Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Entwicklung von Algorithmen zur Planung der Wege von fahrerlosen Transportsystemen in einem Logistik-Warehouse

Dirk Braunschweiger

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Dipl.-Inf. Jan Königsberger
Beginn am:	1. März 2017
Beendet am:	1. September 2017
CR-Nummer:	F.2.2, G.2.2, H.4.2

Kurzfassung

In der Automobilindustrie ist in den letzten Jahren die Anforderungen an die Logistik-Warenhäuser gestiegen. Die steigende Individualisierung von Fahrzeugen ist der Grund dafür. Um die Anforderungen erfüllen zu können, werden in modernen Logistik-Warenhäusern die Waren durch fahrerlose Transportfahrzeuge transportiert. Es existieren viele Algorithmen zur Berechnung des kürzesten Weges für einzelne Fahrzeuge. Diese können in Warenhäusern mit vielen Fahrzeugen nicht eingesetzt werden, da es zu Staus, Deadlocks oder Kollisionen kommen kann. Es existieren bereits Algorithmen, die versuchen diese Probleme zu lösen. Wenige dieser Algorithmen wurden bisher auf die Praxistauglichkeit getestet. Die Algorithmen werden oft mit wenigen Fahrzeugen oder auf kleinen Straßennetzen getestet.

Diese Arbeit stellt Algorithmen zur Berechnung von Wegen für mehrere Fahrzeuge vor und analysiert diese anschließend. Die Performanz der Algorithmen wird anhand realer Szenarien aus der Automobilindustrie gemessen. Dafür werden zuerst Straßennetze basierend auf echten Lagerhallen erstellt. Anschließend wird in verschiedenen Benchmarks die Performanz ausgewählter Algorithmen miteinander verglichen. Basierend auf den besten Algorithmen wird ein neuer Algorithmus entwickelt und mit bestehenden Algorithmen verglichen. Der neue Algorithmus benötigt weniger Rechenzeit und berechnet kürzere Wege. Die Ergebnisse werden abschließend mithilfe einer Simulations-Software validiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Fahrerlose Transportfahrzeuge (FTF) führen in modernen Logistik-Warenhäusern die Transportaufträge durch. Die Mitarbeiter werden durch diese Roboter unterstützt. Ein FTF holt ein Regal ab und transportiert es zu einer Entnahmestation. Dort kann der Mitarbeiter das benötigte Teil aus dem Regal nehmen. Anschließend wird das Regal zurück auf einen freien Stellplatz gebracht [WDM08]. Im Jahr 2012 hat der Online-Lieferant Amazon die Firma *Kiva Systems* gekauft und nutzt seitdem die Roboter *Kiva* in den Warenhäusern, um die Mitarbeiter zu unterstützen [Kir13]. Amazon hat in folgenden Jahren einige Patente, die ihre automatisierten Warenhäuser beschreiben, beantragt [JSHH07; Kaw16; NVB17]. Der Aufbau und die Abläufe in den Warenhäusern wurden in vielen verschiedenen Veröffentlichungen untersucht und optimiert [Kar13].

Nicht nur bei großen Online-Versandhäusern, sondern auch in der Automobilindustrie stiegen die Anforderungen an die Logistik-Warenhäuser in den letzten Jahren. Durch höhere Individualisierung der einzelnen Fahrzeuge werden mehr Bauteile in der Montage benötigt [KLB17]. Dieser wachsenden Anforderung wird durch steigende Automatisierung begegnet [Bau17]. Zu den Abläufen in einem Warenhaus mit fahrerlosen Transportfahrzeugen gehört insbesondere auch das Planen der Wege für die Transportaufträge.

1.1 Motivation

Für die Berechnung der kürzesten Wege existieren viele Algorithmen. Dijkstra und A* sind zwei bekannte Beispiele dafür. Diese Algorithmen berücksichtigen allerdings nur die zugrunde liegende Struktur des Straßennetzes, nicht aber die anderen Transportfahrzeuge. In einem Warenhaus sind, je nach Größe, hunderte Fahrzeuge unterwegs [WDM08]. Daher spielt die Position der anderen Fahrzeuge bei der Planung einer Route eine wichtige Rolle.

Das Problem der Planung von Pfaden für mehrere Fahrzeuge wurde in mehreren Studien untersucht [HS06; Rya08; YL16]. Die verwendeten Modelle vereinfachen oft die Realität. Es wird z. B. angenommen, dass alle FTF dieselbe Geschwindigkeit haben oder sich gleichmäßig auf der Fläche verteilen. In einem Logistik-Warenhaus können allerdings Roboter von unterschiedlichen Herstellern mit unterschiedlichen Eigenschaften zum Einsatz kommen. Zusätzlich transportieren alle FTF die Regale zu den gleichen (wenigen) Entnahmestationen. Daher kann nicht von einer gleichmäßigen Verteilung der Roboter ausgegangen werden. In einer realen Umgebung kann es zusätzlich zu dynamischen Hindernissen kommen. Ein defektes FTF kann einen Weg versperren oder eine Strecke aus anderen Gründen kurzzeitig blockiert sein.

1.2 Ziel der Arbeit

In dieser Arbeit sollen bestehende Algorithmen auf die Praxistauglichkeit untersucht und gegebenenfalls ein neuer Algorithmus, welcher den beschriebenen Anforderungen entspricht, entwickelt werden. Zur Evaluierung werden Szenarien aus der Automobilindustrie verwendet. Neben der Wegstrecke der Fahrzeuge spielt die Reihenfolge der Transportaufträge eine große Rolle. Dieser Aspekt eines Logistik-Warenhauses wird in dieser Arbeit nicht untersucht. Der Fokus dieser Arbeit liegt auf den Wegstrecken der Fahrzeuge.

1.3 Aufbau der Arbeit

Diese Arbeit stellt in Kapitel 2 Grundlagen zu Produktionsabläufen und der Wegplanung vor. Für ein besseres Verständnis des Umfelds werden zuerst produktionstechnische Begriffe und Abläufe erklärt. Im zweiten Teil des Kapitels werden informationstechnische Begriffe und Algorithmen erläutert, die für das Verständnis der folgenden Kapitel notwendig sind. In Kapitel 3 werden bestehende Algorithmen zur Berechnung von Wegen für mehrere Roboter vorgestellt und in drei Gruppen klassifiziert. Anschließend werden die Algorithmen, basierend auf der Performanzanalyse der Autoren, verglichen. Kapitel 4 beschreibt Kriterien nach denen Algorithmen für eine genauere Analyse ausgewählt wurden. Außerdem wird auf Besonderheiten der Implementierungen für die Analyse der Algorithmen eingegangen. Am

Ende des Kapitels werden die Ergebnisse der Tests diskutiert. Im Kapitel 5 wird ein neuer Algorithmus für das reale Szenario beschrieben. Die Ergebnisse des neuen Algorithmus werden in Abschnitt 5.2 mit den anderen Algorithmen verglichen und validiert. Das Kapitel 6 fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf zukünftige Schritte.

2 Grundlagen

In diesem Kapitel werden Grundlagen beschrieben, welche das Verständnis dieser Arbeit fördern sollen. Zuerst werden produktionstechnische Grundlagen erklärt, anschließend informationstechnische.

2.1 Produktionstechnische Grundlagen

Alle Bestandteile einer Fabrik und einem Logistik-Warenhaus, die für diese Arbeit relevant sind, werden in diesem Kapitel erklärt.

2.1.1 Fahrerlose Transportfahrzeuge (FTF)

Als fahrerlose Transportfahrzeuge werden Fahrzeuge bezeichnet, die Regale in der Fabrik oder Lagerhalle transportieren. Das Ausführen der Fortbewegung wird ohne einen menschlichen Fahrzeugführer bewerkstelligt. Diese Fahrzeuge werden auch als Roboter oder AGV (engl. Automated Guided Vehicle) bezeichnet.

Die Abbildung 2.1 zeigt das Kiva FTF, das bei Amazon zum Einsatz kommt. Das 40 cm hohe Fahrzeug kann bis zu 320 KG schwere Regale anheben [Tim14]. Abbildung 2.2 zeigt ein FTF der arculus GmbH. Diese ca. 20 cm hohe FTF kann bis zu einer Tonne anheben und wird in der Automobilindustrie eingesetzt [Hop17].

Für den Transport der Regale existieren verschiedene Methoden. Die oben genannten FTF fahren unter das Regal und heben es an. Um ein Regal anzuheben, gibt es verschiedene Methoden. Die FTF von Amazon heben Regale an, indem sie sich auf der Stelle drehen und eine Plattform wie einen Korkenzieher in die Höhe schrauben [Ger15]. Der Arculee

Abbildung 2.1: Das FTF Kiva [Ger15]

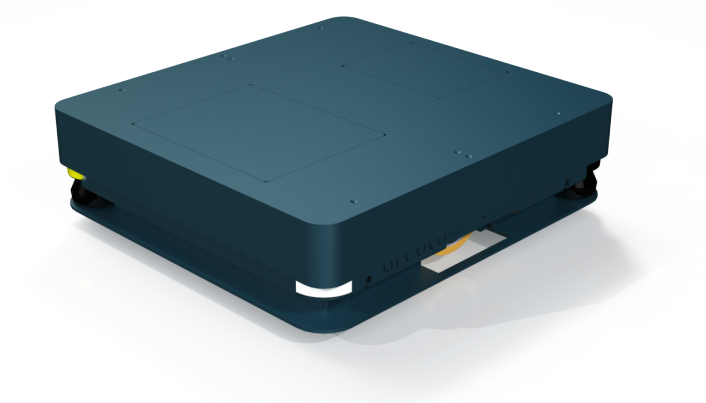

Abbildung 2.2: Das FTF Arculee. Quelle: arculus GmbH

hebt das Regal hydraulisch an. Alternativ zum Anheben haben Regale mancher Hersteller Räder und werden wie Waggons von einem Zug gezogen. Das FTF besitzt dafür eine Art Anhängerkupplung [Hop17].

Die FTFs bekommen von übergeordneten Systemen die Transportaufträge. Diese legen fest, welches Regal von welchem FTF abgeholt und transportiert werden soll. Damit das FTF der berechneten Route folgen kann, benötigt jedes FTF Sensoren zur Bestimmung der eigenen Position. Dafür wird z. B. indoor GPS verwendet.

Neben den Sensoren zur Ortung des Fahrzeuges besitzen die FTF weitere Sensoren, wie z. B. Sensoren die Kollisionen mit Personen erkennen können. Diese Sensoren sind für Fahrzeuge notwendig, wenn diese, in einem für Personen zugänglichen Bereich, fahren. Im Fall einer bevorstehenden Kollision wird vom FTF eine Notbremsung durchgeführt. Man spricht hierbei von personensicheren FTF. In diese Gruppe fällt auch das oben bereits erwähnte Arculee.

Das Kiva FTF besitzt diese Sensoren nicht [Ein17]. In Warenhäusern, in denen es einen abgetrennten Bereich für die Fahrzeuge gibt, sind diese Sensoren nicht notwendig. Es muss allerdings sichergestellt werden können, dass sich keine Person in dem Bereich aufhält.

2.1.2 Regalblock

Als Regalblock wird in dieser Arbeit ein zusammenhängender Bereich von Regalen bezeichnet. Dieser Bereich ist durch Straßen abgeschlossen. In ihm hat jedes Regal einen festen Standplatz und ist von der Straße zu erreichen. In Abbildung 2.3 ist ein Regalblock abgebildet. Die grauen Kreise stellen die Regale dar. Die Straßen sind durch Pfeile und die Kreuzungen als blaue Kreise dargestellt.

2.1.3 Entnahmestation

In einem Logistik-Warenhaus werden die Regale von FTF zu den Entnahmestationen transportiert. An diesen Stationen entnimmt ein Mitarbeiter das benötigte Teil aus dem Regal. Anschließend wird das Regal von dem FTF zurück auf seinen Standplatz im Regalblock transportiert. An den Entnahmestationen gibt es einen Wartebereich für die Regale der folgenden Teile. Dort warten die FTF mit dem entsprechenden Regal. Dadurch entsteht bei dem Mitarbeiter keine Wartezeit. Wenn das aktuelle Teil entnommen wurde und das Regal weiter transportiert wird, rückt sofort das nächste FTF mit dem nächsten Regal nach.

2.1.4 Quellen und Senken

Als Quellen und Senken werden Start- und Zielknoten für die Routenplanung bezeichnet [Len93]. Wenn ein Regal zu einer Entnahmestation transportiert wird, wird der Regalstandplatz als Quelle und die Entnahmestation als Senke bezeichnet.

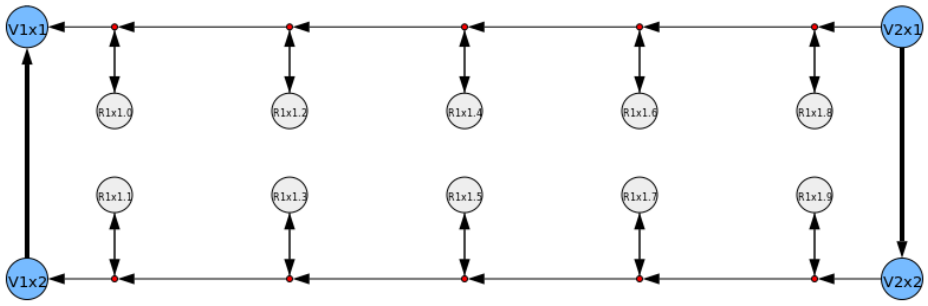

Abbildung 2.3: Graph G eines Regalblocks mit 10 Regalen und vier Straßenkreuzungen

2.2 Informationstechnische Grundlagen

In diesem Abschnitt werden informationstechnische Grundlagen erklärt, die für das Verständnis der Algorithmen notwendig sind.

2.2.1 Graph

Ein Graph ist eine abstrakte Struktur, die den Zusammenhang zwischen Objekten darstellt. Die Objekte werden als Knoten bezeichnet. Kanten stellen die Beziehung der Knoten dar. Mathematisch wird ein Graph mit $G = (V, E)$ beschrieben. V ist die Menge der Knoten und E die Menge der Kanten [Die12].

Graphen können unterschiedliche Eigenschaften haben. Die Kanten in Graphen können ungerichtet oder gerichtet sein. Ungerichtete Kanten stellen nur eine Beziehung zwischen den Knoten dar. Gerichtete Kanten geben zusätzlich eine Reihenfolge der Knoten vor. Abbildung 2.3 zeigt einen gerichteten Graphen. Die Knoten werden als Kreis dargestellt. Die Pfeile stellen die gerichteten Kanten dar. $V1x2$ ist der Vorgänger von $V1x1$.
Eine weitere Eigenschaft von Graphen kann das Gewicht der Kanten sein. Dieses Kantengewicht, das meistens der Länge der Kante entspricht, wird von Algorithmen zur Berechnung von Wegen verwendet, um den kürzesten Weg berechnen zu können.

In dieser Arbeit werden gerichtet Graphen mit Kantengewichten verwendet, um das Layout der Fabrik zu beschreiben. Jeder Fahrweg in der Fabrik entspricht einer gerichteten Kante im Graphen. Daher dürfen die Fahrzeuge die Kante nur in einer Richtung befahren, wie

beispielsweise bei Einbahnstraßen. In den Graphen werden Regale, Stationen und Kreuzungen der Fahrwege durch Knoten dargestellt. Das Kantengewicht entspricht der Länge der Fahrwege.

2.2.2 Graph-Algorithmen

In den folgenden Abschnitten werden die Algorithmen zur Wegefindung erklärt. Diese Algorithmen werden z. B. von Openstreetmap eingesetzt, um den kürzesten Weg zu berechnen [Kar17]. Auch die Algorithmen zur Berechnung der Wege für die FTF basieren auf diesen Algorithmen.

Dijkstra

Der Dijkstra-Algorithmus berechnet kürzeste Wege von einem Knoten zu allen anderen Knoten in einem Graphen. Dafür wird für jeden Knoten die Distanz zum Startknoten und der Vorgänger berechnet [HPS08]. Initial hat kein Knoten einen Vorgänger. Jedem Knoten wird während der Berechnung die Distanz zum Startknoten zugewiesen. Initial ist die Distanz des Startknoten 0 und die aller anderen ∞. Es wird eine *Liste* der zu überprüfenden Knoten gepflegt. Zu Beginn wird nur der Startknoten in die *Liste* eingefügt.

Anschließend werden folgende Schritte durchgeführt bis die *Liste* keine Knoten mehr enthält [HPS08]:

1. Wähle Knoten n mit geringster Distanz aus der *Liste*.
2. Berechne für jeden Nachbarknoten v von n die Distanz zum Startknoten. Dazu wird das Kantengewicht der Kante zwischen n und v auf die Distanz des Knoten n addiert.
3. Ist die berechnete Distanz kleiner als die für v gespeicherte Distanz, aktualisiere diese Distanz. Außerdem wird n als Vorgänger für v gespeichert.
4. Wenn die Distanz aktualisiert wurde: Füge v in die *Liste* hinzu.

Wenn nur der kürzeste Weg zwischen zwei bestimmten Knoten berechnet werden soll, kann der Algorithmus in Schritt 1 abgebrochen werden, sobald der Zielknoten ausgewählt wurde. Anschließend wird der kürzeste Weg durch die gespeicherten Vorgänger rekonstruiert. Die Schritte 2 und 3 werden in der Literatur als Knotenexpansion bezeichnet [HPS08].

A*

Eine Variation des Dijkstra-Algorithmus ist der A*. Er verwendet eine Heuristik (Schätzfunktion), um zielgerichtet zu suchen. Der Distanzwert errechnet sich bei A* durch den Distanzwert des Vorgängers, dem Kantengewicht und dem Wert der Heuristik (vgl. Schritt 2 von Dijkstra). Für eine Routensuche im zweidimensionalen Raum wird häufig der euklidische Abstand zum Ziel als Heuristik verwendet. Dadurch werden Knoten, die geometrisch näher beim Zielknoten liegen, zuerst aus der *List* ausgewählt [HPS08].

2.2.3 Solver

Das mathematische Problem den kürzesten Weg zwischen zwei Knoten zu bestimmen, kann auch durch ein Gleichungssystem beschrieben werden. Dieses Gleichungssystem kann von Programmen, sogenannten Solvern, gelöst werden. Solver, wie z. B. GLPK (GNU Linear Programming Kit), COIN-OR LP, Gurobi und CPLEX, können Gleichungssysteme lösen oder diese optimieren. Die ersten beiden Solver sind Open Source und frei verfügbar. Die letzten beiden sind kommerzielle Programme. Die kommerziellen Solver sind performanter als die Open Source Solver [MT12].

Das Problem optimale Wegstrecken für mehrere FTF zu bestimmen kann auch als mathematisches Problem und durch ein Gleichungssystem beschrieben werden. Das Gleichungssystem kann durch Solver auf kürzeste Wegstrecke oder minimale Fahrzeit aller Roboter optimiert werden.

2.2.4 Deadlock

In dieser Arbeit beschreibt ein Deadlock eine Situation, in der mehrere FTF sich gegenseitig blockieren. Jedes FTF wartet darauf, dass das FTF, das den Weg blockiert weiterfährt und der belegte Knoten frei wird.

Der Begriff stammt aus dem Umfeld von Betriebssystemen. Dort wird von einem Deadlock gesprochen, wenn zwei oder mehr Prozesse gegenseitig zyklisch aufeinander warten [SGG06].

Das folgende Beispiel veranschaulicht dies. Prozess $P1$ verwendet z. B. die Ressource $R1$ und benötigt $R2$, um die Berechnungen zu beenden. Die Ressource $R2$ wird allerdings bereits von Prozess $P2$ verwendet und dieser Prozess benötigt $R1$, um seine Berechnungen zu beenden. In diesem Beispiel wartet $P1$ auf $P2$ und gleichzeitig $P2$ auf $P1$.

2.2.5 SAT

SAT ist die Abkürzung des englischen Worts *satisfiability* und beschreibt das Erfüllbarkeitsproblem der Aussagenlogik [ZM02]. Das Erfüllbarkeitsproblem prüft, ob eine Formel erfüllbar ist. Es wird nicht versucht eine Lösung für das Problem zu finden. Es besagt, nur ob es eine Lösung für das Problem gibt.

3 Klassifizierung von Algorithmen zur Pfadplanung

Alle FTF in einem Logistik-Warenhaus benötigen einen Weg von Start zum Ziel um ihren Transportauftrag erledigen zu können. In der Literatur werden viele Algorithmen für die Berechnung dieser Wege beschrieben. Diese werden in diesem Kapitel vorgestellt und klassifiziert. Bereits 1988 wurden drei Möglichkeiten zur Verkehrskontrolle von FTF beschrieben [Gro88].

- Begrenzungen für Wege, damit es genau einen Weg von jedem Start zu jedem Ziel gibt.
- Auswahl der Wege für die FTFs wird von einer zentralen Instanz getroffen.
- FTF sucht sich im Falle mehrerer möglicher Wege den Weg eigenständig aus.

Die erste Möglichkeit ist sehr restriktiv und verhindert das Planen von optimalen Routen zwischen Start und Ziel. Dieser Ansatz wird daher in dieser Arbeit nicht weiter vertieft. Es wird argumentiert, dass die Berechnung der Fahrwege auf einer zentralen Instanz, aufgrund der hohen Komplexität, nicht möglich ist. Der Anstieg der Rechenleistung in den letzten Jahren ermöglicht dieses inzwischen. Dieser Ansatz wird in vielen Veröffentlichungen diskutiert. Die dritte Möglichkeit beschreibt, dass Roboter selbst die Routenplanung übernehmen. Es gibt also keine zentrale Instanz die für alle Roboter die Wege berechnet, sondern die Entscheidungen werden direkt auf den Robotern getroffen.

Diese Arbeit teilt bestehende Algorithmen in folgende drei Gruppen auf:

Einzelweg-Algorithmen berücksichtigen bestehende Pfade anderer Roboter bei der Planung des eigenen Pfads. Die Pfade der anderen Roboter bleiben unverändert. Die Berechnungen werden zentral durchgeführt.

Mehrweg-Algorithmen planen die Pfade aller Roboter gleichzeitig auf einer zentralen Instanz. Bei der Neuberechnung des Pfades für einen Roboter werden die Pfade aller Roboter neu berechnet.

Kooperative Algorithmen versuchen Kollisionen durch Kommunikation der einzelnen Roboter aufzulösen. Die Entscheidung welchen Weg der Roboter fährt wird vom Roboter selbst getroffen.

3.1 Einzelweg-Algorithmen

In diesem Abschnitt wird ein bestehender Algorithmus vorgestellt, der den Weg für einen Roboter berechnet. Der Algorithmus berücksichtigt alle bereits geplanten Wege der anderen FTF, verändern diese jedoch nicht.

3.1.1 M*

Der Algorithmus M* wurde von Wagner und Choset vorgestellt[WC11]. Er basiert auf A*. Der Algorithmus prüft bei der Betrachtung von allen Nachbarknoten, ob diese Knoten durch einen anderen Roboter belegt sind. Die Wegplanung berücksichtigt nur freie Nachbarn. Alle belegten Knoten werden ignoriert, somit wird ein alternativer Pfad gesucht und Kollisionen vermieden. Es wird eine Kollisions-Liste gepflegt und bei jeder Expansion aktualisiert. Mithilfe dieser Liste kann schneller geprüft werden, ob ein Knoten belegt ist. Da alle Knoten, die durch ein anderes Fahrzeug belegt sind, ignoriert werden, kann nicht für jeden Roboter ein Weg gefunden werden.

Die Autoren beschreiben auch weitere Varianten des Algorithmus. Der *Inflated M** verkleinert den Suchraum von M* durch das Verwenden einer Heuristik (untere Grenze) für die Distanz zum Ziel. Dadurch wird der Suchraum stärker auf das Ziel fokussiert und das Ziel schneller erreicht. Eine weitere Variante beschreibt, wie beim Planen eines Weges bei der Kollisionsprüfung nur Roboter betrachtet werden, die sich im gleichen Kartenbereich befinden. In der Analyse der Autoren wurde mit M* für 20 Roboter in unter 10 Sekunden mit ca. 80 % Erfolg eine kollisionsfreie Route gefunden. Die Zusammenlegung beider Varianten ermöglicht das Berechnen von Wegen für 40 Roboter mit einer Erfolgsquote von über 90 %.

3.2 Mehrweg-Algorithmen

Dieser Abschnitt beschreibt Algorithmen, die für alle Roboter die Pfade gleichzeitig berechnen. Es werden also immer alle Pfade für alle Roboter gleichzeitig berechnet. Diese Berechnungen werden von einer zentralen Instanz durchgeführt.

3.2.1 Optimal Multirobot Path Planning (OMPP)

Die Autoren Yu und LaValle beschreiben in mehreren Veröffentlichungen [YL16; YR15] einen Algorithmus zum Planen von kollisionsfreien Wegen für mehrere Roboter. Die grundlegende Idee ist, das Problem der kürzesten Wegesuche auf einen Flussgraphen zu übertragen. Diese Flussgraphen können durch Gleichungssysteme gelöst werden. Das Lösen des Gleichungssystems ergibt ein optimales Ergebnis. Die Autoren verwenden zum Lösen den Gurobi Optimizer. Gurobi ist ein kommerzieller Löser für mathematische Probleme[1]. Das Gleichungssystem kann für zwei Werte optimiert werden. Es kann die kürzeste Fahrstrecke jedes Roboters bestimmen oder Wege mit der minimalen Fahrzeit aller Roboter berechnen. In dieser Arbeit wird nur letztere Berechnung betrachtet. Die Anzahl der Gleichungen nimmt mit steigender Roboter Anzahl und steigender Knotenanzahl sehr stark zu. Daher steigt auch die Berechnungszeit. Aus diesem Grund haben die Autoren in einer weiteren Veröffentlichungen beschrieben, wie sich der Suchraum des Algorithmus, inklusive des Gleichungssystems, verkleinern lässt [YR15]. Die Idee ist, den Algorithmus mehrere Teilstrecken ausrechnen zu lassen. Zuerst wird für jeden Roboter der kürzeste Weg vom Startknoten zum Zielknoten gesucht. Diese Wege werden in der Mitte geteilt. Falls zwei Mittelpunkte kollidieren sollten, ist für einen der beiden Roboter ein benachbarter Knoten zu wählen. Für beide Teilabschnitte ist die optimale Lösung mithilfe des Gleichungssystems zu bestimmen. Dieses Verfahren wird von den Autoren als *k-way split* bezeichnet. *k* gibt an, in wie viele Teile, die kürzesten Strecken geteilt werden sollen. Durch das Fixieren des Mittelpunkts sinkt die Optimalität der Ergebnisse. Die Optimalität ist ein Faktor, um den das Ergebnis zum optimalen Ergebnis abweicht. Das Berechnen eines Weges mit $k = 0$ wird von den Autoren als optimales Ergebnis definiert.

[1] http://www.gurobi.com/products/gurobi-optimizer

Für die Analyse wurden Berechnungen auf einem Grid ausgeführt. Um Hindernisse zu simulieren, wurde eine gewisse Anzahl an Knoten aus dem Grid gelöscht. Für 0 % bis 25 % gelöschter Knoten werden Berechnungen durchgeführt und die Ergebnisse verglichen.

Die Ergebnisse der Autoren zeigen, dass für 100 Roboter in ca. 12 Sekunden auf einem voll besetzten Grid eine Lösung gefunden wird. Wenn 25 % der Knoten im Grid fehlen, werden in der gleichen Zeit nur noch 40 Wege berechnet. Durch *k-Way Split* sinkt die Berechnungszeit und die Optimalität. Bei einem k-Wert von 2 können Wege für 120 Roboter berechnet werden. Diese Wege sind maximal um den Faktor 1.02 länger. Mit einem k-Wert von 16 können Wege für 75 bis 300 Roboter in unter 100 Sekunden berechnet werden, je nachdem wie viele Knoten aus dem Grid entfernt wurden. Die Strecken sind jedoch um Faktor 1.8 länger.

3.2.2 Cobopt

In der Veröffentlichung [Sur12] wird beschrieben wie mit einem SAT-Löser die späteste Ankunftszeit schrittweise verbessert werden kann. Zuerst wird mit einem anderen Algorithmus eine nicht optimale Lösung für das Problem bestimmt. Anschließend wird iterativ Cobopt angewendet, um das Ergebnis zu verbessern. Die Berechnungen sind aufwendig und benötigen viel Rechenzeit. Die späteste Ankunftszeit für 24 Roboter auf einem Grid mit 64 Knoten zu optimieren, benötigt mehr als 45 Minuten. Jede Iteration benötigt über 5 Minuten.

3.2.3 Teilgraph Pfadplanung

Die Idee für den Algorithmus von Ryan ist das Verkleinern des Suchraums. Dazu wird der Graph, auf dem die Wegesuche durchgeführt wird, in viele Teilgraphen mit bekannter Struktur aufgeteilt [Rya08].

Jeder dieser Teilgraphen hat dabei eine der folgenden Strukturen:

Stack Eine lineare Verkettung von Knoten, von denen nur der erste Knoten mehrere Nachbarn hat.

Hall Eine lineare Verkettung von Knoten, wobei jeder Knoten weitere Nachbarn haben kann.

Clique Eine Gruppe an Knoten, bei der jedes Knotenpaar durch eine Kante verbunden ist. Jeder Knoten kann dabei weitere Knoten außerhalb der Clique als zusätzlichen Nachbarn haben.

Ring Eine Anzahl an Knoten, die ringförmig miteinander verbunden sind.

Jeder Teilgraph hat bestimmte Bedingungen für den Ein- bzw. Austritt von Robotern. So können Roboter einen Stack nur in der gleichen Reihenfolge verlassen, wie sie ihn betreten haben. Die Eigenschaften der Teilgraphen werden in der Veröffentlichung [Rya08] genauer beschrieben. Da die Bedingungen bekannt sind, kann effektiv ein optimaler Weg innerhalb des Teilgraphen für jeden Roboter berechnet werden. Unter Berücksichtigung der Ein- und Austrittbedingungen lässt sich ein abstrakter Weg für jeden Roboter bestimmen. Anschließend sind, basierend auf den Bedingungen für die Bewegung innerhalb der Teilgraphen, konkrete Wege der Roboter zu berechnen. Dabei kann jeder Teilgraph parallel berechnet werden.

Der Algorithmus wurde von den Autoren in drei verschiedenen Experimenten getestet. Die ersten beiden Experimente wurden für zehn Roboter auf Graphen verschiedener Größen durchgeführt. Die Berechnungszeit lag dabei zwischen einer und zehn Sekunden. Für Experiment drei wurde ein Graph mit 113 Knoten erstellt und auf 47 Teilgraphen aufgeteilt. Diesmal wurde die Anzahl der Roboter erhöht. Für 18 Roboter liegt die Berechnungszeit zwischen 1 und 10 Sekunden. Bei 20 Robotern wird die 10-Sekunden-Marke überschritten.

3.3 Kooperative Algorithmen

Die Algorithmen in diesem Abschnitt versuchen durch kooperatives Verhalten der Roboter Kollisionen aufzulösen. Meist wird dieser Ansatz verwendet, wenn die Roboter eigenständig ihren Weg finden sollen.

3.3.1 Cooperative multi-robot path planning by heuristic priority adjustment

Die Idee des verteilten Algorithmus ist die Kommunikation zwischen den kooperativen Robotern um Konflikte gemeinsam zu lösen [RL06]. Die Entscheidung, welcher Roboter zuerst eine Engstelle passieren darf, wird durch dynamische Prioritäten geregelt. Die Dringlichkeit des Transportauftrags bestimmt dabei die Priorität eines Roboters. Ein Roboter mit niedriger Priorität nimmt auf Roboter mit höherer Priorität Rücksicht, womit die meisten Konflikte gelöst werden können. Für die Wegfindung werden zwei Karten verwendet. Die erste Karte ist statisch. Das heißt alle beweglichen Objekte und Hindernisse werden ignoriert. Sie gibt nur die Distanz zum gewünschten Ziel an. Diese Werte können mit Dijkstra berechnet werden. Beginnend mit dem Zielpunkt wird der Graph so lange expandiert, bis alle Knoten eine Distanz zugewiesen bekommen haben. In der zweiten, kleineren, Karte werden dynamische Werte gespeichert. Diese Karte enthält eine zusätzliche Zeitdimension und speichert die Positionen der beweglichen Hindernisse. Mit den Informationen aus beiden Karten wird dann der Weg berechnet. Jeder Roboter sendet seine Position und den geplanten Weg an alle anderen Roboter. Mit diesen Informationen baut jeder Roboter die zweite Karte auf.

Eine Auswertung der Laufzeit existiert nicht, da jeder Roboter seinen eigenen Weg plant und es keine zentrale Instanz gibt. Eine Simulation der Autoren war mit 100 Robotern erfolgreich. In komplexeren Szenarien finden die Roboter allerdings keine optimale Lösung. Die Ergebnisse sind 50 - 100 % schlechter als das Optimum, das durch z. B. OMPP (siehe Abschnitt 3.2.1) berechnet wurde.

3.3.2 Multi-robot path planning using co-evolutionary genetic programming

Dieser Algorithmus verwendet genetische Programmierung, um einen optimalen Weg für die Roboter zu finden [Kal12]. Die evolutionäre Entwicklung der Wege der Roboter ist dabei in zwei Algorithmen, Master und Slave genannt, aufgeteilt. Der Slave-Algorithmus optimiert die Wege der individuellen Roboter. Für jeden Roboter wird dieser Algorithmus individuell ausgeführt. Für die Pfadplanung werden die anderen Roboter bereits mit einbezogen. Dieser Algorithmus versucht nicht das gesamte System zu optimieren. Diese Aufgabe übernimmt

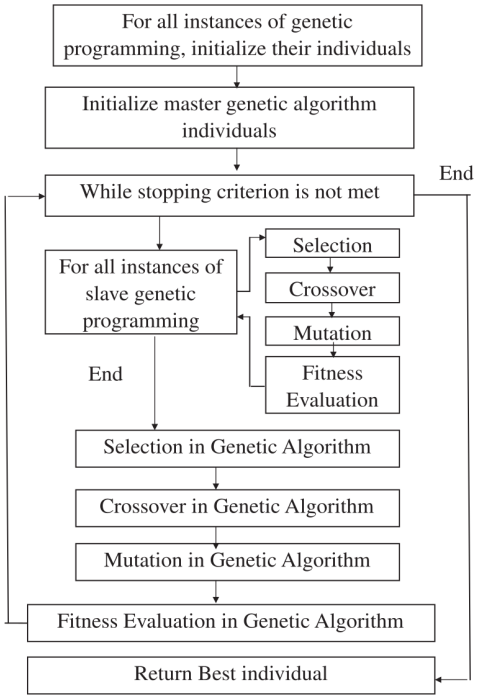

Abbildung 3.1: Der grundlegende evolutionäre Algorithmus [Kal12]

der Master-Algorithmus. Die Bewertungsfunktion verwendet die Zeit, die alle Roboter zum Ziel brauchen, als Kriterium. Zusätzlich wird geprüft, ob alle Roboter kollisionsfrei zum Ziel gekommen sind. Wenn ein Roboter sein Ziel nicht erreicht, wird die Bewertung herabgestuft.

Die Abbildung 3.1 zeigt wie die Master- und Slave-Algorithmen verbunden sind. Der Master nimmt direkten Einfluss auf die Entwicklung der Slave-Algorithmen und optimiert das Gesamtergebnis. Im Vergleich zu den bisher vorgestellten Algorithmen betrachtet dieser auch unterschiedliche Geschwindigkeiten der einzelnen Roboter. Der Geschwindigkeitsunterschied wird bei der Entscheidung, welcher Roboter auf einen anderen an einer Engstelle warten muss, berücksichtigt. Die Evaluierung des Algorithmus betrachtet allerdings nur 5 Roboter und benötigt für diese Lösung bereits 15 Minuten.

3.3.3 Reactive Navigation of Multiple Moving Agents by Collaborative Resolution of Conflicts

Die Kernidee dieses Algorithmus ist, die Geschwindigkeit der Roboter so anzupassen, dass keine Kollisionen entstehen [KHC05]. Die Wegberechnung erfolgt dabei auf den jeweiligen Robotern. Für die Lösung von Konfliktsituationen wird vorausgesetzt, dass die Roboter miteinander kooperieren. Wenn es zum Konflikt kommt, wird in drei Phasen vorgegangen. Zuerst versucht der Roboter eigenständig eine Lösung für die Kollision zu finden. Wenn dies nicht gelingt, wird in der zweiten Phase versucht durch Anpassung der Geschwindigkeiten beider Roboter die Kollision zu verhindern. Dafür kommunizieren die Roboter miteinander. Nur wenn die zweite Phase das Problem nicht lösen kann, kommt es zu Phase drei. In der dritten Phase wird die Konfliktsituation an weitere, bisher unbeteiligte, Roboter gemeldet und so versucht den Konflikt zu lösen. Ein großer Unterschied zu den anderen Algorithmen besteht darin, dass nur die Geschwindigkeiten der Roboter angepasst werden, der geplante Weg bleibt unverändert. Außerdem kann dieser Algorithmus nicht garantieren, dass ein kollisionsfreier Weg gefunden wird.

3.3.4 Fast and Memory-Efficient Multi-Agent Pathfinding

Die Autoren Wang, Botea et al. stellen den verteilten Algorithmus FAR in [WB+08] vor. Dieser besteht aus mehreren Schritten. Zuerst wird für jeden Roboter eine angepasste A*-Suche durchgeführt. Dabei werden die anderen Roboter auf der Karte ignoriert. Kollisionen in einem gerichteten Graphen können nur an Kreuzungen auftreten, wenn sich dort die Wege zweier Roboter kreuzen. Der angepasste A* bevorzugt geradlinige Routen gegenüber kurvenreicher Routen. Dadurch wird die Anzahl der sich kreuzenden Routen reduziert [WB+08]. Nachdem der Weg für einen Roboter geplant ist, startet der eigentliche Algorithmus. Dabei versucht der Roboter den ersten Knoten des berechneten Weges zu reservieren. Ist dieser belegt oder durch einen anderen Roboter bereits reserviert, wartet der Roboter bis der Knoten frei wird. Da jeder Roboter die folgenden Knoten reserviert kann es nicht zu Kollisionen kommen. In Situationen, in denen viele Roboter in einem Bereich des Graphen konzentriert sind, werden die Bewegungen der Roboter eingeschränkt. Zuerst dürfen sich Roboter nur noch horizontal bewegen. Anschließend nur vertikal. Dieses Prinzip ist mit dem Konzept einer Ampel an einer Kreuzung vergleichbar.

Wenn mehrere Roboter zyklisch aufeinander warten, wird von einem Deadlock gesprochen. FAR erkennt Deadlocks und löst diese auf, indem einer der wartenden Roboter auf einen beliebigen freien benachbarten Knoten geschickt wird. Im Anschluss wird für diesen Roboter der Rückweg zum vorherigen Knoten mittels A* geplant. Alle anderen, in den Deadlock verwickelten Roboter, können ihre Fahrt fortsetzen und der Deadlock wird aufgelöst. In der Auswertung der Autoren wird FAR mit weiteren Algorithmen verglichen. FAR benötigt weniger als 25 Sekunden, um Wege für 800 Roboter zu berechnen. Die berechneten Wegstrecken der Roboter sind dabei kürzer als die der anderen Algorithmen.

3.4 Vergleich der Algorithmen

In diesem Kapitel wurden verschiedene Algorithmen vorgestellt. Es wird für jede der drei Kategorien der vielversprechendste Algorithmus ausgewählt. In der Literatur wurden verschiedene Szenarien und Implementierungen für die Performanzanalyse verwendet. Basierend auf den Ergebnissen dieser Analysen und den folgenden Kriterien werden Algorithmen, für einen detaillierteren Vergleich, bewertet und ausgewählt.

Folgende Kriterien basieren auf realen Rahmenbedingungen aus der Industrie.

Anzahl Roboter In kleinen bis mittleren Lagerhäusern werden 30 Roboter eingesetzt. In größeren Lagerhäusern auch mehr. Daher muss der Algorithmus Wege für 30 oder mehr Roboter berechnen.

Rechenzeit Die durchschnittliche Bearbeitungszeit für einen Transportauftrag liegt bei 5 Minuten (siehe Abschnitt 4.1.1) Die Zeit für die Berechnungen der Wege müssen daher minimal sein. Für die erste Auswahl wird daher eine obere Schranke von 60 Sekunden festgelegt.

Weglänge Einige der Algorithmen achten nicht auf möglichst optimale Fahrwege um z. B. Deadlocks zu vermeiden. Sind diese Umwege um Kollisionen und Deadlocks zu verhindern gering? Berechnet der Algorithmus kurze Wegstrecken?

Deadlock Werden Deadlocks vermieden oder aufgelöst?

Erfolg Wird immer ein kollisionsfreier Weg für alle Roboter gefunden?

Grid-Größe Wurde auf einem Grid mit 100 oder mehr Knoten gearbeitet? Dieses Kriterium basiert auf der Größe eines kleinen bis mittleren Lagerhauses. Jedes Regal, jede Entnahmestation und alle Kreuzungen dazwischen, sind ein Knoten im Grid. Daher ist 100 eine sehr niedrige Grenze.

In Tabelle 3.1 werden alle vorgestellten Algorithmen anhand dieser Kriterien verglichen. Zellen mit "-" bedeuten, dass in der Veröffentlichungen keine Aussage dazu getroffen wurde.

Aus der Kategorie *Einzelweg-Algorithmen* wurde M* ausgewählt.

OMPP wurde aus der Kategorie *Mehrweg-Algorithmen* ausgewählt. Dieser Algorithmus erfüllt als einziger dieser Kategorie alle genannten Kriterien. Cobopt ist aufgrund der sehr langen Rechenzeit ausgeschieden. Das Teilgraphrouting liefert unter manchen Umständen keine kurzen Wege.

In der Kategorie *kooperative Algorithmen* wurde für die meisten Algorithmen keine Rechenzeit angegeben, da es sich hier um verteilte Algorithmen handelt. FAR wurde mit mehreren hundert FTF getestet und liefert kurze Wege. Bei den anderen Algorithmen wurden zwei oder mehr Kriterien nicht erfüllt oder es gab keine Aussage dazu. Daher wurde FAR ausgewählt.

Diese drei Algorithmen werden im folgenden Kapitel genauer miteinander verglichen.

Tabelle 3.1: Vergleich bestehender Algorithmen

Algorithmus	Rechenzeit	#Roboter	Weglänge	Deadlock	Erfolg	Grid-Größe
Einzelweg-Algorithmen						
M* (3.1.1)	Ja 10 Sekunden	Ja 40 Roboter	-	Ja	Nein	-
Mehrweg-Algorithmen						
OMPP (3.2.1)	Ja $\sim$ 12 Sekunden	Ja 100 Roboter	Ja	Ja	Ja	Ja 432 Knoten
Cobopt (3.2.2)	Nein	Nein	Ja	Ja	Ja	Nein
Teilgraph (3.2.3)	Ja 10 Sekunden	Nein 10 Roboter	Nein	Ja	-	Ja 133 Knoten
Kooperative Algorithmen						
Prioritäten (3.3.1)	-	Ja 100 Roboter	Nein	Ja	Ja	-
genetischer Algorithmus (3.3.2)	-	Nein 5 Roboter	Ja	Ja	Ja	Nein < 100 Knoten
Reactive Navigation (3.3.3)	-	Ja 30 Roboter	Ja	-	Nein	-
FAR (3.3.4)	Ja $\sim$ 15 Sekunden	Ja 1600 Roboter	Ja	Ja	Ja	Ja 2 Mill. Knoten

4 Benchmark und Performanzanalyse

Aus den in Kapitel 3 beschriebenen drei Gruppen von Pfadplanungsalgorithmen wurde jeweils der vielversprechendste Algorithmus ausgewählt und die Performanz genauer getestet. In diesem Kapitel werden die Straßennetze, die darauf basierenden Szenarien und die Bewertungskriterien beschrieben. Anschließend werden die Ergebnisse dargestellt und diskutiert.

Straßennetze

In den Abbildungen 4.1 bis 4.3 werden die Straßennetze gezeigt. Die Entnahmestationen sind mit *S* und einer fortlaufenden Zahl bezeichnet. Die Regale sind in grau dargestellt. Jede Kante in den Graphen stellt einen Fahrweg dar. Der Pfeil gibt die Fahrtrichtung an, das heißt alle Kanten sind gerichtet. Die FTF dürfen sich daher nur in einer Richtung bewegen. Das erste Netz (Abbildung 4.1) zeigt ein Beispiel mit drei mal drei Regalblöcken mit je zehn Regalen und zwei Entnahmestationen. Der Verkehrsfluss ist aufgrund der gerichteten Graphen (Einbahnstraßen) sehr eingeschränkt. Diese Einschränkung besteht im zweiten Netz (Abbildung 4.2) nicht. Dort gibt es zwischen jeder Kreuzung zwei gegenläufige Kanten. Ansonsten unterscheiden sich diese beiden Netze nicht voneinander. Einbahnstraßen haben den Vorteil den Verkehrsfluss zu steuern und den Verkehr an Kreuzungen zu verringern, da Fahrzeuge nicht in alle Richtungen fahren und Abbiegen dürfen. Beidseitig befahrbare Straßen sorgen für mehr Flexibilität bei der Routenplanung und ermöglichen das Umfahren von Engstellen. Dies führt an den Kreuzungen zu erhöhtem Verkehrsaufkommen und sorgt dafür, dass Kreuzungen länger blockiert sind.

Die Anzahl der Regale in den beiden Netzen kann auf zwei Arten verändert werden. Die Anzahl der Reihen und Spalten mit Regalblöcken kann verändert werden. Außerdem kann

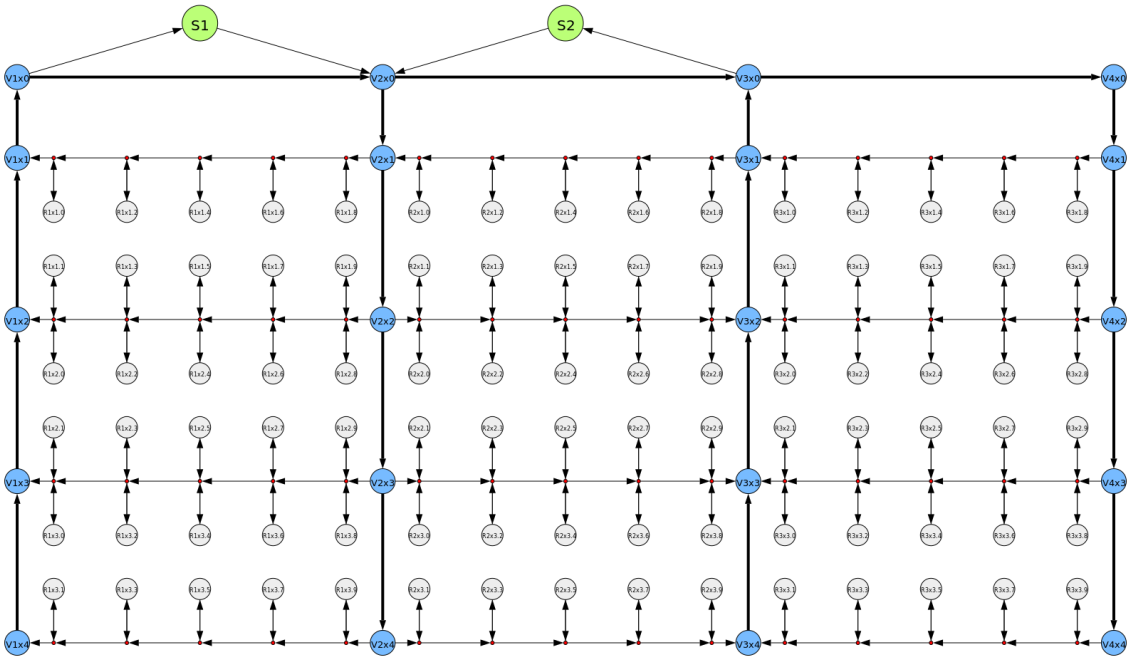

Abbildung 4.1: Straßennetz 1 - Einbahnstraßen

die Anzahl der Regale in einem Regalblock verändert werden. Die Anzahl der Stationen ist auch veränderbar, kann jedoch nicht höher als die Anzahl der Regalblock-Spalten sein.

Die Besonderheit am dritten Netz (Abbildung 4.3) ist der lange Korridor neben den Entnahmestationen *S5 - S7*. Die Abbildung zeigt 4 Regalblöcke mit je 10 Regalen und einer zugehörigen Station *S1 - S4*. Die Positionen der Produktionsstellen und Stationen sind durch die Fertigung bedingt. Der 'freie' Platz rechts neben den Stationen *S5-S7* ist durch weitere Lagerfläche belegt. Diese ist der Grund für den langen Korridor in dem Straßennetz. Auch dieses Straßennetz kann in der Größe verändert werden. Es können unten weitere Regalblöcke mit der dazugehörigen Station hinzugefügt werden. Weitere Spalten mit Regalblöcken sind jedoch nicht möglich. Durch die Anpassung der Größe der Regalblöcke kann außerdem die Anzahl der Regale verändert werden. Die Anzahl der Entnahmestationen kann außerdem variiert werden. Dadurch kann die Länge des Korridors verändert werden.

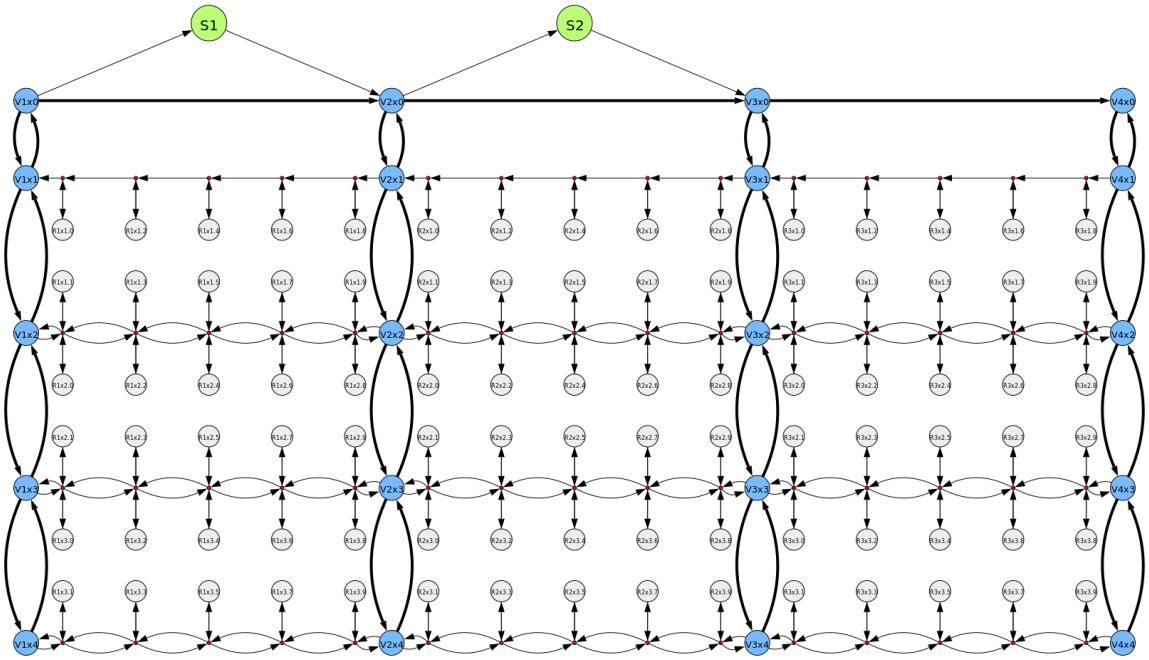

Abbildung 4.2: Straßennetz 2 - Beidseitig befahrbare Straßen

4.1 Szenarien

Die verwendeten Szenarien basieren auf drei zugrunde liegenden Straßennetzen. Die Szenarien variieren in der Anzahl der Quellen und Senken (Entnahmestationen), der Anzahl der zu transportierenden Elemente (Regale) und der Anzahl der FTF. Die Anzahl der Regale wird durch die Größe eines zusammenhängenden Regalblocks und der Anzahl dieser Blöcke bestimmt.

4.1.1 Reales Szenario 1

Ein reales Szenario aus der Automobilindustrie verwendet das Straßennetz aus Abbildung 4.1 mit 34 FTF, um 400 Regale zu vier Entnahmestationen zu transportieren. In jedem Regalblock sind 20 Regale aufgestellt. Täglich werden über 7000 Transporte durchgeführt. Ein Transport dauert durchschnittlich 5 Minuten. Daher wird für jedes FTF alle 5 Minuten ein Pfad für den nächsten Transportauftrag benötigt. Daraus ergeben sich zeitliche Schranken für den Algorithmus.

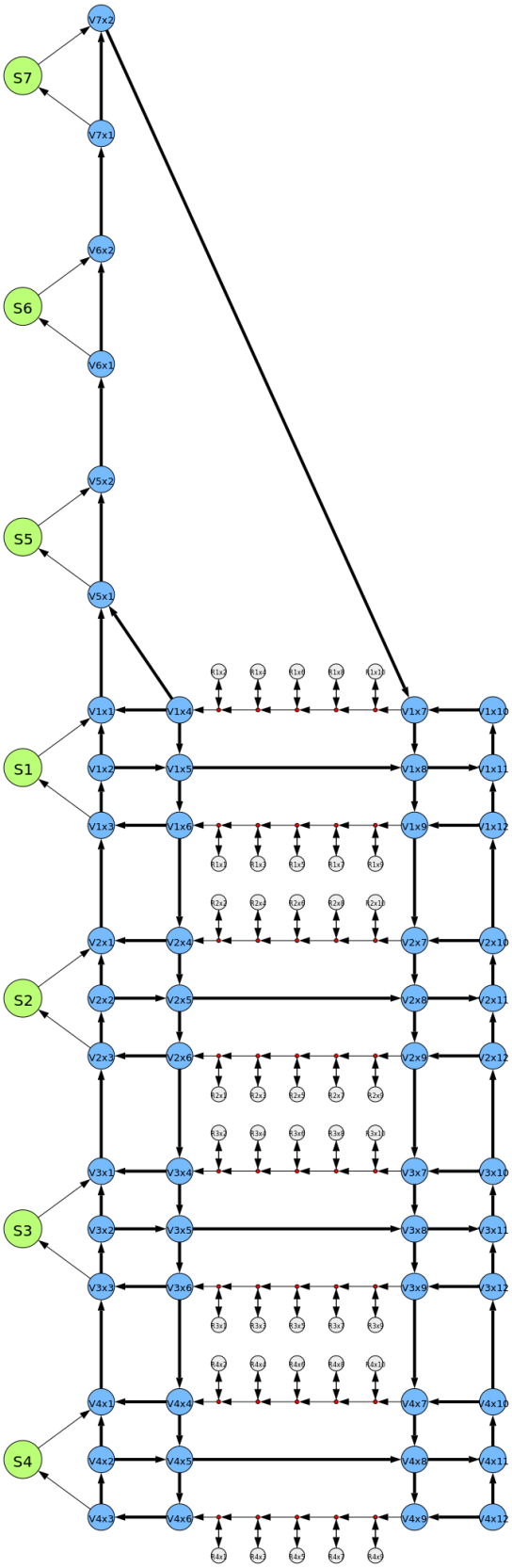

Abbildung 4.3: Straßennetz 3 - Langer Korridor

4.1.2 Reales Szenario 2

In diesem Szenario transportieren 16 FTF 144 Regale zu neun Produktionsstellen. Es sind 16 Regale in jedem Regalblock aufgestellt. Nachdem die Teile an den Produktionsstellen verarbeitet wurden, transportieren die FTF diese Teile zu zehn Entnahmestationen. Dieses Szenario aus der Automobilindustrie ist im Straßennetz aus Abbildung 4.3 vereinfacht dargestellt. Die Produktionsstellen entsprechen den Stationen $S1 - S4$ in der Abbildung. Die Stationen im langen Korridor (S5, S6 und S7) stellen die Entnahmestationen dar.

4.2 Bewertungskriterien und Benchmark

Folgende Kriterien werden für die Auswertung der Algorithmen verwendet:

Anzahl der Roboter Für wie viele Roboter können Pfade geplant werden? Im ersten realen Szenario werden 34 FTF eingesetzt. Daher sollte diese Anzahl mindestens erreicht werden.

Rechenzeit Die Berechnungszeit sollte minimal sein. Aus der Zahl und der Durchführungsdauer der Aufträge lässt sich eine obere Schranke von 1 Minute für die Rechenzeit der Routen ableiten.

Minimale Fahrzeit Die Fahrzeit aller Roboter soll minimal sein (späteste Ankunftszeit).

Deadlock Entstehen Deadlocks bei der Planung der Wege, da sich Roboter gegenseitig den Weg versperren?

Gesucht wird demnach ein Algorithmus, der in geringer Zeit für viele Roboter Pfade ohne Deadlock Situation berechnet und die Gesamtfahrzeit aller Roboter minimiert.

Jeder der drei ausgewählten Algorithmen wurde mit den gleichen Szenarien getestet. Dabei waren die Start- und Zielpositionen der einzelnen FTF identisch. Auf jedem Straßennetz wurden drei verschiedene Benchmarks durchgeführt. Für jeden Benchmark wurde ein Parameter kontinuierlich erhöht. Tabelle 4.1 gibt eine Übersicht über die verwendeten Parameter. Benchmark 1 erhöht die Anzahl der Roboter. Benchmark 2.1, 2.2 und 3 erhöhen die Anzahl der Regale. Der Unterschied liegt im Verhältnis von den Regalen zu den Straßen.

Benchmark 2 erhöht die Anzahl der Regalblöcke und Benchmark 3 erhöht die Anzahl der Regale in einem Block. Die zwei Versionen des zweiten Benchmarks ergeben sich aus den Besonderheiten des dritten Straßennetzes (Abbildung 4.3). In diesem Netz werden die Regalblöcke nur untereinander platziert. Für jeden Regalblock gibt es außerdem eine Produktionsstelle. Es gibt einen weiteren Parameter für dieses Straßennetz: Die Anzahl der Stationen. Benchmark 2.1 wird nur für die ersten beiden Straßennetze verwendet. Benchmark 2.2 nur für das Dritte. In Benchmark 2.2 werden nur 20 Roboter verwendet, da es auf einem realen Szenario basiert.

Tabelle 4.1: Parameter für unterschiedliche Szenarien

Name	#Entnahmestation	#Regale	#Regale per Block	#Roboter
Benchmark 1	4	400	20	**10-200**
Benchmark 2.1	4	**10-1000**	20	30
Benchmark 2.2	**2-50**	10-250[1]	10	20
Benchmark 3	4	**10-1000**	variabel (10-20)	30

4.3 Implementierung

Jeder Algorithmus wurde für den Vergleich in Python implementiert. Für die Algorithmen M* und FAR gibt es keinen öffentlich zugänglichen Quellcode. Diese Algorithmen wurden basierend auf der Beschreibung und Pseudocode aus den Veröffentlichungen implementiert [WB+08; WC11]. Für beide Algorithmen wird die Library *igraph* verwendet, um den Graph zu speichern. Für die Algorithmen wurde die gleiche A*-Implementierung als Grundlage verwendet. Die *List* wird in einem Heap, der nach der Distanz sortiert ist, gespeichert. Jeder Knoten wird, entsprechend seiner Distanz, in die *List* eingefügt. Durch diese Datenstruktur kann performant der Knoten mit der kleinsten Distanz aus dem Heap ausgewählt werden. Für jedes FTF wird ein Objekt angelegt. Dort wird eine eindeutige Nummer, der Start- und Zielknoten und Informationen zur berechneten Route gespeichert.

Die Implementierung von OMPP basiert auf dem Code der Autoren, wurde jedoch leicht angepasst. Damit im Test mehrere FTF das Ziel erreichen können, wurde die Beschränkung,

[1] Wird durch die Anzahl der Stationen bestimmt. Pro Produktionsstelle (*S1-S4*) gibt es einen Regalblock

welche die Anzahl der Fahrzeuge auf einem Knoten limitiert, für die Senken (Entnahmestationen) entfernt. OMPP arbeitet mit diskreten Zeitschritten und setzt daher Kantengewichte von 1 voraus. Für einen ersten Vergleich werden alle Kantengewichte auf 1 gesetzt. Um diesen Algorithmus in realen Szenarien nutzen zu können, müssten alle Kanten länger als 1 in mehrere Kanten aufgeteilt werden. In den beschriebenen Szenarien führt eine Normierung der Kantenlängen nur an einer Stelle zu neuen kürzesten Wegen. An den Entnahmestationen auf dem dritten Straßennetz (4.3) ist die Strecke von $V2x3$ nach $V2x1$ über $S2$ und $V2x2$ gleich lang. Für alle anderen Wege gibt es keine Alternative. Alle Wege unterscheiden sich in der Anzahl an Kanten.

4.4 Performanzanalyse

In diesem Abschnitt wird die Performanz der einzelnen Algorithmen gezeigt. Anschließend wird ein Vergleich der Algorithmen durchgeführt. In den folgenden Abbildungen sind jeweils die Ergebnisse der drei Benchmarks zu sehen. Der jeweils Linke der drei Graphen zeigt die Performanz des Algorithmus bei steigender Anzahl von FTF. Auf der X-Achse ist die Anzahl der FTF abgebildet, auf der Y-Achse die Berechnungszeit und die Weglänge. Die Zeit wird in Sekunden angegeben. Die beiden anderen Graphen zeigen die Performanz bei steigender Regalanzahl. Die X-Achse gibt die Anzahl der Regale an. Auf Straßennetz 3 wird außerdem die Anzahl der Stationen erhöht, wie in Abschnitt 4.2 beschrieben wurde.

4.4.1 OMPP

Alle Berechnungen für OMPP wurden mit einer Kantenlänge von 1 durchgeführt. Auf dem Straßennetz 1 werden für 40 Fahrzeuge 80 Sekunden benötigt. Auch die Betrachtung von 30 FTF und 25 Regalblöcken mit je 20 Regalen dauert die Berechnung mit 59 Sekunden bereits zu lange. Auf den anderen Straßennetzen ist die Performanz vergleichbar. Auf Straßennetz 2 wurde mit 20 FTF und 400 Regalen sowie mit 30 FTF bei 300 Regalen das Limit von 60 Sekunden erreicht. Auf dem dritten Straßennetz wurden in Benchmark 1 Wege für 50 FTF in 60 Sekunden berechnet. In den weiteren Benchmarks wurde mit 30 FTF und 200 Regalen bereits die Minute Rechenzeit erreicht.

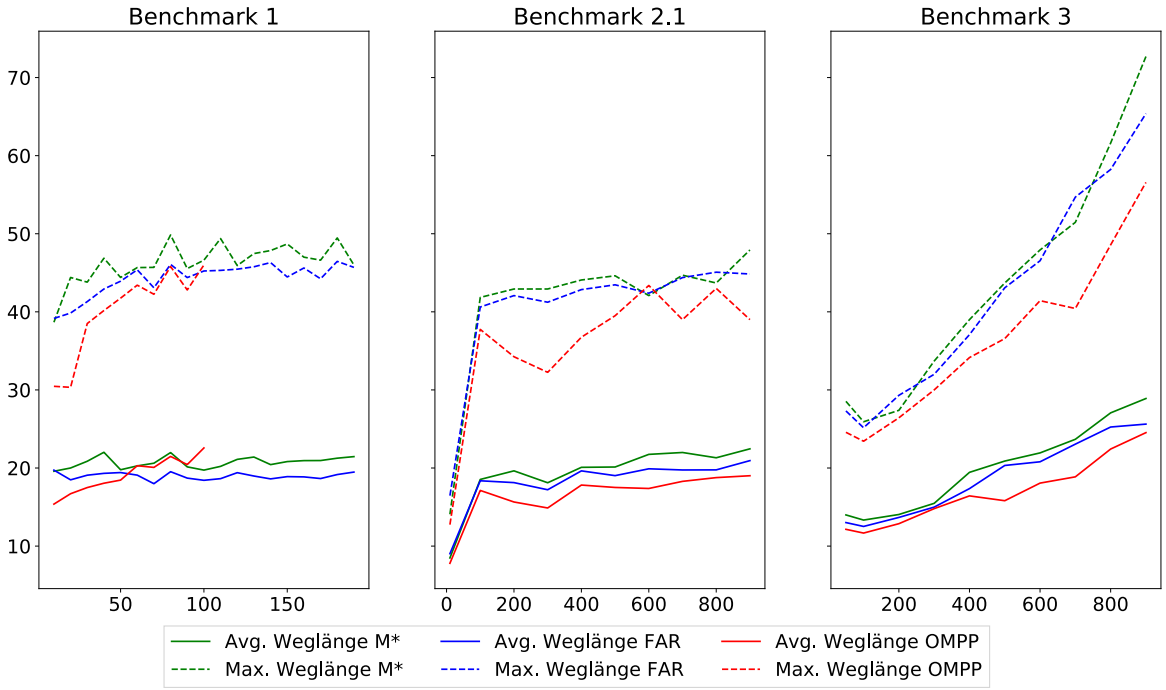

Abbildung 4.4: OMPP, M* und FAR für Straßennetz 1 mit dem Kantengewicht von 1

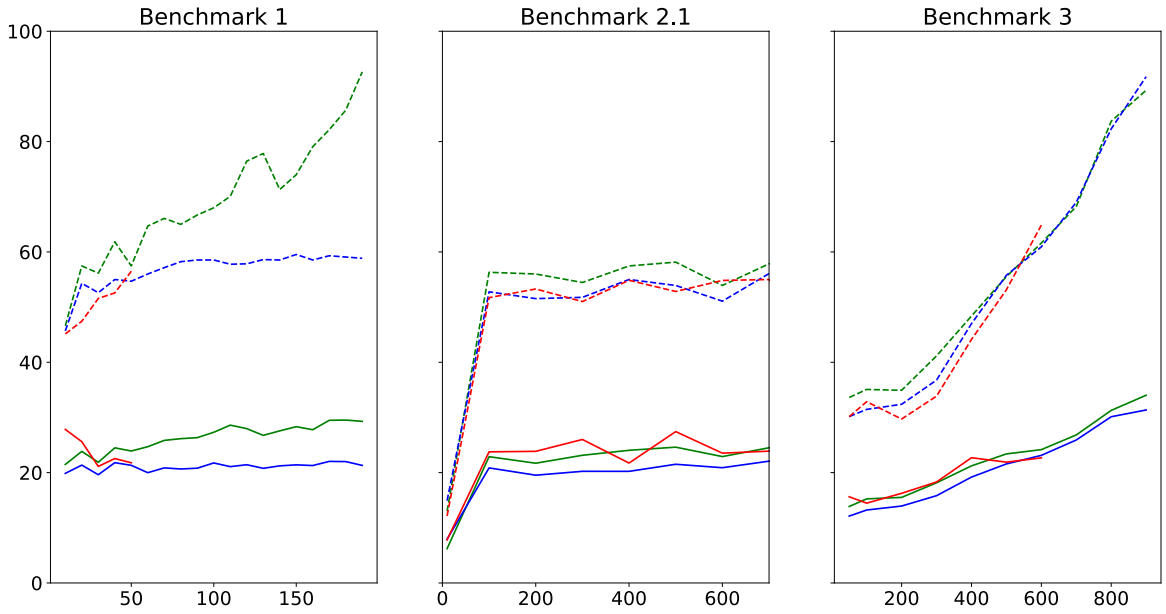

Abbildung 4.5: OMPP, M* und FAR für Straßennetz 2 mit dem Kantengewicht von 1

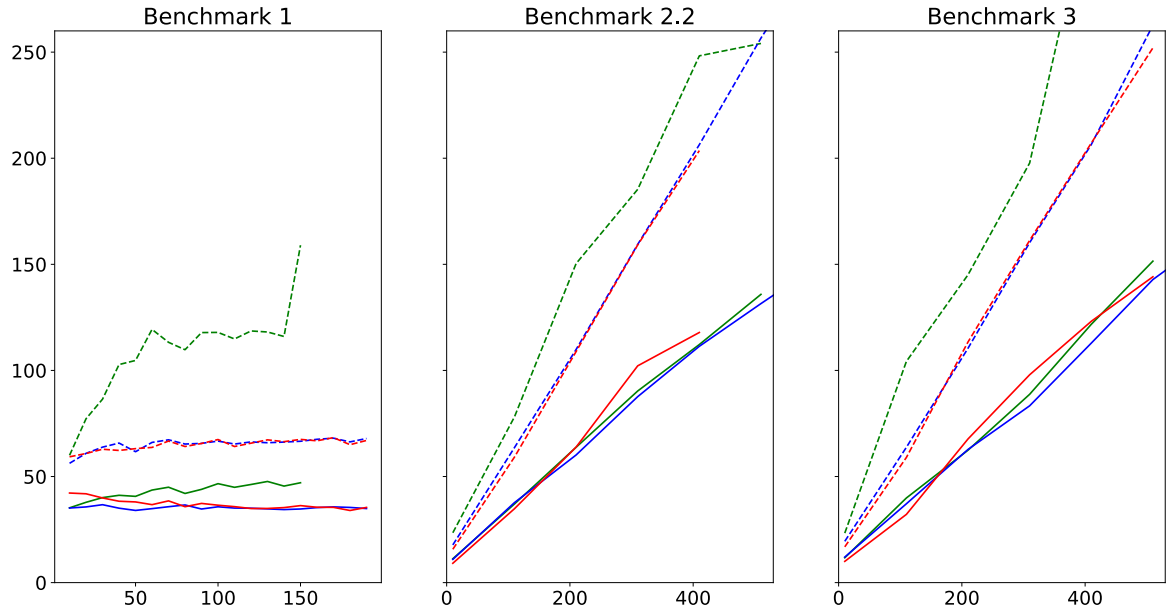

Abbildung 4.6: OMPP, M* und FAR für Straßennetz 3 mit dem Kantengewicht von 1

Der Algorithmus benötigt die meiste Rechenzeit, insbesondere, wenn die Probleme größer werden. Er liefert allerdings sehr gute Ergebnisse. Abbildung 4.4 zeigt, dass die durchschnittliche und die maximale Weglänge auf dem ersten Straßennetz in den meisten Fällen geringer als bei den anderen Algorithmen ist. Für diesen Vergleich wurden alle Algorithmen auf Straßennetzen mit einer Kantenlänge von 1 getestet.

In Abbildung 4.5 ist der Vergleich der Algorithmen auf dem zweiten Straßennetz zu sehen. Die Legende zu diesem Graphen befindet sich unter der ersten Abbildung. Die durchschnittliche Weglänge, die durch FAR berechnet wurde, ist teilweise kürzer als die von OMPP. Da OMPP auf die späteste Ankunftszeit optimiert ist, ist eine höhere durchschnittliche Weglänge nicht verwunderlich. In dem Graphen ist auch zu sehen, dass FAR bei der maximalen Weglänge in manchen Situationen kürzere Wege berechnet hat. Die Wegstrecken unterscheiden sich in Abbildung 4.6 kaum. Das liegt am Aufbau des Straßennetzes.

Die Abbildungen 7.1 bis 7.3 im Anhang zeigen eine detaillierte Auswertung der Performanz von OMPP auf den drei verschiedenen Straßennetzen.

4.4.2 M*

Die Berechnungen wurden auf einem Grid durchgeführt, bei dem die Kantenlänge dem Kantengewicht entspricht. Daher sind die Wege in dieser und den folgenden Graphen länger als bei OMPP. Die Berechnungen für M* wurden mit einem Zeitlimit von 200 Sekunden durchgeführt. Das heißt, dass sobald eine Berechnung dieses Limit überschreitet, das Benchmark beendet ist. Im Gegensatz zu OMPP garantiert M* nicht immer einen Weg zu finden. Daher ist in den Graphen zusätzlich der Erfolg dargestellt. Die Skala von 0 bis 1 ist auf der rechten Seite der Graphen zu sehen.

Interessant an der Erfolgsquote ist, dass sie im dritten Straßennetz bei Benchmark 2 und 3 steigt. Das kann durch die steigende Anzahl an Stationen erklärt werden, da im dritten Straßennetz mit steigender Zahl an Regalen auch die Anzahl der Stationen steigt (siehe Abschnitt 4.1). Auf allen Straßennetzen sinkt die Erfolgsquote im ersten Benchmark, da dort auf dem gleichen Straßennetz die Roboteranzahl erhöht wird. Der rapide Anstieg der Weglänge am Anfang von Benchmark 2 liegt daran, dass der erste Test mit nur 10 Regalen durchgeführt wurde und der Weg von den Regalen zu den Stationen daher sehr kurz ist. Die Weglänge steigt bei Benchmark 2 bis 200 Regale und fällt dann wieder. Diese Spitze ist dem Aufbau des Netzes geschuldet. Das Grid (siehe Abbildungen 4.1 und 4.2) wird blockweise mit Regalen aufgebaut. Zuerst wird die erste Zeile von links nach rechts aufgebaut. Bei 200 Regalen ist die zweite Zeile der Regale gefüllt, daher wird die Weglänge dort wieder kürzer.

Die Berechnungszeit von M* auf Straßennetz 3 wächst mit steigender FTF oder Regalanzahl besonders stark.

In den Abbildungen 7.4 bis 7.6 im Anhang ist die Performanz vom Algorithmus M* zu sehen.

4.4.3 FAR

Auch bei diesem Algorithmus steigt die Rechenzeit mit der Anzahl der Roboter oder den Regalen. Auf dem ersten Straßennetz benötigt FAR für Benchmark 1 mehr Rechenzeit als M*. FAR benötigt für 125 Roboter bereits mehr als 200 Sekunden. Der Benchmark wurde daher frühzeitig beendet, daher sind für mehr FTF keine Daten vorhanden. Die Ergebnisse von

Benchmark 2.1 und Benchmark 3 sind vergleichbar mit denen von M*. Auf dem Straßennetz 2 und 3 ist FAR schneller und liefert kürzere Strecken.

Die Berechnungszeit wächst für Benchmark 1, Benchmark 2.1 und Benchmark 2.2 fast linear. In Benchmark 3 wächst sie schneller. Je nach Szenario berechnet FAR Wege für 30 bis 120 FTF in einer Minute. Die Ergebnisse von Benchmark 2.1, 2.2 und 3 zeigen, dass die Berechnungszeit bei 400 Regalen unter einer Minute liegt.

Die Performanz von FAR ist in den Abbildungen 7.7 bis 7.9 im Anhang zu sehen. FAR findet immer einen Weg zum Ziel, daher ist die Erfolgsquote immer 1 und daher nicht in den Graphen dargestellt.

4.4.4 Auswertung und Zusammenfassung

Die einzelnen Graphen der Algorithmen zeigen die Entwicklung des jeweiligen Algorithmus. Abbildungen 4.7 bis 4.9 zeigen den direkten Vergleich von M* und FAR auf den drei Straßennetzen. Die Legende zu den Graphen befindet sich unter der ersten Abbildung. Auf der X-Achse ist die Anzahl der FTF abgebildet, auf der Y-Achse die Berechnungszeit und die Weglänge. Ein Vergleich mit OMPP auf Graphen mit Kantengewichtung ungleich 1 wurde nicht durchgeführt. Da OMPP im ersten Test der langsamste Algorithmus war und die geforderten Kriterien (Abschnitt 4.2) nicht erfüllt, wird dieser Algorithmus in den folgenden Vergleichen nicht betrachtet.

FAR liefert in allen getesteten Situationen bessere oder gleiche Weglängen und benötigt in den meisten Fällen dafür auch weniger Rechenzeit. Außerdem ist die Erfolgsquote bei FAR immer 100 %. M* fand in Tests mit vielen Robotern nur noch für 30 % der Roboter einen kollisionsfreien Pfad. Der Unterschied der beiden Algorithmen sieht in den Abbildungen sehr gering aus. Dies liegt insbesondere daran, dass Umwege und Warten relativ zur Gesamtstrecke sehr gering ausfallen. In den Abbildungen 4.4 und 4.5 ist der Unterschied deutlicher zu sehen, da dort die Gesamtstrecke geringer ist.

Der Algorithmus FAR schneidet in den Tests am besten ab. Er plant jedoch keine Umwege um Hindernisse, sondern wartet nur bis der Weg wieder frei ist. Eine Kombination von M* und FAR würde dieses Problem lösen. Diese Idee wird im folgenden Kapitel genauer beschrieben.

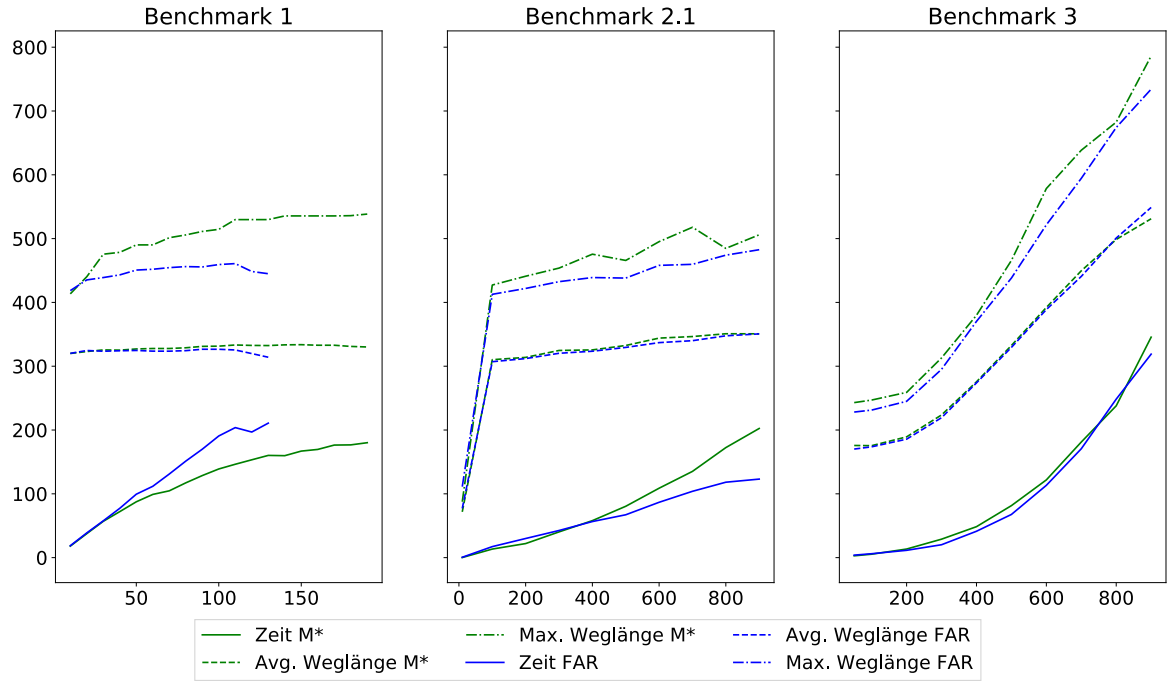

Abbildung 4.7: M* und FAR auf Straßennetz 1

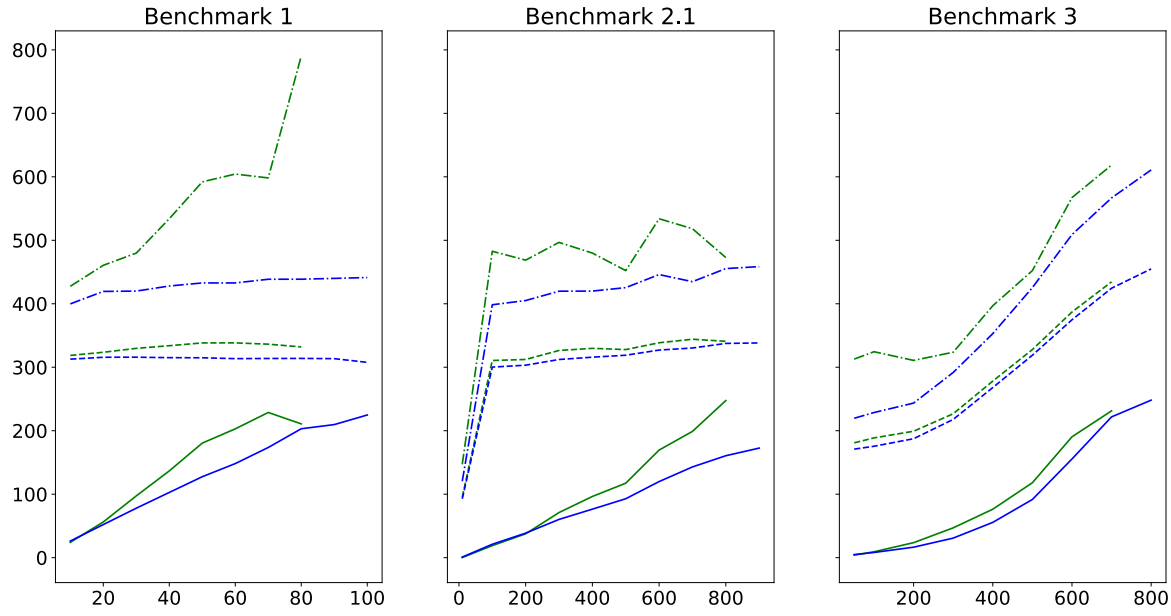

Abbildung 4.8: M* und FAR auf Straßennetz 2

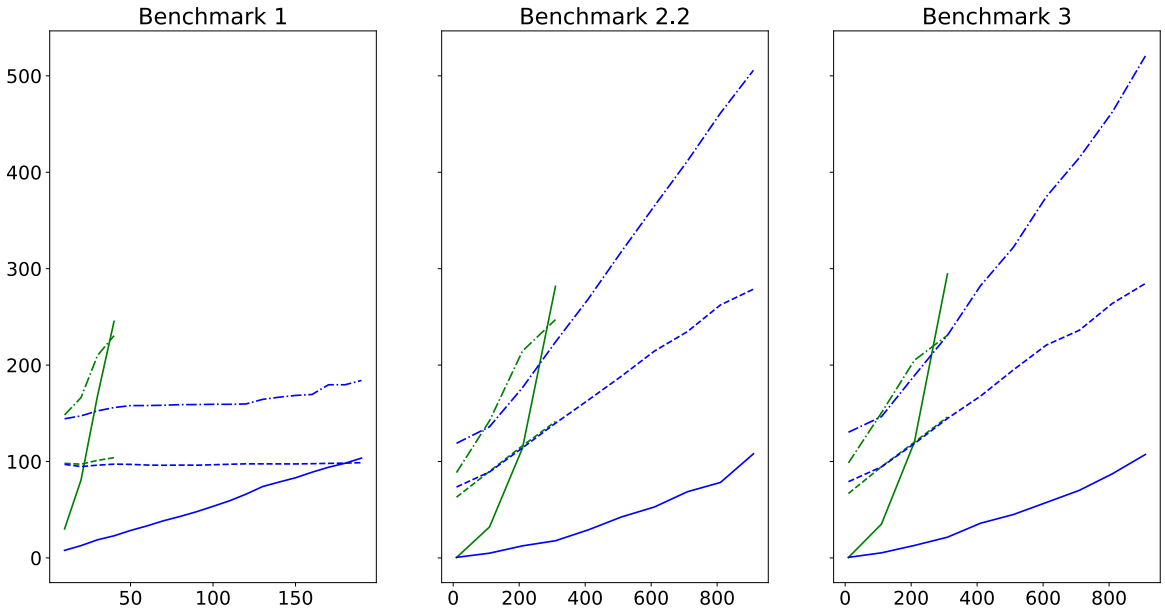

Abbildung 4.9: M* und FAR auf Straßennetz 3

5 Implementierung und Validierung eines Pfadplanungs-Algorithmus

In diesem Kapitel wird beschrieben, wie die Vorteile von FAR und M* miteinander kombiniert werden können. Anschließend wird die Performanz des neuen Algorithmus validiert.

5.1 Implementierung

FAR orientiert sich an dem kürzesten Weg und versucht durch Warten und Auflösen von Deadlocks für alle FTFs einen Weg zu finden. M* plant bei der Wegsuche bereits Umwege um andere FTF herum ein. Dabei wird allerdings nicht darauf gewartet, dass ein Weg frei wird.

Der neue Algorithmus wird W* genannt, da er Warten und M* kombiniert. Er soll durch taktisches Warten, wenn eine Umfahrung der Engstelle teurer ist, kürzere Wege finden als M*. Durch das Umfahren von "Verstopfungen" sollen Deadlocks bereits bei der Planung vermieden werden. Als Grundlage für W* wird der Dijkstra-Algorithmus verwendet. A* wird nicht als Grundlage verwendet, da aufgrund der Heuristik kürzere Wegstrecken bevorzugt werden, auch wenn ein Umweg schneller zum Ziel führt. W* wird auch in Python implementiert. Daher können die gleichen Benchmarks verwendet werden und die Performanz mit FAR und M* verglichen werden.

Die Kosten einer Kante werden aus der Fahrzeit des jeweiligen FTFs berechnet. So kann der Algorithmus Wege für FTF mit unterschiedlichen Geschwindigkeiten berechnen. Beim Expandieren eines Knotens wird geprüft, ob der Knoten bereits belegt ist. Wenn er belegt

ist, wird geprüft, wie lange der Knoten belegt sein wird. Da die FTF dicht hintereinander fahren können, kann ein Knoten für mehrere FTF in Folge reserviert sein.

Die Wartezeit wird auf die Kosten für diesen Knoten addiert. Die Kosten der einzelnen Knoten enthalten die Summe der Wegkosten und der Kosten für das Warten. Wenn der Zielknoten erreicht wurde, wird der kürzeste Weg, genau wie bei Dijkstra, durch die gespeicherten Vorgänger rekonstruiert.

Der triviale Ansatz zur Ermittlung, ob ein Knoten durch ein anderes Fahrzeug belegt ist, ist das Vergleichen der Position aller Fahrzeuge. In ersten Tests führt dieser Ansatz bereits bei wenigen Fahrzeugen zu hohen Laufzeiten. Für den Algorithmus ist nur relevant, ob zu einem zukünftigen Zeitpunkt X ein Knoten blockiert ist. Daher wird in jedem Knoten gespeichert, zu welchem Zeitpunkt dieser belegt ist. Dafür müssen die Zeiten diskretisiert werden. Ein FTF befindet sich allerdings nicht genau zu einem Zeitpunkt auf einem Knoten. Der Knoten entspricht in der Fabrik einer Kreuzung mit einer gewissen Grundfläche. Daher wird zusätzlich die Zeit gespeichert, die dieses FTF zum Überqueren der Kreuzung benötigt. Mit diesen beiden Daten kann geprüft werden, ob ein Knoten zu einem gewissen Zeitpunkt belegt ist.

Der Pseudocode des neuen Algorithmus ist in Algorithmus 5.1 zu sehen. Er bekommt den Graphen G, den Startknoten v_0, den Zielknoten v_e und eine Liste aller Fahrzeuge $FTFs$ als Argumente übergeben. Bei der $openlist$ handelt es sich um einen Heap, der die Einträge nach den Kosten sortiert. In Zeile 10 wird daher immer der Knoten mit den geringsten Kosten aus der $openlist$ heraus genommen. In Zeile 17 wird, falls v_l zu diesem Zeitpunkt bereits belegt ist, ausgerechnet, wie lange gewartet werden muss, bis v_l wieder frei ist. Diese Kosten werden auf die Wegkosten addiert. In Zeile 20 wird geprüft, ob der kürzeste Weg zu v_l über v_k geht. Falls dies der Fall ist, wird der Knoten v_l aktualisiert.

Dieser Algorithmus wurde in einer Simulationsumgebung integriert. Diese Umgebung simuliert den Ablauf eines Logistik-Warenhauses. So konnte der Algorithmus in realen Szenarien getestet werden.

Algorithmus 5.1 Pseudocode vom neuen Algorithmus W*

```
1:  procedure W*(G,v_0,v_e,FTFs)
2:      for all v ∈ G do
3:          v.cost ← ∞
4:      end for
5:      for all f ∈ FTFs do
6:          MARKIERE_BELEGTE_KNOTEN(f)
7:      end for
8:      openlist = {v_0}
9:      while |openlist| > 0 do
10:         v_k = OPENLIST.POP()
11:         if v_k = v_e then
12:             return CREATE_PATH(G, v_e)
13:         end if
14:         for all v_l ∈v_k.NEIGHBORS( ) do
15:             cost_new ← v_k.cost+BERECHNE_WEGKOSTEN(v_k,v_l)
16:             if CHECKE_KNOTEN_BELEGUNG(v_l,cost_new) then
17:                 cost_new ← cost_new+BERECHNE_WARTEZEIT(v_l)
18:             end if
19:             OPENLIST.ADD(v_l)
20:             if cost_new < v_l.cost then
21:                 AKTUALISIERE_KNOTEN(v_l)
22:             end if
23:         end for
24:     end while
25: end procedure
```

5.2 Validierung

Die Performanz des neuen Algorithmus W* wird mit den gleichen Benchmarks, die in Kapitel 4 beschrieben wurden, gemessen. Für die Berechnungen wurde wieder ein Zeitlimit von 200 Sekunden gesetzt. Daher sind teilweise für Tests mit vielen FTF oder Regalen keine Daten vorhanden.

In 60 Sekunden wurden Wege für 55 bis 80 FTF auf dem ersten Straßennetz berechnet. In Benchmark 2.1 und Benchmark 3 wurden Wege für 700 Regale in einer Minute berechnet. Die Ergebnisse der Benchmarks auf dem Straßennetz 2 unterscheiden sich kaum vom Straßennetz 1. Auf Straßennetz 2 wird etwas mehr Rechenzeit benötigt. Auch auf dem Stra-

ßennetz 3 verhält sich die Berechnungszeit ähnlich zu den vorausgegangenen Messungen.

Die Abbildungen 5.1 bis 5.3 zeigen den direkten Vergleich von M*, FAR und W*. Die Legende zu den Graphen befindet sich unter der ersten Abbildung.

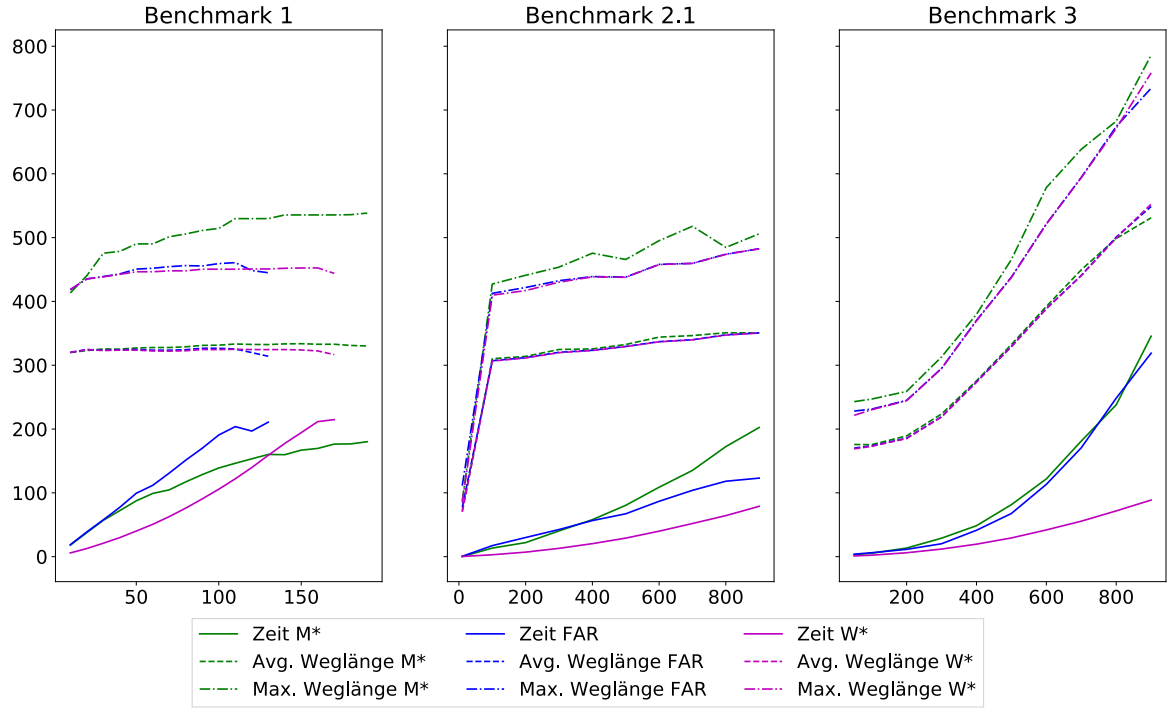

Abbildung 5.1: W*, M* und FAR auf Straßennetz 1

Die berechneten Weglängen sind in allen Benchmarks auf allen drei Straßennetzen kürzer oder vergleichbar zu den anderen beiden Algorithmen FAR und M*. Der Unterschied der Weglängen ist sehr gering und ist daher nicht in den Graphen zu erkennen. In Abbildung 5.4 ist ein Ausschnitt aus Abbildung 5.2 zu sehen. Die Abbildung zeigt, dass W* etwas kürzere Wegstrecken berechnet.

Die Berechnungszeit von W* ist in Benchmark 1 auf dem ersten Straßennetz bis 125 FTF am geringsten. Ab 125 FTF ist M* schneller. In allen weiteren Benchmarks auf Straßennetz 1 und Straßennetz 2 ist W* am schnellsten. W* benötigt teilweise weniger als die Hälfte der Berechnungszeit oder berechnet in der gleichen Zeit Wege für 30 % mehr Fahrzeuge. Die Berechnungszeit auf Straßennetz 3 steigt annähernd exponentiell an und ist nur bis

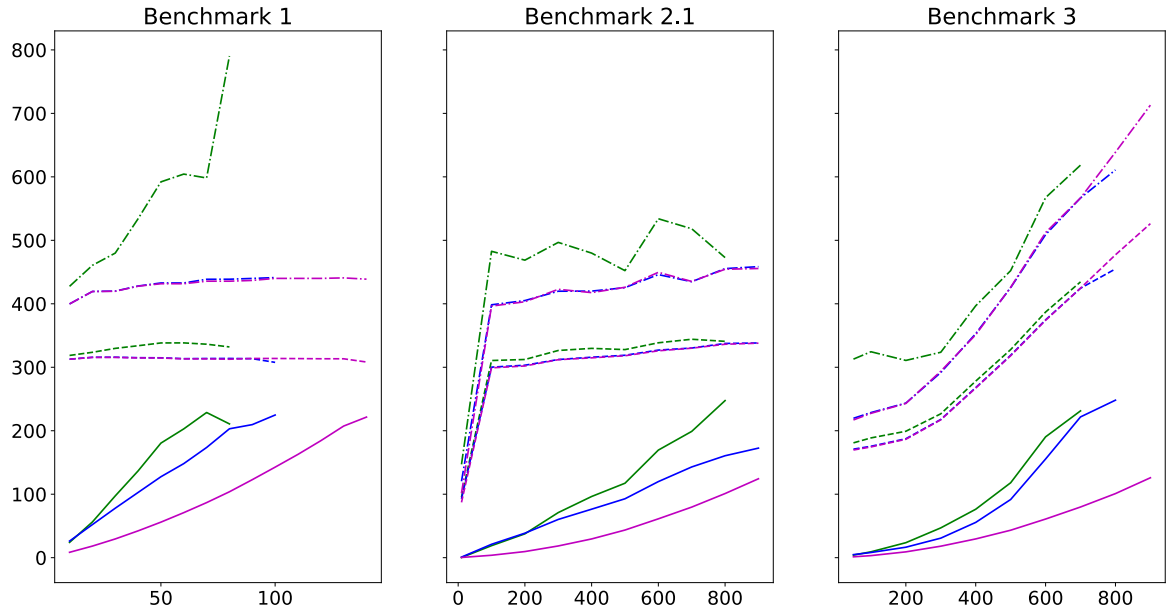

Abbildung 5.2: W*, M* und FAR auf Straßennetz 2

40 FTF schneller als die anderen beiden. Ab 40 FTF ist FAR schneller. In Benchmark 2.2 und 3 benötigen W* und FAR für Berechnungen bis 500 Regalen vergleichbar viel Zeit. Anschließend ist FAR etwas schneller. W* benötigt also in den meisten Benchmarks weniger Berechnungszeit und berechnet kürzere Wegstrecken. Die Abbildungen 7.10 bis 7.12 im Anhang zeigen eine detaillierte Auswertung der Performanz von W* auf den drei verschiedenen Straßennetzen.

Der neue Algorithmus wurde auch auf Praxistauglichkeit getestet. Dazu wurde er in eine Simulationsumgebung der Firma Arculus integriert und der Durchsatz einer Fabrik gemessen. Der Durchsatz von zwei Simulationen unterschiedlicher Warenhäusern wurde durch W* um wenige Prozent gesteigert. Auch wenige Prozent mehr Durchsatz bedeuten bei einem Warenhaus mit mehr als 7000 täglichen Transporten finanzielle Einsparungen. Konkrete Zahlen und eine genauere Analyse der Verbesserung ist nicht möglich, da derzeit nicht ausreichen Testergebnisse vorliegen. Außerdem wurde die Anzahl der Staus verringert. Aufgrund der hohen Verkehrsdichte in manchen Teilen der Fabrik kommt es dort, trotz der Verbesserung, immer noch zu Staus.

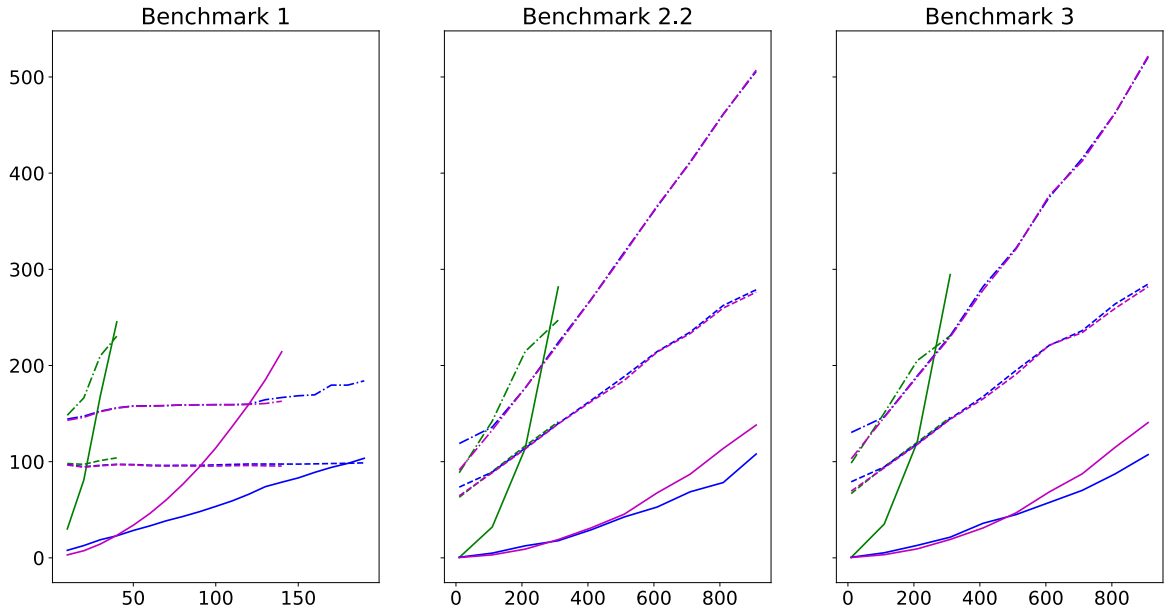

Abbildung 5.3: W*, M* und FAR auf Straßennetz 3

Insgesamt liefert der neue Algorithmus gute Ergebnisse und kombiniert die Vorteile von FAR und M* erfolgreich. Er kann jedoch noch verbessert werden. Im nächsten Kapitel werden daher weitere Verbesserungsvorschläge erläutert.

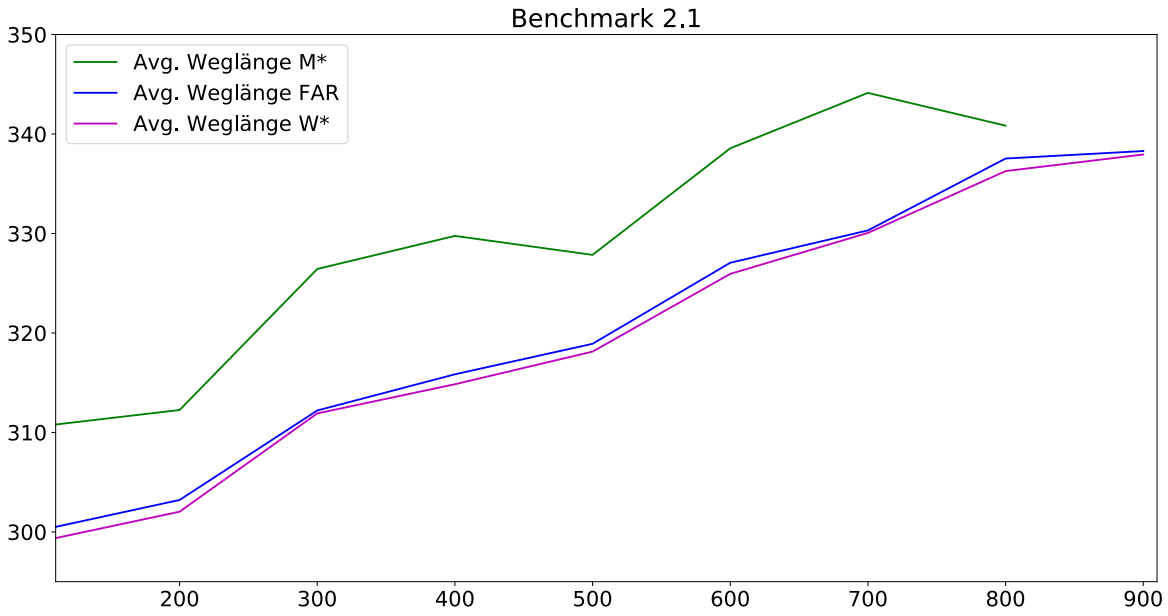

Abbildung 5.4: Ausschnitt der Abbildung 5.2 von Benchmark 2.1

6 Zusammenfassung und Ausblick

In modernen Logistik-Warenhäusern werden FTF eingesetzt, um Waren zu transportieren. In dieser Arbeit wurden verschiedene Algorithmen zur Planung der Wege für diese FTF vorgestellt. Anschließend wurden drei Algorithmen nach definierten Kriterien ausgewählt.

Diese drei Algorithmen (OMPP, FAR und M*) wurden in Python implementiert, um die Performanz in unterschiedlichen Benchmarks zu testen.
OMPP erstellt ein Gleichungssystem, um den optimalen Weg für alle FTF zu berechnen. Ein Solver berechnet für dieses Gleichungssystem eine Lösung mit minimaler Fahrzeit für alle Roboter. FAR orientiert sich an dem kürzesten Weg und versucht durch Warten und Auflösen von Deadlocks für alle FTFs einen Weg zu finden. M* plant bei der Wegsuche bereits Umwege um andere FTF herum ein. Dabei wird allerdings nicht darauf gewartet, dass ein Weg frei wird.

Die Performanz der Algorithmen wurde in insgesamt neun verschiedenen Szenarien getestet und ausgewertet. Die berechneten Wegstrecken von OMPP sind dabei am besten. Die Berechnungszeit von OMPP ist jedoch viel zu lang. Daher kann dieser Algorithmus nicht in der Praxis eingesetzt werden. Die Berechnungszeit von FAR und M* sind geringer und überschreiten das definierte Zeitlimit von einer Minute nur bei Tests mit sehr vielen FTF oder sehr vielen Regalen. Weder FAR noch M* ist in allen Tests besser als der jeweils andere Algorithmus, weder bei der Berechnungszeit noch bei den berechneten Wegen.

Basierend auf dieser Analyse der Algorithmen wurde ein neuer Algorithmus W* entwickelt. Er basiert auf Dijkstra und verbindet die Vorteile von FAR und M*. Die grundlegende Idee ist, bei der Berechnung der Kosten für einen Knoten, die Kosten für die Wartezeit auf die Kosten der Wegstrecke zu addieren. So wird immer dann gewartet, bis ein Knoten frei wird, wenn kein kürzerer Weg um die Engstelle existiert. Dieser Algorithmus wurde auch in Python implementiert und mit den gleichen Benchmarks getestet. In den meisten Test ist W* schneller

als FAR oder M*. Außerdem sind die berechneten Wege besser oder zumindest vergleichbar. Die Vorteile beider Algorithmen wurden also erfolgreich miteinander kombiniert.

Der neue Algorithmus wurde in eine Simulations-Software integriert. In ersten Tests wurde mit W* ein erhöhter Durchsatz in den Simulationen erreicht.

Ausblick

Im Rahmen dieser Arbeit konnten einige Aspekte nicht untersucht werden. In diesem Kapitel wird daher beschrieben, wie der Algorithmus weiterentwickelt werden kann.

In Warenhäusern können Transportaufträge unterschiedliche Prioritäten haben. Diese Prioritäten bestimmen primär, in welcher Reihenfolge die Transportaufträge durchgeführt werden sollen. Bei der Routenplanung werden diese Prioritäten derzeit nicht berücksichtigt. Um einen Transportauftrag zu bevorzugen, darf das FTF nicht auf andere FTF warten. Die Reihenfolge, in der die FTFs aufeinander warten, wird derzeit durch die Reihenfolge der Routenplanung bestimmt. In einem Lagerhaus werden die Aufträge nacheinander bearbeitet. Das heißt, es wird der Reihe nach für jeden Auftrag eine Route berechnet und das FTF startet anschließend direkt die Fahrt. Um die Reihenfolge der Routenberechnung zu ändern, muss demnach einem bereits fahrenden FTF eine neue Route zugewiesen werden. Das FTF darf während der Berechnung nicht stehen bleiben, da es andere Fahrzeuge dadurch blockieren würde. Daher kann die aktuelle Position des Fahrzeuges nicht als Startpunkt verwendet werden. Für die neue Route muss ein passender Startpunkt, der sich auf der aktuellen Route befindet, gewählt werden. Dieser Startpunkt muss so gewählt werden, dass das FTF an diesem Punkt nicht während der Berechnung vorbei fährt. Unter Berücksichtigung dieser Aspekte kann einem FTF eine höhere Priorität zugewiesen werden.

Unabhängig von Prioritäten kann untersucht werden, wie stark sich die Reihenfolge der Routenplanung auf das Gesamtergebnis auswirkt. OMPP berechnet die Routen aller FTF immer neu und liefert daher ein optimales Ergebnis. Da W* die Wege der Reihe nach berechnet, kann dies Auswirkungen auf das Ergebnis haben. Wenn beispielsweise ein FTF einmal quer durch das Lager fährt und daher alle anderen FTF warten müssen, wäre es besser das quer fahrende FTF wartet einmal und passiert die jeweiligen Kreuzungen nach den anderen FTF.

In Warenhäusern kann es Straßenabschnitte mit geringer Deckenhöhe geben oder die maximale Durchfahrtshöhe aus anderen Gründen zu gering für ein beladenes FTF sein. Diese Straßenabschnitte können daher im Moment nicht für das Routing verwendet werden, da nicht zwischen Straßen für beladene und unbeladene FTFs unterschieden werden kann. Die Durchfahrtshöhe der Straße kann als Attribut für eine Kante hinzugefügt und bei der Routenberechnung berücksichtigt werden. Mit dieser Erweiterung ist es auch möglich Fahrwege durch niedrige Straßenabschnitte zu planen. Unbeladene FTF könnten z. B. unter den Regalen in den Regalblöcken fahren oder diesen Platz zum Auflösen von Konflikten verwenden. Weitere Analysen können zeigen, wie sich diese Erweiterung auf die Performanz des Warenhauses auswirkt.

In der Literatur werden Kriterien, nach denen Roboter mit unterschiedlichen Geschwindigkeiten an Engpässen aufeinander warten, untersucht [Kal12]. Der aktuelle Algorithmus bezieht die Geschwindigkeiten, bei der Entscheidung, welcher Roboter warten muss, nicht mit ein. Eine entsprechende Implementierung würde es erlauben, dass schnelle FTF andere überholen können. In weiteren Benchmarks kann untersucht werden, wie sich dies auf die Weglängen auswirkt.

Der Algorithmus W* kann aktuell keine Deadlocks erkennen. Aufgrund der beschriebenen Performanz-Optimierung ist nicht bekannt, welches FTF auf welches wartet. Es ist nur bekannt, wie lange ein Knoten belegt ist. Zusätzlich zu der Information, dass ein Knoten zu einem bestimmten Zeitpunkt belegt ist, muss gespeichert werden, welches FTF diesen Knoten belegt. Mit diesen Informationen können Deadlocks erkannt werden. Es kann anschließend untersucht werden, welche Methode zur Auflösung von Deadlocks am effizientesten ist.

Die bisher beschriebenen Anpassungen versuchen das Ergebnis zu verbessern. Die folgenden Abschnitte beschreiben, wie die Berechnungszeit verringert werden kann.

Der Dijkstra-Algorithmus besteht aus mehreren Schritten. Der Schritt, der alle Nachbarknoten betrachtet und deren Distanzwert berechnet, benötigt am meisten Rechenzeit. Dieser Distanzwert kann normalerweise nicht im Voraus berechnet werden, da die Anzahl der Start- bzw. Zielknoten zu groß ist. Alle Fahrten der FTF in einem Warenhaus enden oder starten an einer der Entnahmestationen. Da die Anzahl dieser Stationen gering ist, ist es möglich die Distanzen zu jedem Knoten im Voraus zu berechnen. Dazu wird der Dijkstra-Algorithmus verwendet. Als Startknoten wird die jeweilige Station gewählt. Der Algorithmus beendet sich erst, wenn alle Knoten besucht wurden. Es wurde dann für jeden Knoten die Distanz

zum Startknoten und der Vorgängerknoten berechnet. Diese Daten müssen gespeichert werden und können bei zukünftigen Routingberechnungen verwendet werden. Auch die Wege zu den Entnahmestationen können vorberechnet werden. Dazu muss Dijkstra bei der Suche nach Nachbarknoten nur eingehenden Kanten betrachten. Also die Kanten gegen ihre Fahrtrichtung verwenden.

Die Algorithmen wurden alle in Python implementiert. Python ist eine interpretierte Sprache und daher nicht so Performant wie C oder C++. Um die Performanz von W* weiter zu steigern, kann der Algorithmus auch in C oder C++ umgesetzt werden. Die verwendete Graph Library ist in C geschrieben und implementiert Dijkstra zur Berechnung des kürzesten Weges zwischen zwei Punkten. Diese Funktion berechnet alle Wege der in dieser Arbeit vorgestellten Benchmarks in unter einer Sekunde. Dies zeigt einen möglichen Performanz-Gewinn einer Implementierung von W* in C. Um den Performanzunterschied testen zu können, kann W* in der Graph Library implementiert werden und anschließend mit den Benchmarks dieser Arbeit getestet werden.

Literaturverzeichnis

[Bau17] T. Bauernhansl. „Die vierte industrielle Revolution–der Weg in ein wertschaffendes Produktionsparadigma“. In: *Handbuch Industrie 4.0 Bd. 4*. Springer, 2017, S. 1–31 (zitiert auf S. 11).

[Die12] R. Diestel. *Graph Theory: Springer Graduate Text GTM 173*. Springer Graduate Texts in Mathematics (GTM). Math. Forschungsinst., 2012 (zitiert auf S. 18).

[Ein17] B. Einstein. *Meet the drone that already delivers your packages, Kiva robot teardown | Robohub*. 2017. URL: http://robohub.org/meet-the-drone-that-already-delivers-your-packages-kiva-robot-teardown/ (zitiert auf S. 17).

[Ger15] Gerry. *How robots will eat our jobs - That's How The Light Gets In*. 2015. URL: https://gerryco23.wordpress.com/2015/03/01/how-robots-will-eat-our-jobs/ (zitiert auf S. 15, 16).

[Gro88] D. D. Grossman. „Traffic control of multiple robot vehicles“. In: *IEEE Journal on Robotics and Automation* 4.5 (1988), S. 491–497 (zitiert auf S. 23).

[Hop17] M. Hoppe. personal communication. 2. Aug. 2017 (zitiert auf S. 15, 16).

[HPS08] G. Heineman, G. Pollice, S. Selkow. *Algorithms in a Nutshell*. In a Nutshell (O'Reilly). O'Reilly Media, 2008. ISBN: 9781449391133 (zitiert auf S. 19, 20).

[HS06] M. Hompel, T. Schmidt. *Warehouse management: automation and organisation of warehouse and order picking systems*. Springer Science & Business Media, 2006 (zitiert auf S. 12).

[JSHH07] P. Janert, J. Shakes, N. Hanssens, D. Hodge. *Time-based warehouse movement maps*. US Patent 7,243,001. Juni 2007 (zitiert auf S. 11).

[Kal12] R. Kala. „Multi-robot path planning using co-evolutionary genetic programming“. In: *Expert Systems with Applications* 39.3 (2012), S. 3817–3831 (zitiert auf S. 28, 29, 59).

[Kar13] J. Karásek. „An overview of warehouse optimization“. In: *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems* 2.3 (2013), S. 111–117 (zitiert auf S. 11).

[Kar17] P. Karich. *Now flexible routing is at least 15 times faster - GraphHopper Directions API*. 2017. URL: https://www.graphhopper.com/blog/2017/08/14/flexible-routing-15-times-faster/ (zitiert auf S. 19).

[Kaw16] Y. Kawano. *Automated warehouse system*. US Patent 9,315,320. Apr. 2016 (zitiert auf S. 11).

[KHC05] K. M. Krishna, H. Hexmoor, S. Chellappa. „Reactive navigation of multiple moving agents by collaborative resolution of conflicts“. In: *Journal of Field Robotics* 22.5 (2005), S. 249–269 (zitiert auf S. 30).

[Kir13] S. Kirsner. *Will Amazon-owned robot maker sell to e-tailer's rivals? - The Boston Globe*. 2013. URL: https://www.bostonglobe.com/business/2013/12/01/will-amazon-owned-robot-maker-sell-tailer-rivals/FON7bVNKvfzS2sHnBHzfLM/story.html (zitiert auf S. 11).

[KLB17] W. Kern, H. Lämmermann, T. Bauernhansl. „An integrated logistics concept for a modular assembly system“. In: (2017) (zitiert auf S. 11).

[Len93] T. Lengauer. *Algorithms - ESA '93: First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993. Proceedings*. Lecture Notes in Computer Science. Springer, 1993. ISBN: 9783540572732 (zitiert auf S. 17).

[MT12] B. Meindl, M. Templ. „Analysis of commercial and free and open source solvers for linear optimization problems“. In: *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS* (2012) (zitiert auf S. 20).

[NVB17] C. Nauseda, M. Verminski, T. Bragg. *Fabric pods*. US Patent 9,572,426. Feb. 2017 (zitiert auf S. 11).

[RL06] R. Regele, P. Levi. „Cooperative multi-robot path planning by heuristic priority adjustment“. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE. 2006, S. 5954–5959 (zitiert auf S. 28).

[Rya08] M. R. K. Ryan. „Exploiting subgraph structure in multi-robot path planning“. In: *Journal of Artificial Intelligence Research* 31 (2008), S. 497–542 (zitiert auf S. 12, 26, 27).

[SGG06] A. Silberschatz, P. Galvin, G. Gagne. *OPERATING SYSTEM PRINCIPLES, 7TH ED*. Wiley student edition. Wiley India Pvt. Limited, 2006. ISBN: 9788126509621 (zitiert auf S. 21).

[Sur12] P. Surynek. „Towards optimal cooperative path planning in hard setups through satisfiability solving“. In: *PRICAI 2012: Trends in Artificial Intelligence* (2012), S. 564–576 (zitiert auf S. 26).

[Tim14] Time. *Amazon's New Robots Are Shipping You Order This Holiday*. 2014. URL: http://time.com/3605924/amazon-robots/ (zitiert auf S. 15).

[WB+08] K.-H. C. Wang, A. Botea et al. „Fast and Memory-Efficient Multi-Agent Pathfinding“. In: *ICAPS*. 2008, S. 380–387 (zitiert auf S. 30, 40).

[WC11] G. Wagner, H. Choset. „M*: A complete multirobot path planning algorithm with performance bounds“. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, S. 3260–3267 (zitiert auf S. 24, 40).

[WDM08] P. R. Wurman, R. D'Andrea, M. Mountz. „Coordinating hundreds of cooperative, autonomous vehicles in warehouses“. In: *AI magazine* 29.1 (2008), S. 9 (zitiert auf S. 11).

[YL16] J. Yu, S. M. LaValle. „Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics“. In: *IEEE Transactions on Robotics* 32.5 (2016), S. 1163–1177 (zitiert auf S. 12, 25).

[YR15] J. Yu, D. Rus. „An effective algorithmic framework for near optimal multi-robot path planning“. In: *arXiv preprint arXiv:1505.00200* (2015) (zitiert auf S. 25).

[ZM02] L. Zhang, S. Malik. „The quest for efficient boolean satisfiability solvers“. In: *Automated Deduction—CADE-18* (2002), S. 313–331 (zitiert auf S. 21).

Alle URLs wurden zuletzt am 05. 07. 2017 geprüft.

7 Anhang

In diesem Kapitel werden die Performanz-Graphen der einzelnen Algorithmen gezeigt.

7.1 OMPP Performanz-Graphen

Die Abbildungen 7.1 bis 7.3 zeigen die Performanz von OMPP auf den drei verschiedenen Straßennetzen. Die Legende zu den Graphen befindet sich unter der ersten Abbildung.

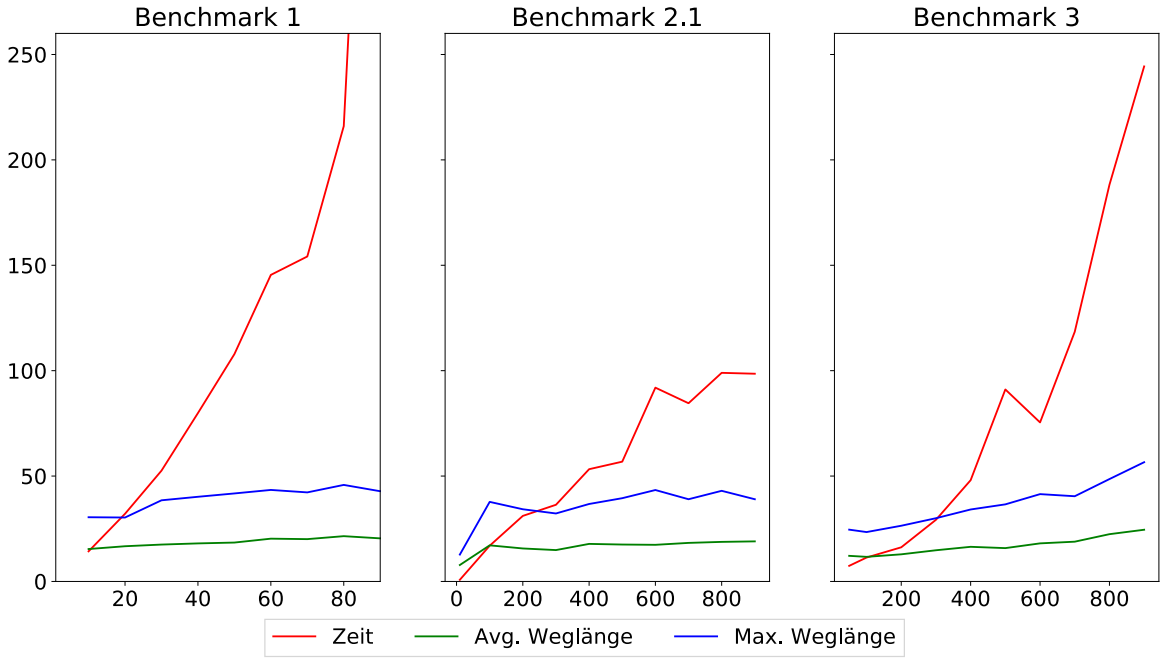

Abbildung 7.1: Algorithmus OMPP auf Straßennetz 1

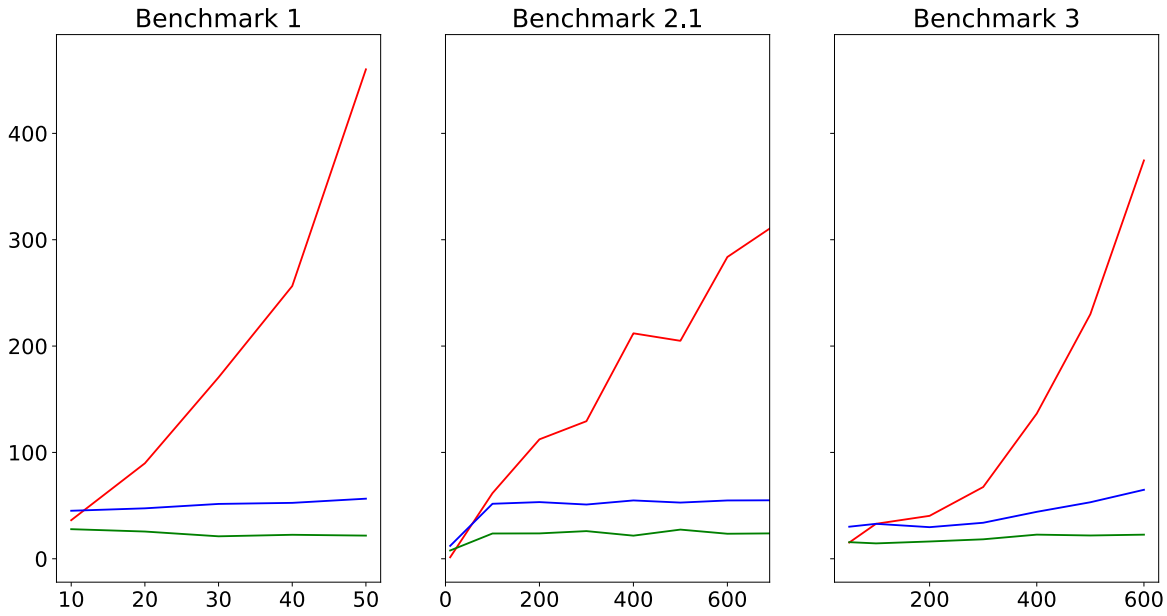

Abbildung 7.2: Algorithmus OMPP auf Straßennetz 2

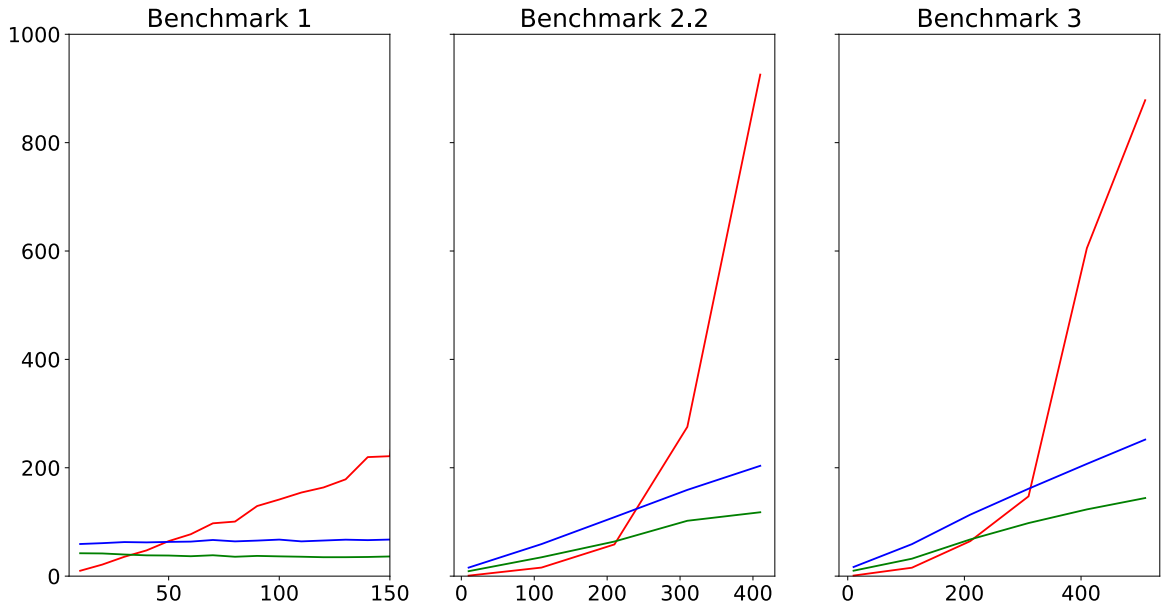

Abbildung 7.3: Algorithmus OMPP auf Straßennetz 3

7.2 M* Performanz-Graphen

In den Abbildungen 7.4 bis 7.6 ist die Performanz vom Algorithmus M* auf den drei verschiedenen Straßennetzen zu sehen. Die Legende zu den Graphen befindet sich unter der ersten Abbildung.

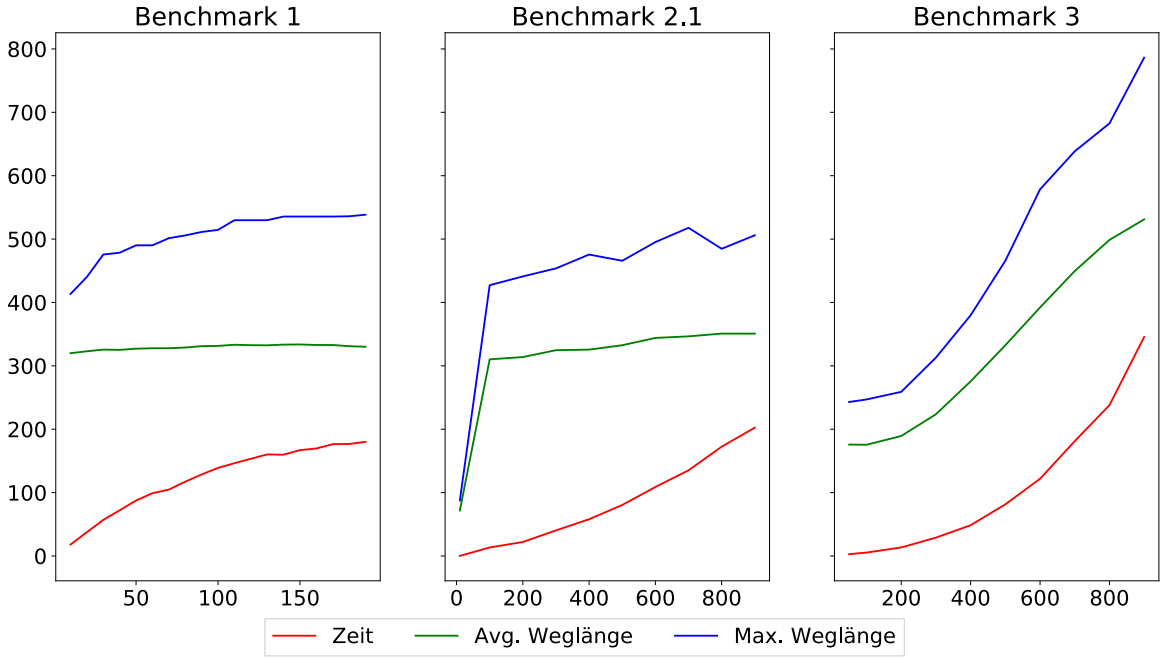

Abbildung 7.4: Algorithmus M* auf Straßennetz 1

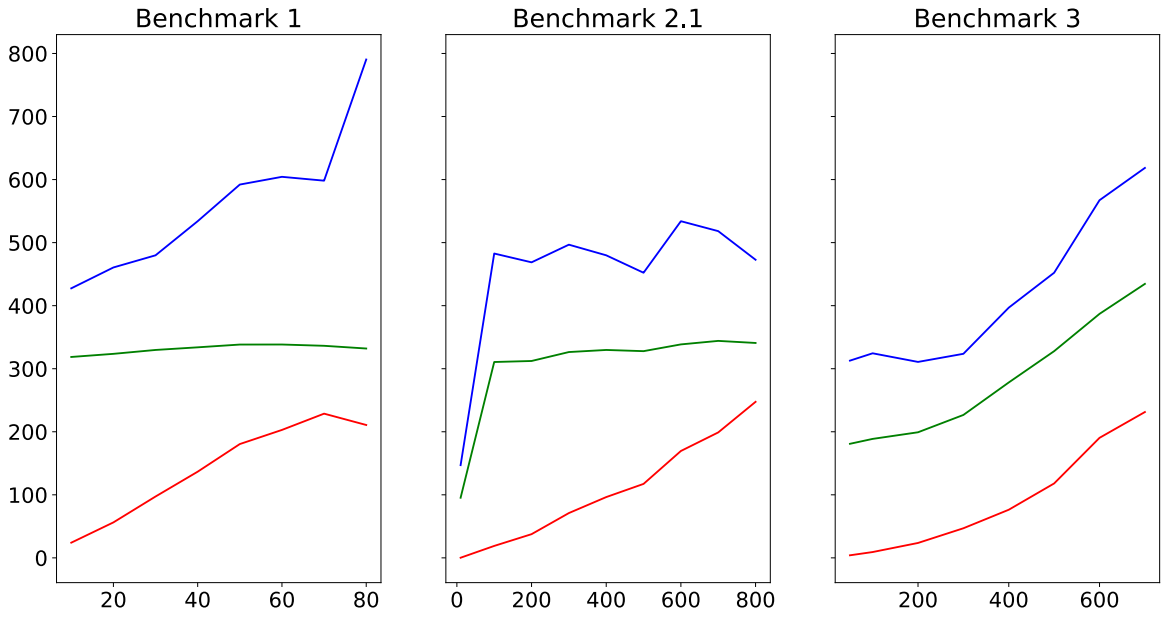

Abbildung 7.5: Algorithmus M* auf Straßennetz 2

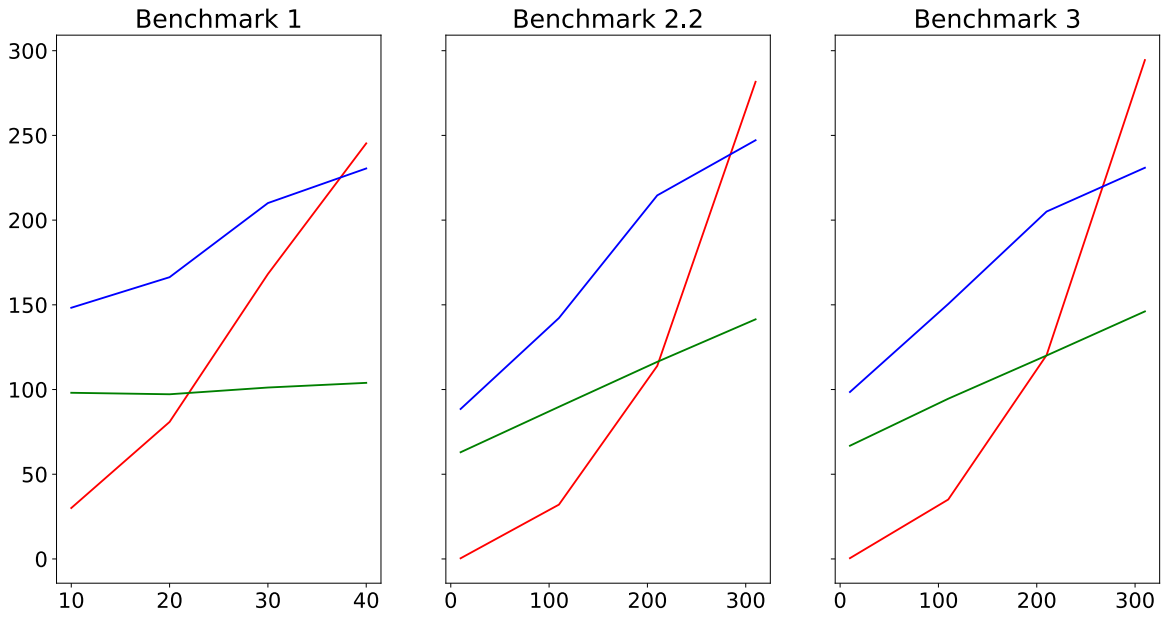

Abbildung 7.6: Algorithmus M* auf Straßennetz 3

7.3 FAR Performanz-Graphen

In den Abbildungen 7.7 bis 7.9 ist die Performanz vom Algorithmus FAR auf den drei verschiedenen Straßennetzen zu sehen. Die Legende zu den Graphen befindet sich unter der ersten Abbildung.

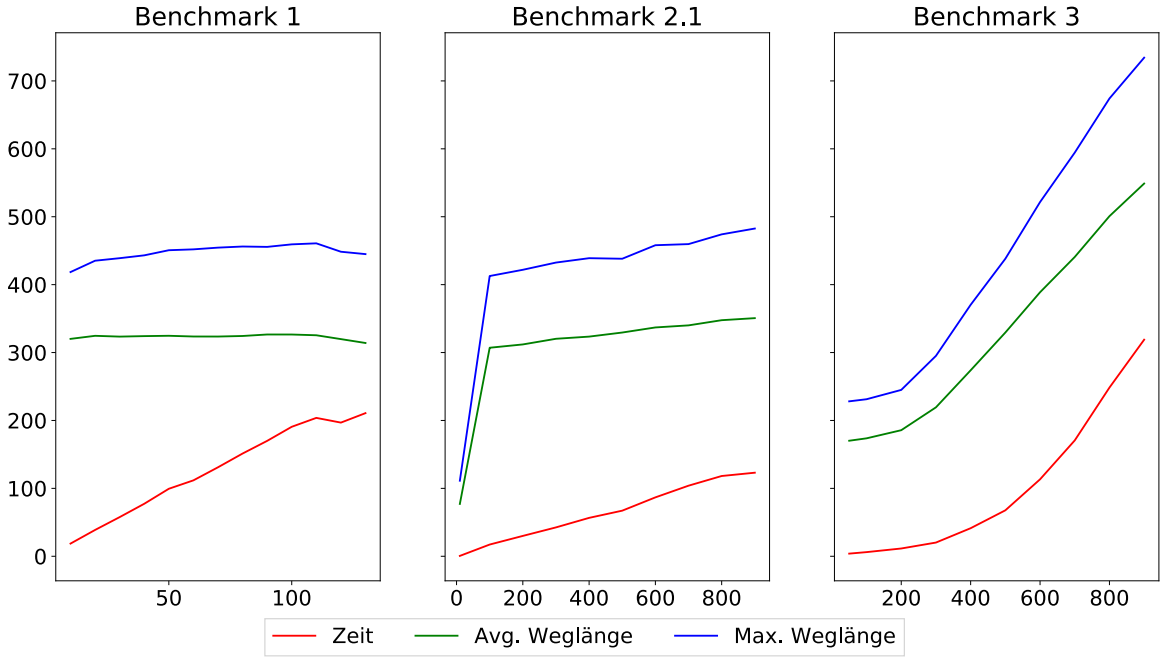

Abbildung 7.7: Algorithmus FAR auf Straßennetz 1

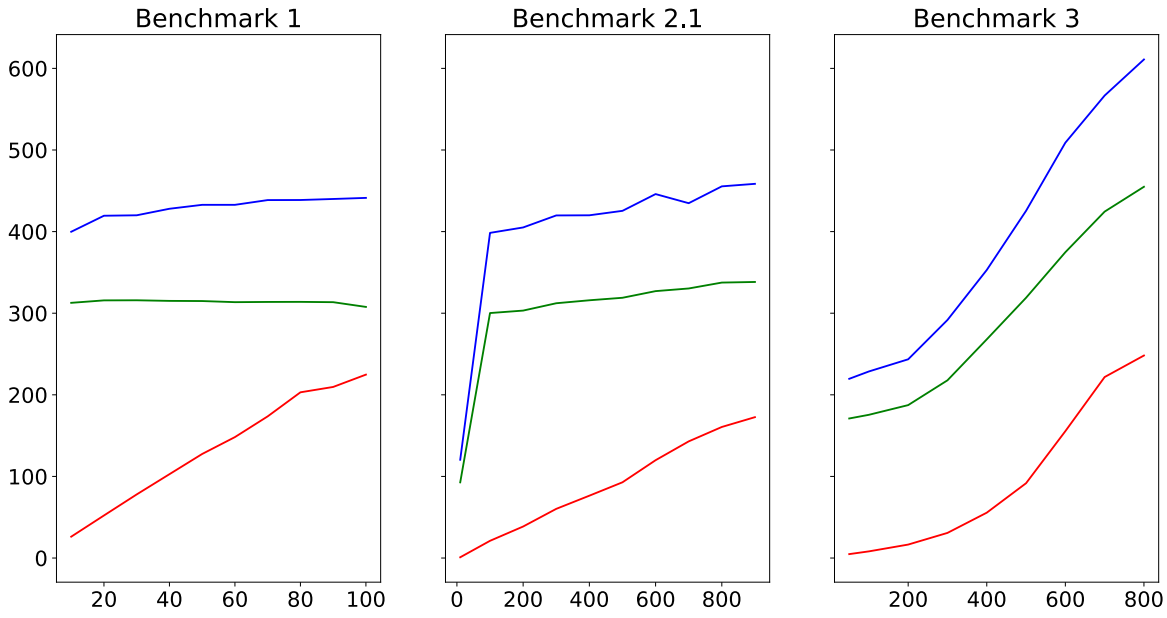

Abbildung 7.8: Algorithmus FAR auf Straßennetz 2

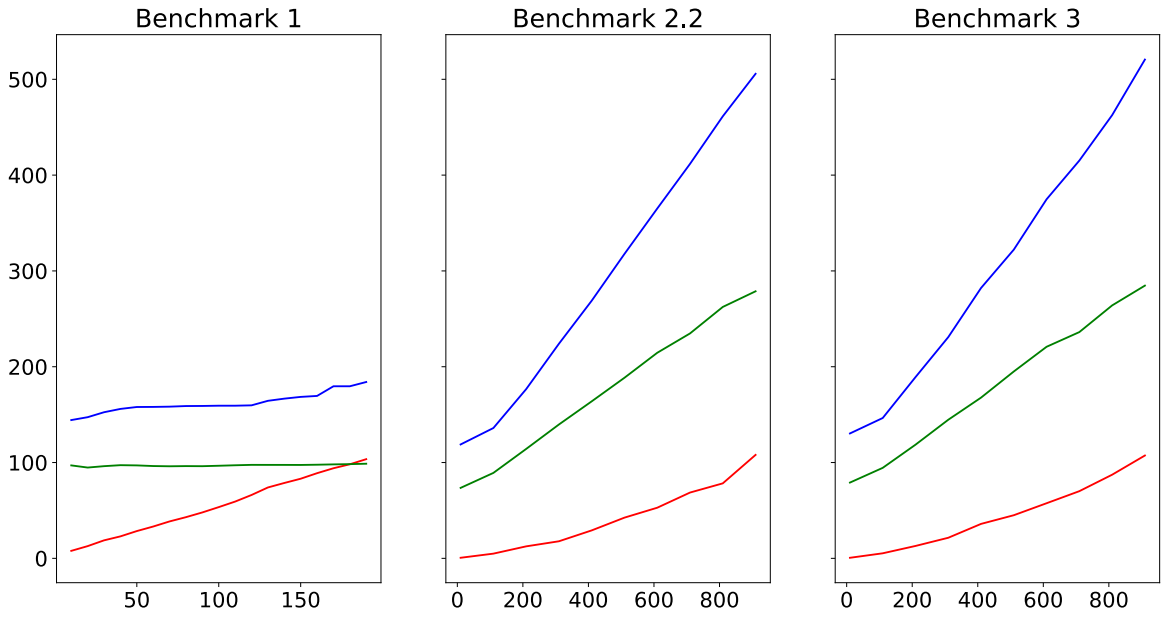

Abbildung 7.9: Algorithmus FAR auf Straßennetz 3

7.4 W* Performanz-Graphen

In den Abbildungen 7.10 bis 7.12 ist die Performanz vom Algorithmus W* auf den drei verschiedenen Straßennetzen zu sehen. Die Legende zu den Graphen befindet sich unter der ersten Abbildung.

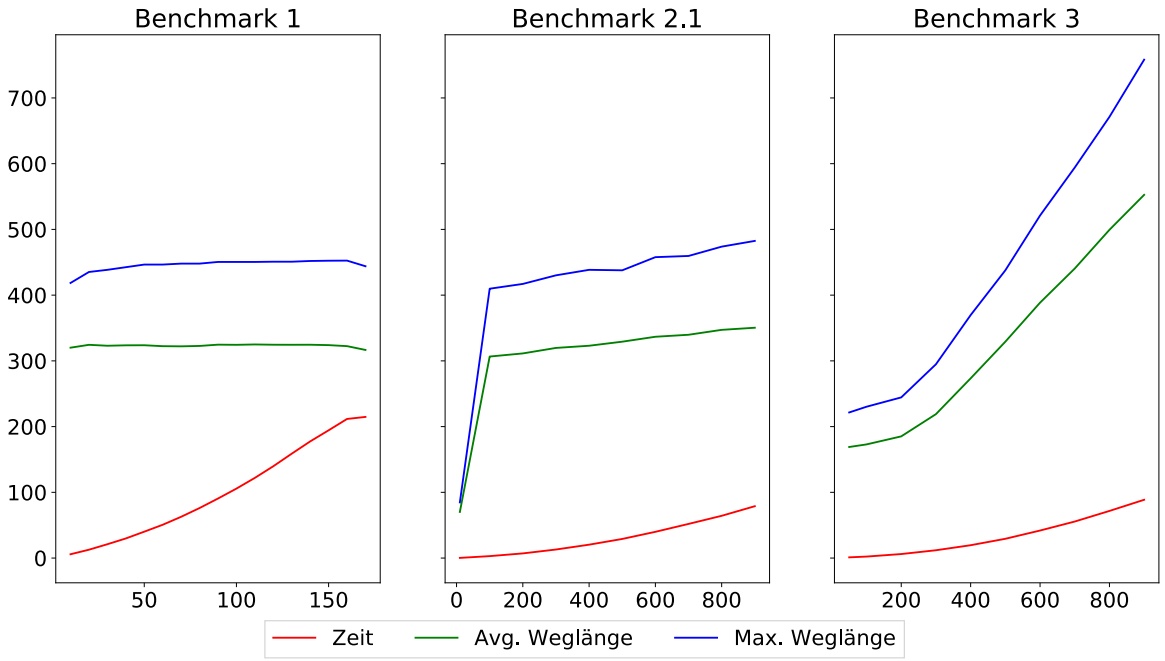

Abbildung 7.10: Algorithmus W* auf Straßennetz 1

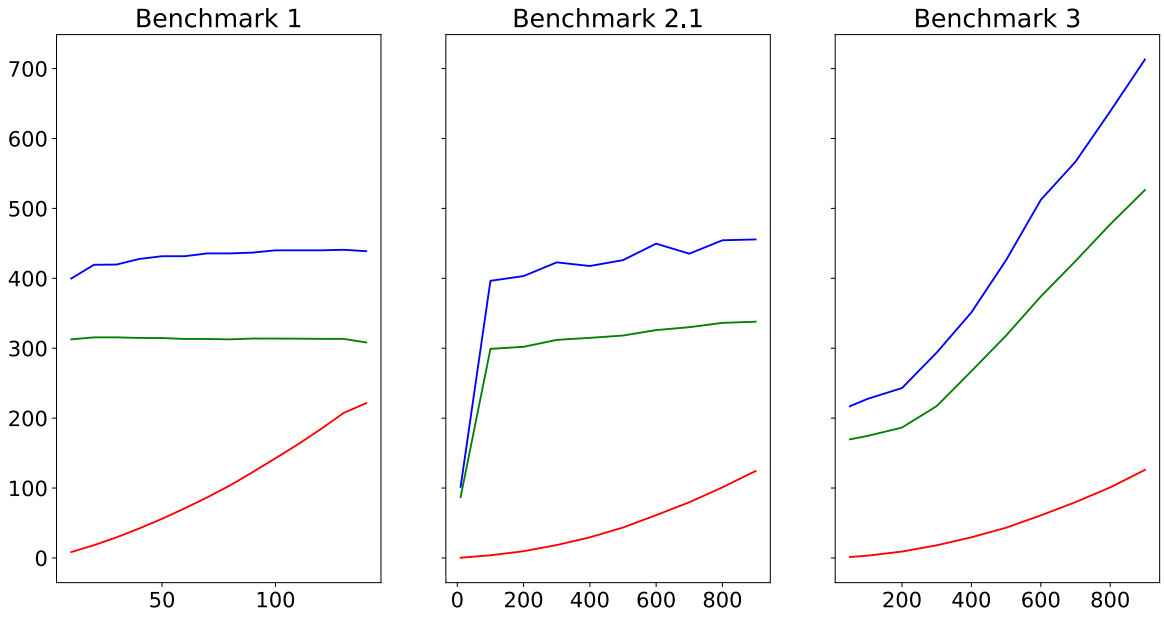

Abbildung 7.11: Algorithmus W* auf Straßennetz 2

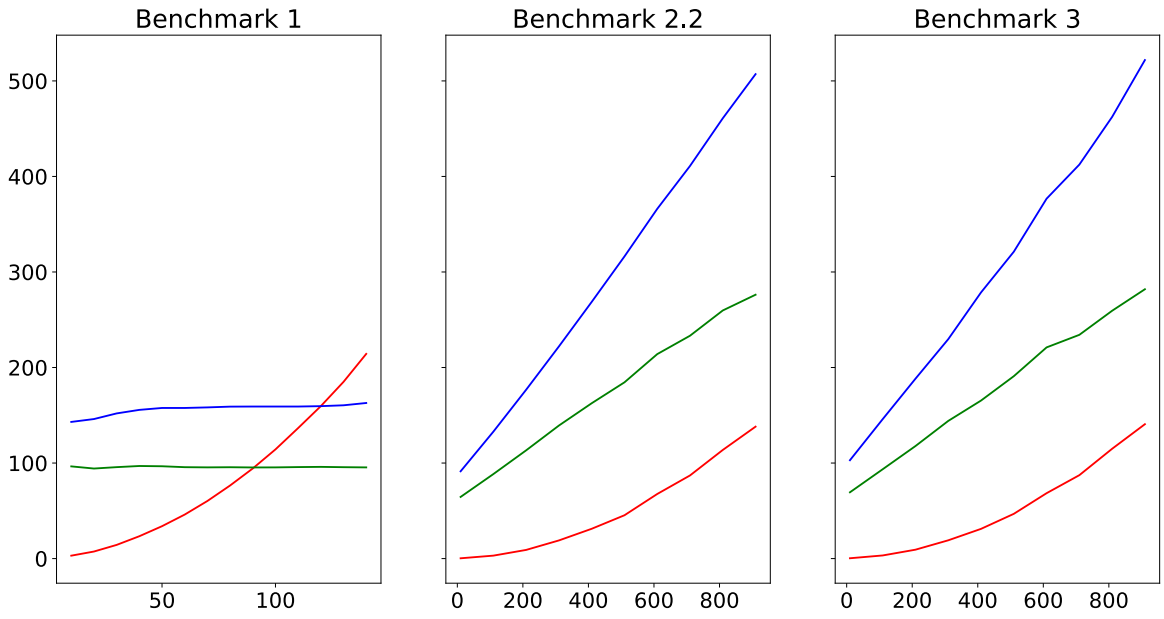

Abbildung 7.12: Algorithmus W* auf Straßennetz 3

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift