-Schleife im optimalen Routenauswahlverfahren wählt den Knoten mit der kürzesten Pfadlänge vom momentanen Indikator knoten als permanenten Markierung-knoten, und der Dijkstra-Algorithmus iteriert, bis der aktuelle und der geplante oder alle Knoten erreicht sind. Der Knoten würde dann so lange bestehen bleiben, bis er durch einen permanenten Markierung-knoten ersetzt wird[ZG13].

---

<sup>2</sup> Dijkstra Algorithmus Definition [Jav19]

### 2.3.3 Dijkstras Vorteile

- Minhang Zhou und Nina Gao haben zitiert „Der Dijkstra-Algorithmus kann alle optimalen Pfade finden, und die Trefferquote dieser optimalen Pfade liegt bei 100 %“<sup>3</sup>.
- Wenn der geplante Zielknoten erreicht ist, besucht der Dijkstra-Algorithmus die restlichen unerwünschten Knoten nicht[AIS<sup>+</sup>20].

### 2.3.4 Dijkstras Nachteile

Der Hauptnachteil des Algorithmus besteht darin, dass er eine blinde Suche durchführt, wodurch viel Zeit und relevante Ressourcen verschwendet werden, oder anders ausgedrückt, er ist zeitintensiv. Ein weiterer Nachteil ist, dass er keine negativen Seiten verwalten kann, was zu azyklischen Graphen führt, und dass er häufig nicht die beste Route findet[MS20b].

### 2.3.5 Dijkstras Pseudo-Code

Der Dijkstra-Algorithmus arbeitet mit der Zuweisung einiger vorläufiger Entfernungswerte und versucht, diese schrittweise zu verbessern. Der Pseudocode des Algorithmus ist in der folgenden Abbildung dargestellt[HG12].

---

<sup>3</sup> Erster Vorteil von Dijkstra [ZG13]

---

**Algorithm 1** Dijkstra Algorithm

---

```

1: function DIJKSTRA (Graph, source)
2:   create vertex set D
3:   for each vertex v in Graph:
4:     distance[v]  $\leftarrow$  INFINITY
5:     previous[v]  $\leftarrow$  UNDEFINED
6:     add v to D
7:     distance[source]  $\leftarrow$  0
8:
9:   while D is not empty do:
10:    u  $\leftarrow$  in D with min distance[u]
11:    remove u from D
12:    for each neighbour v of u:
13:      alt  $\leftarrow$  distance[u] + length (u, v)
14:      if alt < distance[v]
15:        distance[v]  $\leftarrow$  alt
16:        previous[v]  $\leftarrow$  u
17:   end while
18:   return distance [], previous []
19: end function

```

---

Abb. 2.1: Dijkstra Algorithmus Pseudo-Code [AIS<sup>+</sup>20].

Abusalim Samah, Ibrahim Rosziati, Saringat Mohd, Jamel Sapiee und Wahab Jahari haben diesen Pseudocode wie folgt beschrieben:

„Beim Dijkstra-Algorithmus ist der Pfad nicht bekannt. Die Knoten werden in zwei Gruppen unterteilt: temporäre (t) und permanente (p).“

- Zunächst wird die Entfernung des Quellknotens mit dem Wert Null initialisiert [ $\text{distance}(a) = 0$ ], und die Entfernung der anderen Knoten wird mit dem Wert Unendlich belegt [ $\text{distance}(x) = \text{infinity}$ ].
- Schritt 2: Suche nach dem Knoten x mit dem kleinsten Wert von  $d(x)$ . Wenn es keine temporären Knoten gibt oder der Wert von  $d(x)$  gleich unendlich ist, wurde der Knoten x als permanent eingestuft, was bedeutet, dass sich  $d(x)$  und der übergeordnete Wert von  $d(x)$  nicht mehr ändern werden.

- Schritt 3: Wenden Sie den folgenden Vergleich für jeden temporären Knoten mit der Bezeichnung vertex  $y$  an, der an  $x$  angrenzt<sup>4</sup>.

## 2.4 Bellman-Ford-Algorithmus

### 2.4.1 Definition

Vaibhavi Patel und Prof. Chitra Baggar haben benotet dass, „Der Bellman-Ford-Algorithmus verwendet Entspannung, um kürzeste Pfade auf gerichteten Graphen zu finden, die nur eine Quelle haben“<sup>5</sup>.

### 2.4.2 Prinzip

Wenn es negative Gewichtsnusszyklen gibt, wird der Algorithmus sie erkennen. Eine negative Länge gibt es nicht, wenn es sich um Bereiche auf einer Karte handelt. Der Ballmann-Ford-Algorithmus ist im Allgemeinen mit dem Dijkstra-Algorithmus vergleichbar. Er entspannt alle Kanten  $|V|$  mal, wobei  $|V|$  die Menge der Scheitelpunkte ist[VP14].

### 2.4.3 Bellman-Ford Pseudo-Code

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

---

<sup>4</sup> Ausführliche Beschreibung von Dijkstra Pseudo Code [AIS<sup>+</sup>20]

<sup>5</sup> Bellman-Ford-Algorithmus Definition [VP14]

---

**Algorithm 2 Bellman-Ford Algorithm**

---

```

1: function bellmanFord (G, S)
2:   for each vertex V in G
3:      $distance[v] \leftarrow \text{INFINITY}$ 
4:      $previous[v] \leftarrow \text{NULL}$ 
5:      $distance[s] \leftarrow 0$ 
6:   for each vertex V in G
7:     for each edge (u, v) in G
8:        $alt \leftarrow distance[u] + length(u, v)$ 
9:       if  $alt < distance[v]$ 
10:         $distance[v] \leftarrow alt$ 
11:         $previous[v] \leftarrow u$ 
12:
13:   for each edge (u, v) in G
14:     if  $distance[u] + length(u, v) < distance(v)$ 
15:       Error: Negative Cycle Exists
16: return  $distance[], previous[]$ 

```

---

Abb. 2.2: Bellman-Ford Pseudo-Code[AIS<sup>+</sup>20].

Abusalim Samah, Ibrahim Rosziati, Saringat Mohd, Jamel Sapiee und Wahab Jahari haben sie diesen Pseudocode wie folgt beschrieben:

- „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“<sup>6</sup>.

---

<sup>6</sup> Ausführliche Beschreibung von Bellman-Ford Pseudo Code [AIS<sup>+</sup>20]

# 3

---

## Optimierungsstrategien

Die Anforderungen an die Laufzeit von Algorithmen zur Pfadplanung steigt auf Grund der immer komplexer werdenden Systemen stetig weiter an. In diesem Kapitel wird beschrieben, wie die bisher vorgestellten Algorithmen zur Pfadplanung für bestimmte Anwendungsgebiete 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 [RN10, 90,91]. 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 [RN10, 90,91].

### 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 [RN10, 92-102]. 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 [RN10, 92-102]. 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 [RN10, 92-102]. Eine elaboriertere Umsetzung der Informed Search Strategie ist der A\* Algorithmus zur Berechnung des kürzesten Pfades zwischen zwei Knoten [RN10, 92-102]. A\* basiert auf Dijkstras Algorithmus und erweitert diesen um eine heuristische Funktion, um die Laufzeit zu reduzieren [PAG<sup>+</sup>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:

$$f(n) = g(n) + h(n) \quad (3.1)$$

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\* [RN10, 92-102].

### 3.3 Preprocessing

Eine weitere Optimierungsstrategie ist das Preprocessing, also die Vorverarbeitung des Graphen. Dabei wird gefordert, dass die Platzkomplexität der vorverarbeiteten Daten linear in der Größe des zu bearbeitenden Graphen ist, da in der Realität oft mit sehr großen Graphen gearbeitet wird. [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*<sup>1</sup>.

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. Die so bestimmte Distanz wird in Kombination mit der Dreiecksungleichung als Heuristik für die Bewertungsfunktion der A\* Suche verwendet.

---

<sup>1</sup> Dreiecksungleichung

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.

$$dist(n, s) \geq dist(l, n) - dist(l, s) \quad (3.2)$$

Diese Differenz stellt 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<sup>2</sup> 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

$$r(v, P) = \min (dist(s, v, P), dist(v, t, P)) \quad (3.3)$$

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

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

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.

---

<sup>2</sup> 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***: Bidirektonaler 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 mit 330024 Knoten und 793681 Kanten verwendet und jeder dieser Algorithmen wurde auf 10.000 zufällig gewählten Paaren von Knoten angewendet. Dabei wurden die Laufzeit der Preprocssing-Phase und der Query-Phase der Algorithmen gemessen<sup>3</sup> und in Tabelle 3.1 dargestellt.

Algorithmus	Laufzeit Preprocessing	Laufzeit Query
<i>B</i>	-	30,49 ms
<i>ALT</i>	5,7 s	2,91 ms
<i>RE</i>	45,4 s	0,55 ms
<i>REAL</i>	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.

<sup>3</sup> Die Laufzeitmessungen wurden auf einem Toshiba Tecra 5 Laptop mit 2GB RAM und Dual-Core 2 GHz Processor durchgeführt

# 4

---

## Anwendungen

In diesem Kapitel wird beschrieben, wie Dijkstra- und Bellman-Algorithmen in realen Anwendungen angewendet werden können.

### 4.1 GIS(Geoinformationssystem)

#### 4.1.1 Definition

nach Vaibhavi Patel und Prof.Chitra Baggar „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“<sup>1</sup>.

#### 4.1.2 Geoinformationssystem mit Dijkstra-Algorithmus

Obwohl dieser Dijkstra-Algorithmus sehr effektiv zu sein scheint, kann es im Grunde genommen sehr lange dauern, die Route vom Start bis zum Ende weiterzuleiten, wobei eine große Menge an Informationen verwendet wird, wie in der Abbildung 4.1 zu sehen ist, was überhaupt nicht angemessen ist, so dass eine Alternative entwickelt werden muss, um die Zeit zu verkürzen[HAMTMA20].

---

<sup>1</sup> GIS Definition [VP14]

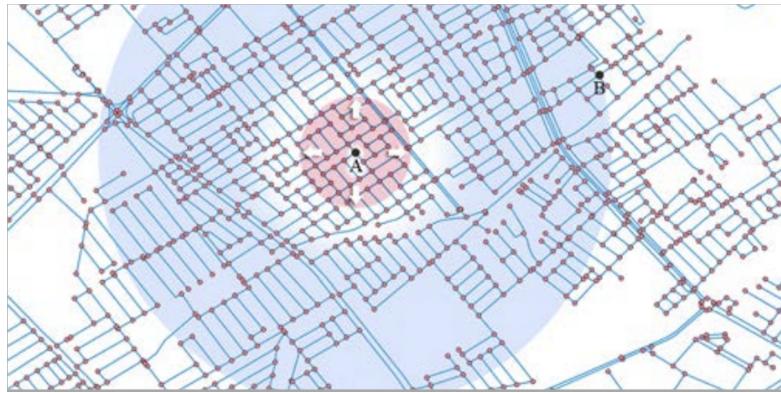


Abb. 4.1: Dijkstra's Fortschritt[HAMTMA20].

So haben die Autoren des Artikels<sup>2</sup> ihr Experiment wie folgt beschrieben:

„Der rote Kreis stellt den Verlauf des Dijkstra-Algorithmus dar, während der blaue Kreis die tatsächlichen Knoten und Linien des Graphen anzeigen, die der Algorithmus verarbeiten muss, bevor er die korrekte Wurzel von A nach B berechnen kann. Der verbesserte Algorithmus kann den Suchbereich erheblich verkleinern, wie in Abbildung 4.2 dargestellt. Dies ist nur ein Bruchteil des Bereichs in der vorherigen Abbildung. Er ermöglicht wesentlich schnellere Berechnungen und verkürzt damit die Zeit, die der Routing-Prozess benötigt, um einen gültigen Pfad zu finden. Die Verbesserungsmethode besteht im Wesentlichen darin, einen temporären Datensatz zu erstellen, bevor der Dijkstra-Algorithmus selbst gestartet wird, und diesen so zu behandeln, als wäre er der Graph, mit dem der Algorithmus arbeiten muss. Dieser Datensatz wird erstellt, nachdem die Start- und Endknoten erfasst wurden. Mit den Koordinaten der Start- und Endknoten kann der Datensatz aus den Hauptdaten ausgeschlossen werden, indem nur die Knoten ausgewählt werden, die sich innerhalb des von den beiden Knoten gebildeten Quadrats befinden“.



Abb. 4.2: Verbesserter Suchbereich[HAMTMA20].

<sup>2</sup> GIS Beispiel [HAMTMA20]

## 4.2 Mobile Roboter mit verbessertem Dijkstra-Algorithmus

### 4.2.1 traditioneller Dijkstra vs besserter Dijkstra

Die Autoren des Artikels<sup>3</sup> haben zitiert dass: „Der traditionelle Dijkstra-Algorithmus beruht auf einer gierigen Strategie zur Pfadplanung. Er wird verwendet, um den kürzesten Weg in einem Graphen zu finden. Er befasst sich mit der Lösung des kürzesten Weges, ohne sich formal um die Pragmatik der Lösung zu kümmern.“

Der modifizierte Dijkstra-Algorithmus zielt darauf ab, alle Knoten mit gleichem Abstand zum Ausgangsknoten als Zwischenknoten zu reservieren, und sucht dann von allen Zwischenknoten aus weiter, bis er erfolgreich zum Zielknoten durchläuft, siehe Abbildung 4.3. Durch Iteration werden alle möglichen kürzesten Wege gefunden und können dann ausgewertet werden“.

---

<sup>3</sup> Vergleich zwischen Standard und verbessert Dijkstra [KSDS21a]

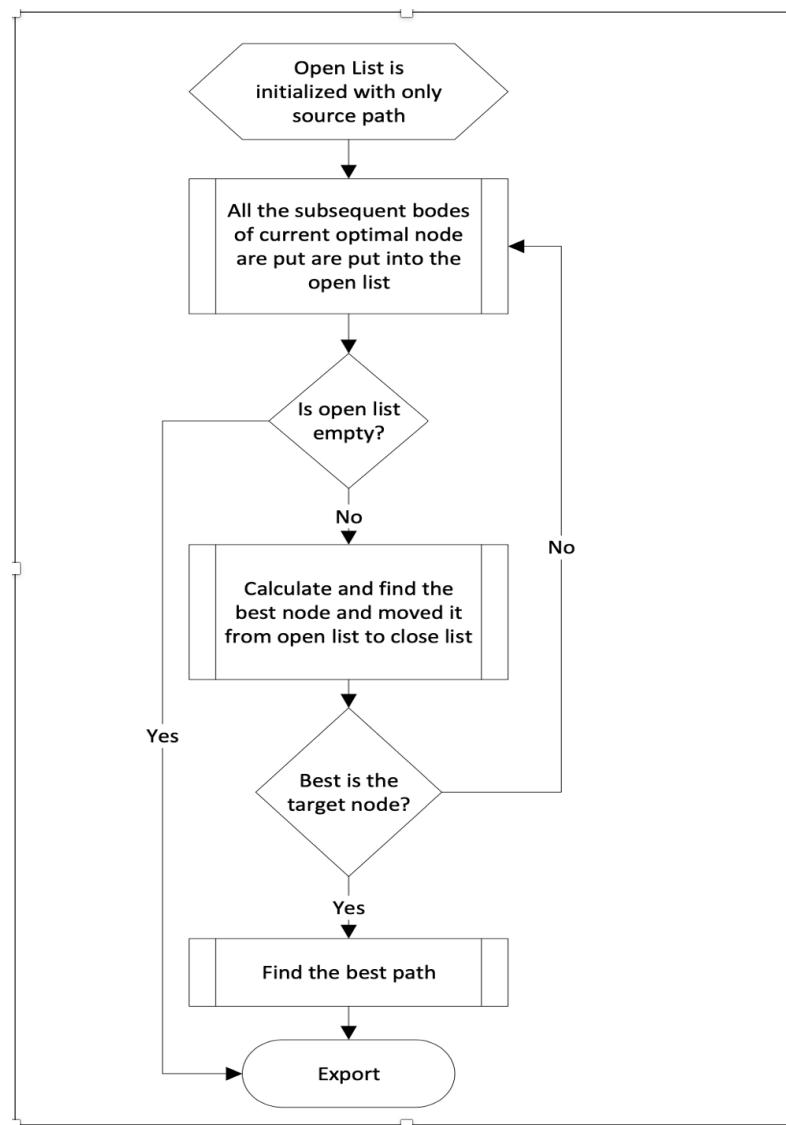


Abb. 4.3: Flussdiagramm des verbesserten Dijkstra für die Pfadplanung[KSDS21a].

#### 4.2.2 Verbesserter Dijkstra

Die Autoren des Artikels<sup>4</sup> haben der verbesserte Dijkstra-Algorithmus so beschrieben: „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.“

<sup>4</sup> verbesserter Dijkstra für Mobile Roboter [KSDS21a]

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“.

### 4.3 Autonome Navigation

Pfadsuchalgorithmen werden ebenfalls im Bereich des Autonomen Fahrens, oder auch bei der unbemannten Flugfahrzeuge verwendet um sichere, effiziente, kollisionsfreie und kostengünstige Wege von Start zum Ziel zu führen, was die Wahl des richtigen Pfadsuchalgorithmen zu einer wichtigen Aufgabe macht. Es hängt unter anderen Faktoren die Geometrie des Fahrzeugs von dieser Wahl ab [KSDS21b]. Mit der zunehmenden Verbreitung von autonomen Fahrzeugen, die immer mehr Wegfindung und -planung erfordert, sind Pfadsuchalgorithmen zu einem neuen Schwerpunkt der autonomen Steuerung geworden [KSDS21b]. Da mobile Roboter in vielen Anwendungen eingesetzt werden, haben Forscher Methoden entwickelt, um die Anforderungen an mobile Roboter effektiv erfüllen zu können und einige Herausforderungen für die Umsetzung einer vollständig oder teilweise autonomen Navigation in unübersichtlichen Umgebungen zu bewältigen [KSDS21b]. So wird die Wahl des richtigen Pfadplanungsalgorithmus von der kinematischen Bewegungsgestaltung des Roboters/Fahrzeugs, den zur Verfügung stehenden Rechenressourcen, sowie der sensorischen Ausstattung des Fahrzeugs bestimmt [KSDS21b]. Die Leistung und Komplexität des verwendeten Algorithmus hängt auch vom Anwendungsfall ab [KSDS21b].

Somit gibt es nicht *den perfekten Pfadsuchalgorithmus für Autonomes Fahren*, aber es finden viele verschiedene Algorithmen eine Anwendung in der autonomen Navigation.

## Zusammenfassung und Ausblick

In den letzten Jahren hat die Pfadplanung immer mehr an Relevanz gewonnen, zum Beispiel wird beim autonomen Autofahren immer mehr Forschung im Bereich der Pfadplanungsalgorithmen betrieben. [KSDS21b]

- Bevor ein Programm mit der Suche nach der besten Lösung (dem besten Weg) beginnen kann, muss erst ein Ziel deklariert und das Problem (Umgebung) genau definiert werden.
- Suchalgorithmen behandeln Zustände und Aktionen atomar: Sie berücksichtigen keine interne Struktur, die sie besitzen könnten [RN10, 108,109].
- Uninformierte Pfadsuchalgorithmen haben nur Zugriff auf die grundlegenden Problemdefinition. Die wichtigsten Algorithmen sind die folgenden:
  - Die Breitensuche (Kapitel 2.2.1) expandiert zuerst die flachsten Knoten. Sie ist vollständig, optimal für einheitliche Pfadkosten, hat jedoch eine exponentielle Raumkomplexität [RN10, 81].
  - Die Tiefensuche (Kapitel 2.2.2) expandiert zuerst den tiefsten nicht expandierten Knoten. Sie ist weder vollständig noch optimal, hat aber eine lineare Raumkomplexität [RN10, 85,86].
  - Die iterative Vertiefungssuche (Kapitel 2.2.2) 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 [RN10, 85,86].
  - Der Greedy Dijkstra-Algorithmus (Kapitel 2.3) der zwar bei der blinden Suche Zeit vergeudet, aber dafür optimal ist und eine Trefferquote von 100% hat [KSDS21b].
  - Die Optimierung durch eine bidirektionale Suche kann die Zeitkomplexität enorm reduzieren, sie ist allerdings nicht immer anwendbar und kann viel Speicherplatz beanspruchen.
- Informierte Suchmethoden basieren auf heuristischen Funktionen, die die Kosten einer Lösung schätzen

- Der generische Best-First-Search-Algorithmus (Kapitel 3) wählt den nächsten Knoten gemäß einer Bewertungsfunktion aus.
- Der Greedy Best-First-Search-Algorithmus (Kapitel 3) expandiert Knoten nach einem minimalem heuristischem Funktionswert. Er ist zwar nicht optimal, aber oft effizient [RN10, 108,109].
- A\*-Suche (Kapitel 3.2) expandiert Knoten mit minimalem Heuristischen Funktions- und Pfadkostenwerten. A\* ist vollständig und optimal, vorausgesetzt die heuristische Funktion ist zulässig.
- ALT-Algorithmen (Kapitel 3.3.1), die aufgebaut auf A\* durch Preprocessing noch optimiertere Ergebnisse erzeugt.
- Reach-based Pruning (Kapitel 3.3.2), welches den Dijkstra-Algorithmus um eine Metrik erweitert und dadurch optimiert.

Alle diese Algorithmen haben diverse Vor- und Nachteile und daraus ergeben sich verschiedene Anwendungsbereiche, auf welche wir in Kapitel 4 gesondert eingegangen sind.

---

## Literaturverzeichnis

- AIS<sup>+</sup>20. ABUSALIM, SAMAH, ROSZIATI IBRAHIM, MOHD SARINGAT, SA-PIEE JAMEL und JAHARI WAHAB: *Comparative Analysis between Dijkstra and Bellman-Ford Algorithms in Shortest Path Optimization*. IOP Conference Series: Materials Science and Engineering, 917:012077, 09 2020.
- Esr22. *Esri GIS Dictionary: Pathfinding*. online, Juli 2022. <https://support.esri.com/en/other-resources/gis-dictionary/term/7f861382-d88c-4828-8272-c3da4bdc8fa6>.
- FGK<sup>+</sup>21. FOEADA, DANIEL, ALIFIO GHIFARIA, MARCHEL BUDI KUSUMAA, NOVITA HANAFIAHB und ERIC GUNAWANB: *A Systematic Literature Review of A\* Pathfinding*. Elsevier B.V, 2021.
- 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.
- HAMTMA20. HAMID ALI, ABED ALASADI, AZIZAND MOHAMMED TALIB, DHI-YA MOHAMMED und ABDULMAJE AHMED: *A Network Analysis for Finding the Shortest Path in Hospital Information System with GIS and GPS*. Journal of Network Computing and Applications (2020) 5: 10-22 Clausius Scientific Press, Canada, 2020.
- HG12. HUANG, HAOSHENG und GEORG GARTNER: *Collective intelligence-based route recommendation for assisting pedestrian wayfinding in the era of Web 2.0*. Journal of Location Based Services, 6:1–21, 03 2012.

- 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.
- Jav19. JAVAID, ADEEL: *Research on Optimal Path based on Dijkstra Algorithms*. SSRN Electronic Journal, 3, 2019.
- KSDS21a. KARUR, KARTHIK, NITIN SHARMA, CHINMAY DHARMATTI und JOSHUA E. SIEGEL: *A Survey of Path Planning Algorithms for Mobile Robots*. Vehicles, 3(3):448–468, 2021.
- KSDS21b. KARUR, KARTHIK, NITIN SHARMA, CHINMAY DHARMATTI und JOSHUA E. SIEGEL: *A Survey of Path Planning Algorithms for Mobile Robots*. Vehicles, 3(3):448–468, aug 2021.
- MKL19. MEHTA, HEEKET, PRATIK KANANI und PRIYA LANDE: *Google Maps*. International Journal of Computer Applications, 178(8):41–46, may 2019.
- MS20a. MUKHLIF, FADHIL und ABDU SAIF: *Comparative Study On Bellman-Ford And Dijkstra Algorithms*. Int. Conf. Comm. Electric Comp. Net., 2020.
- MS20b. MUKHLIF, FADHIL und ABDU SAIF: *Comparative Study On Bellman-Ford And Dijkstra Algorithms*. International Conference on Communication, Electrical and Computer Networks, 02 2020.
- PAG<sup>+</sup>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.
- RN10. RUSSELL, STUART J. und PETER NORVIG: *Artificial Intelligence: A Modern Approach*. Prentice Hall, Third Auflage, 2010.
- VP14. VAIBHAVI, PATEL und PROF. CHITRABAGGAR: *A survey paper of Bellman-ford algorithm and Dijkstra algorithm for finding shortest path in GIS application*. International Journal of P2P Network Trends and Technology (IJPTT), 2014.
- Wik. *Wikipedia - Online Lexikon*. [de.wikipedia.org/wiki/Pruning](https://de.wikipedia.org/wiki/Pruning).
- ZG13. ZHOU, MINHANG und NINA GAO: */Understanding Dijkstra Algorithm*. Proceedings of the 3rd International Conference on Mechatronics Engineering and Information Technology, 3, 01 2013.

---

## Arbeitsverteilung

### Teilnehmer 1: Mohammed Salih Mezraoui

Inhalte:

- Kapitel 2.1: Was ist Pfadplanung?
- Kapitel 2.3: Dijkstra Algorithmus
- Kapitel 2.4: Bellman-Ford-Algorithmus
- Kapitel 4.1: GIS(Geoinformationssystem)
- Kapitel 4.2: Mobile Roboter mit verbessertem Dijkstra-Algorithmus

### Teilnehmer 2: David Gruber

Inhalte:

- Kapitel 1: Einleitung und Problemstellung
- Kapitel 2.2: Uninformierter Ansatz
- Kapitel 4.3: Autonome Navigation
- Kapitel 5: Zusammenfassung und Ausblick

### Teilnehmer 3: Marius Müller

Inhalte:

- Kapitel 3: Optimierungsstrategien