Carl von Ossietzky Universität Oldeburg
Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Ein Bienen-inspiriertes Schwarmintelligenz-Verfahren zum Routing im Straßenverkehr

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
vorgelegt von

Dipl.-Inf. Sebastian Senge

Oldenburg
2014

1. Gutachter: Prof. Dr. Axel Hahn, Universität Oldenburg
2. Gutachter: Prof. Dr. Günter Rudolph, Technische Universität Dortmund

Tag der Disputation: 17.11.2014
Kandidat: Dipl.-Inf. Sebastian Senge, geb. am 29.08.1981 in Freiburg i. B.

Danksagungen

Gaudeamus igitur,
iuvenes dum sumus, [...]

Vivat academia,
vivant professores,
vivat membrum quodlibet,
vivant membra quaelibet,
semper sint in flore!, [...]

(C.W. Kindleben, 1781)

Ich bedanke mich an dieser Stelle bei allen, die mich bei dieser Arbeit in Dortmund und Oldenburg unterstützt haben. Bei Herrn Prof. Dr. Hahn und Herrn Prof. Dr. Rudolph bedanke ich mich für die Erstellung der Gutachten. Prof. Dr. Steffen, Prof. Dr. Hahn, Prof. Dr. Rudolph und Prof. Dr. Lehnhoff danke ich für die nicht selbstverständliche Unterstützung während des Transfers der Arbeit an die Universität Oldenburg.

Entstanden ist diese Arbeit während meiner Zeit als wissenschaftlicher Mitarbeiter am Lehrstuhl III der Fakulktät Informatik an der TU Dortmund bei Herrn Prof. Dr. Wedde. Allen Studenten des Lehrstuhls, die ich betreuen durfte und die das Projekt weiter entwickelt haben, gilt mein Dank: Bastian Hoffmann, Florian Will, Niclas Raabe, Roland Kahlert, Tim Lohmann und Fabian Knobloch für ihre Abschlussarbeiten und Manuel Sträßer, Niclas Raabe und Robert Niehage für ihre ergiebige Arbeit als HiWis.

Mit dem benachbarten Lehrstuhl X kam es zu einem erfreulich befruchtenden Austausch: Dr. Stefan Dissmann, Dr. Ingo Battenfeld und Dipl.-Inf. Pascal Hof sei für die vielfältigen Gespräche über die und jenseits der Informatik gedankt.

Mein Dank gilt ebenso allen Korrekturlesern, allen voran Dipl.-Ing. agr. Annika Sasse, für ihre unendlich wertvolle Unterstützung.

Abkürzungen und Symbole

BeeJamA	Bee Jam Avoidance
FZ	Foraging Zone
FR	Foraging Region
(E)LCP	(Estimated) Least Cost Path
VRGS	Vehicle Route Guidance System
GRF	Generisches Routing Framework
LPF	Link Performance Function
ITS	Intelligent Transportation Systems
TMC	Traffic Message Channel
ADAS	Advanced Driver-Assistance System
PNA	Personal Navigation Assistant
MAS	Multi-Agent System
FFT	Free Flow Time
TIC	Traffic Information Center
TIOA	Timed I/O Automata
RSU	Road Side Unit
WSN	Wireless Sensor Network
MANET	Mobile AdHoc Network
VANET	Vehicular AdHoc Network
UE	User Equilibrium
SO	System Optimum
DUE	Dynamic User Equilibrium
DSO	Dynamic System Optimum
PoA	Price of Anarchy
ACP	Average Cost Pricing
CCP	Current Cost Pricing
MCP	Marginal Cost Pricing
PCP	Predicted Cost Pricing
TCP/IP	Transmission Control Protocol / Internet Protocol

b	Maximale Übertragungszeit auf einem Kommunikationskanal
γ	Durchmesser eines Graphen
$\delta(\pi)$	Kosten eines Pfads π
ω_{ij}	Gewicht der Kante (i, j), i.d.R. die Transitzeiten eines Links
ζ_{ij}	Kosten für eine Nachricht zur Passage der Kante (i, j)
$G = (V, L)$	Graph mit Knoten V und Links L
D, Q, V	Makroskopische(r) Dichte, Fluss und Geschwindigkeit
α	Bezeichner für ein Fahrzeug
v_α	Geschwindigkeit des Fahrzeugs α
η	Link Performance Function
i	Aktueller Knoten einer Nachricht oder Tokens (Datenpaket, Fahrzeug)
o	Scoutursprung
$A(i)$	Area des Knotens i
$\overrightarrow{S}_i^z$	Menge der Pfadknoten an i zu z
S_i, P_i, N_i	Menge der Nachfolger, Vorgänger bzw. Nachbarn von i
N_i^d	d-Umgebung von i
$a \to b$	Es existiert eine Kante (a, b)
$a \rightsquigarrow b$	Es existiert ein Pfad von a zu b
E_A, F_A	Menge der Entry-/ Forwardingknoten der Area A
$\mathfrak{A}_i, \mathfrak{N}_i, \mathfrak{E}_i$	Area-, Net-, ETA-Tabelle des Knotens i
$\min T_i^z$	Kleinster (Kosten-)Eintrag zu z der Routingtabelle T von i
$\arg\min T_i^z$	Pfadknoten mit geringsten Kosten von i zu z
σ_X	Varianz der Zufallsvariable X
$cov(X, Y)$	Kovarianz der Zufallsvariablen X, Y
$cv(X, Y)$	Varianzkoeffizient der Zufallsvariablen X, Y
$\rho(X, Y)$	Korrelationskoeffizient der Zufallsvariablen X, Y
$\ominus, \circledcirc$	Link-Aggregierungs-, Knoten-Aggregierungs-Operation
$\oplus, \oslash, \bigcirc\!\!\!\!\to$	Next Hop-Selektions-, Scout-Selektions-, Propagierungs-Operation
$C_\mathcal{S}, C_\mathcal{F}$	Kostenvariable des Scouts $\mathcal{S}$ bzw. Foragers $\mathcal{F}$

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im September 1908 rollte das erste Ford Model T vom Band. Es war das erste am Fließband gefertigte Automobil und führte – mit 15 Millionen produzierten Stück – zur Massenmobilisierung und der Verbreitung des Individualverkehrs. Innerhalb weniger Dekaden erreichte die Anzahl der Fahrzeuge weltweit Hunderte Millionen. Die tägliche Begleiterscheinung: Stau.

1.1. Motivation

Schätzungen [28] zufolge existieren etwa 800 Millionen Kraftfahrzeuge weltweit und es wird erwartet, dass sich diese Anzahl bis zum Jahr 2030 verdreifacht. Laut Interessenvertretung der spanischen Automobilindustrie entfallen gegenwärtig davon rund 260 Millionen Fahrzeuge auf Europa [131]. Allein in Deutschland sind davon rund 50 Millionen dieser Kraftfahrzeuge zugelassen, die laut Bundesministerium für Verkehr, Bau und Stadtentwicklung jeweils ca. 39 Kilometer pro Tag zurücklegen und insgesamt eine Jahresfahrleistung von ca. $7 \cdot 10^{14}$ Kilometer aufweisen [133]. Stau als negative Folge dieses enormen Verkehrsaufkommens hat vielfache sozioökonomische Auswirkungen, von denen drei besonders relevante hier genannt seien:

Erhöhte Fahrzeiten: Für das Jahr 2012 registrierte der ADAC in Deutschland 285.000 Staus mit einer Gesamtlänge von 595.000 Kilometer und einer Gesamtdauer von 230.000 Stunden [1]. Dem TomTom European Congestion Index 2012 [141] zufolge ist Stuttgart Deutschlands Stadt mit dem höchsten Stauaufkommen. Im Schnitt dauern Fahrten 33% länger gegenüber dem staufreien Zustand, in der abendlichen Rush Hour sogar 67%. Ein Pendler mit einer Fahrtdauer von 30 Minuten (bei staufreier Strecke) steht in Stuttgart der Studie zufolge 89 Stunden pro Jahr im Stau. Gemittelt über ganz Europa verlängern sich Fahrten aufgrund von Staus um 24%. Neben diesen offensichtlichen, objektiven Auswirkungen, stellen Verzögerungen und Lärm auch eine psychische, subjektive Belastung [62] dar.

Ökonomische Ausfälle: Summiert man die volkswirtschaftlichen Schäden, die durch Ausfälle von Arbeitszeit, verkehrsbedingten Verlustzeiten im Güterverkehr, erhöhtem Kraftstoffverbrauch und anderen Faktoren gegenüber einem staufreien Idealzustand entstehen, sind, gemäß des Instituts für Wirtschaftspolitik und Wirtschaftsforschung der Universität Karlsruhe, allein die europäischen Verluste auf fast 270 Milliarden Euro pro Jahr zu schätzen, was ca. 3% des europäischen Bruttoinlandsproduktes entspricht [118].

Umweltbelastungen: Neben ökonomischen Ausfällen sind auch die Kosten durch Luftverschmutzungen, Folgekosten des Verkehrslärms und die Auswirkungen auf das Klima zu beachten. Würde all diesen Belastungen vollumfänglich entgegengewirkt, entstünden, der letztgenannten Studie folgend, geschätzte europäische Gesamtkosten von 650 Milliarden Euro pro Jahr, was einem Anteil von 7,3% des europäischen Bruttoinlandsprodukt entspricht [118].

Insgesamt wird Stau als „verkehrspolitisches Primärproblem“ eingestuft [66]. Dabei kann in urbanen Umgebungen die Infrastruktur für motorisierten Verkehr kaum noch verbessert werden. Zusätzliche Straßen benötigen Raum, welcher in der Regel nicht mehr zur Verfügung steht. Immer häufiger konfligieren bei infrastrukturellen Großprojekten die Ansprüche und Bedürfnisse der Bevölkerung und des Bauherrn. Ein infrastruktureller Ausweg ist demnach nicht in Sicht. Ganz im Gegenteil, aufgrund der zu erwartenden Erhöhung des Personen- und Lastverkehrs weltweit ist mit einer Verschärfung der oben genannten Probleme zu rechnen.

Kann die gegenwärtige Stauproblematik und die dräuende Verschärfung nicht infrastrukturell angegangen werden, drängt sich notwendigerweise die folgende Effizienzfrage auf: *Können durch geeignete Wahl von Fahrstrecken geringere Fahrzeiten, weniger Staus, geringere Umweltbelastungen, geringerer Kraftstoffverbrauch, oder allgemeiner, Verbesserungen bezüglich festgelegter Kriterien erreicht werden?*

Mittels klassischer Optimierungsverfahren wurde in der Verkehrsplanung gezeigt, dass dies prinzipiell möglich ist (näheres hierzu in Kapitel 4). Deutlicher Nachteil dieser klassischen Verkehrsoptimierungsstrategien ist jedoch, dass erhebliche Voraussetzungen erfüllt sein müssen. So müssen z.B. sämtliche Informationen über die Verkehrsteilnehmer (wann und wo starten wie viele Fahrzeuge mit welchem Ziel) und die Auslastung des Verkehrsnetzes zu jedem Zeitpunkt bekannt sein. Ferner muss a priori bekannt sein, wie lange Fahrzeuge bei einer bestimmten Auslastung exakt für bestimmte Straßenabschnitte benötigen. Anhand dieser Daten lässt sich sodann eine Offline-Optimierung durchführen, d.h. vorab des Fahrantritts. Berechnet werden dabei einmalig die optimalen Pfade aller Fahrzeuge zur Minimierung der Fahrzeiten. Träten keinerlei Veränderungen zu dem vorausberechneten Ablauf ein, würde dies eine passable Lösung darstellen.

Jedoch ist der Straßenverkehr ein Paradebeispiel eines *offenen Systems*: A priori ist kein definitives Wissen über Verkehrsteilnehmer bekannt. Im Gegenteil, jedem Verkehrsteilnehmer steht es frei, das System zu jedem beliebigen Zeitpunkt zu betreten bzw. zu verlassen, d.h. eine Fahrt anzutreten, sich während der Fahrt bzgl. des Ziels umzuentscheiden oder eine Fahrt abzubrechen. Auch ist der Straßenverkehr zu nicht geringen Anteilen beeinflusst von unvorhersagbaren, aber relevanten Ereignissen wie Unfällen und Witterungseinflüssen. Zwar ließen sich gewisse Prognosen treffen und Vorhersagen anstellen, nur ist deren Genauigkeit nicht zwangsläufig hoch. Anders ausgedrückt: Selbst wenn ein möglichst präziser Startzustand bekannt wäre, der potentiell positive Effekt einer Offline-Optimierung wäre aufgrund des hohen dynamischen Charakters des Straßenverkehrs sehr schnell verpufft. Eine kontinuierliche, an die veränderte Situation angepasste, Wiederholung der Optimierung ist aber aufgrund des hohen Berechnungsaufwandes nicht realisierbar.

Stattdessen ist es sinnvoll, zur Laufzeit kontinuierlich eine individuelle, auf aktuellen Verkehrsdaten basierende, Wegfindung und -führung durchzuführen. So kann auf unvorhergesehene Ereignisse für jedes Fahrzeug einzeln während der Fahrt reagiert werden. Je weniger Voraussetzungen dabei einfließen und notwendig sind, desto flexibler kann das System reagieren. Es gab in der Vergangenheit vielfältige Bestrebungen, solch eine Online-Wegfindung zu ermöglichen. Ver-

kehrsmeldungen im Radioprogramm, heute noch weitverbreitet, können als der erste Versuch gewertet werden, Verkehrsteilnehmern während der Fahrt Verkehrsinformationen zukommen zu lassen. Allerdings müssen die Fahrer diese Meldungen manuell in ihre Routenplanung einbinden. Anschließend gab und gibt es kommerzielle Anbieter, welche ihre Navigationsgeräte mit Zusatzfunktionen ausstatten, die automatisch während der Fahrt relevante Verkehrsinformationen erhalten und die Wegfindung dementsprechend anpassen. Technisch passiert in beiden Fällen dasselbe: In einem sogenannten *Traffic Information Center* (TIC) werden alle relevanten Verkehrsdaten *zentral* gesammelt, diese ggf. aufgearbeitet und dem Fahrer zur Verfügung gestellt. Neben dem manuellen Weg über das Radioprogramm existiert mit Traffic Message Channel (TMC) [69] ein standardisiertes Verfahren und mit Systemen wie bspw. TomTom HD Traffic [140] proprietäre Systeme zur automatischen Übermittlung. Navigationssysteme inkorporieren diese Daten und bestimmen Routen mit möglichst geringer Fahrzeit. Wesentliches Unterscheidungsmerkmal dieser Systeme ist die Aktualisierungsfrequenz der Verkehrsinformation. So bieten diese Systeme Aktualisierungsfrequenzen in der Größenordnung von mehreren Minuten: TomTom HD Traffic bietet in manchen Gebieten eine Frequenz von zwei Minuten, TMC je nach Betriebsmodi zwischen 15 und 30 Minuten. Zudem sind die abgedeckten Gebiete eingeschränkt. Ursächlich und begrenzender Faktor ist die zentrale Verfügbarkeit und Verarbeitung sämtlicher Informationen. Das TIC stellt dabei einen Flaschenhals dar, welcher einer zentralen Sammlung und Verbreitung inhärent zu eigen ist und sich konzeptionell nicht auflösen lässt. Resultierend können die im System bekannten Informationen veraltet und damit – im günstigsten Falle – nutzlos sein. Im schlechtesten Falle sogar kontraproduktiv, falls Fahrzeuge über einen Umweg geleitet werden, der noch als frei betrachtet wird, obwohl dieser bereits überlastet ist. Auch umgekehrt können Verschlechterungen eintreten, wenn freie, günstige Routen fälschlicherweise nicht verwendet werden können.

Das Ziel dieser Arbeit ist, den Flaschenhals der zentralen Verarbeitung vermöge eines verteilten Systems zu beseitigen. Ein verteiltes System nimmt davon Abstand, globale Daten zu erheben oder zu sammeln, eliminiert so den Single Point of Failure und ermöglicht daher konzeptionell eine bessere Skalierbarkeit in Raum und Zeit. Angestrebt wird ein möglichst großes Gebiet abdecken zu können und eine Aktualisierungsfrequenz im Sekundenbereich zu erreichen. Der nächste Abschnitt geht auf diese Ziele genauer ein.

1.2. Ziele, Vorgehensweise und Beiträge

Das übergeordnete Ziel dieser Arbeit ist die Entwicklung eines dynamischen *Vehicle Route Guidance Systems* (VRGS) [68] zur Reduzierung von Fahrzeiten im Straßenverkehr mittels *verteilter Kontrolle*. Ein VRGS, im simpelsten Falle ein klassisches Navigationssystem, dient der individuellen Wegfindung und -führung eines einzelnen Fahrzeugs, ggf. koordiniert mit anderen Fahrzeugen, vom Ausgangspunkt zum Ziel. Ein VRGS ist dynamisch, wenn zur Fahrzeit kontinuierlich aktualisierte Verkehrsinformationen in die Wegfindung mit einbezogen werden.

Genauer besteht ein VRGS aus zwei Teilen: Zum Einen aus einem Routingalgorithmus (allgemeiner: -protokoll) und zum Anderen aus den notwendigen infrastrukturellen, ggf. vernetzten, Hardware-Komponenten. Im Falle eines klassischen, nicht-dynamischen Navigationssystems ist die On Board Unit die einzige Hardware-Komponente des Systems. Vernetzte Systeme, die zur Fahrzeit aktuellere Verkehrsdaten beziehen, benötigen zusätzliche Komponenten außerhalb des Fahrzeugs.

Anders als kommerziell verfügbare dynamische VRGS, soll in dem in dieser Arbeit entwickelten BeeJamA-Ansatz jedoch keine zentrale Instanz, wie das bereits erwähnte Traffic Information Center (TIC) zur Sammlung und Verarbeitung von globalen Informationen notwendig sein. Stattdessen soll ein vollständig *verteilter Ansatz* nur unter Ausnutzung von stets aktuellen lokalen Informationen die Wegfindung ermöglichen. Verteilte Systeme bieten sich unter Anderem stets dann an, wenn eine zentrale Steuerung und/oder Koordinierung gegebene Ressourcen überfordert. Im Straßenverkehr ist schlussendlich genau von dieser Situation auszugehen. Denn je größer das System wird, desto mehr Daten muss das TIC verwalten. Ab einer bestimmten Systemgröße sind die Kapazitäten des TIC unausweichlich erschöpft. Verteilte Systeme besitzen potentiell die Möglichkeit, diese Grenzen der Skalierbarkeit zu verschieben, wenngleich auch nicht vollständig zu beseitigen. Denn auch in einem verteilten System steigt der Aufwand (in der Regel die Anzahl der zur verteilten Kommunikation nötigen Nachrichten, die sog. Nachrichtenkomplexität) mit der Größe der Eingabeparameter. Eine vollständige Entkoppelung von der Systemgröße ist demnach auch in verteilten Systemen nicht möglich. In der Praxis ergibt sich jedoch erhebliches Einsparpotential durch Bildung kleinerer, möglichst autonomer Teilsysteme, die nur mit ihren benachbarten Teilsystemen die notwendigsten Informationen austauschen.

Das *Bottom Up*-Prinzip ist ein wesentliches Paradigma bei Systemen unter verteilter Kontrolle: Statt durch eine zentral berechnete Strategie, ergeben sich die Systemziele durch dezentrale Interaktion der Teilsysteme. Statt einem für die zentrale Berechnung notwendigen globalen Weltbild, stehen nur lokale, mitunter unvollständige oder gar fehlerhafte Informationen zur Verfügung. Nicht die optimale Lösung steht zwingend im Vordergrund, sondern die hinreichende. Vorteil dieser Vorgehensweise ist, dass die Abhängigkeit von einem singulären, zentralen Berechnungsprozess als monolithischen Single Point of Failure eliminiert wird. Sind zeitliche Nebenbedingungen einzuhalten, kann die rein lokale Interaktion mitunter sogar die einzige Möglichkeit sein ebendiese einzuhalten. Zentrale Berechnungen, zum Teil allein schon das Sammeln aller Informationen, würde in diesen Fällen erheblich zu lange dauern. Im Falle eines dynamischen VRGS ergeben sich zeitliche Nebenbedingungen, nämlich Deadlines dadurch, dass Verkehrsinformationen sinnvollerweise nur eine begrenzte Gültigskeitsdauer besitzen. Dauern die Berechnungen zu lange, ist die Information ggf. veraltet. Allein die zentrale Sammlung der gegenwärtigen Auslastungsinformationen eines großen Straßennetzes, bspw. von ganz Deutschland, kann in der Praxis schon erhebliche Verzögerungen mit sich bringen. Da diese Daten aber dezentral anfallen, liegt eine dezentrale Verwendung nahe.

Der in dieser Arbeit vorgeschlagene verteilte Ansatz sei als BeeJamA (für Bee Jam Avoidance) bezeichnet. Der Name bezeichne dabei das gesamte VRGS, im Speziellen aber auch das eingesetzte Routingprotokoll. Wo der Kontext eindeutig ist, wird der Name für die Verwendungsmöglichkeiten synonym verwendet. Ansonsten wird zwischen BeeJamA-VRGS, BeeJamA-Infrastruktur und BeeJamA-Protokoll unterschieden.

Das BeeJamA-VRGS setzt das Paradigma der verteilten Kontrolle um, indem zunächst eine geographische Unterteilung des Verkehrsnetzes vorgenommen wird. Darauf aufbauend werden mittels eines Multi-Agenten Systems (MAS) [167] lokale Informationen verbreitet werden, die wiederum in lokalen Entscheidungsprozessen zur individuellen Routenfindung Verwendung finden. Die geographische Unterteilung in sogenannte *Areas* dient der Schaffung lokaler Umgebungen, die in sich geschlossen und jeweils autark arbeiten können. Solche Areas werden auf einer höheren, logischen Ebene zusammengeschlossen, um ein Routing über weitere Entfernungen zu ermöglichen. Dabei werden nur partielle Informationen übertragen, indem nur benachbarte Areas Daten austauschen – nirgends werden globale Daten angesammelt.

Die prinzipielle Strategie des VRGS lautet, die Fahrzeuge individuell über den zum aktuellen Zeitpunkt jeweils günstigsten Pfad weiterzuleiten. Durch häufige Inkorporation aktueller Fahrzeiten über verschiedene Routen zum Ziel soll auf dynamische Situationen reagiert werden. So werden vorherige *Routenempfehlungen* an aktuelle Verkehrszustände angepasst. Sozusagen wird eine kontinuierliche Korrektur der Routenplanung durchgeführt.

Um die dafür nötigen Informationen effizient im Netz zu verbreiten, nimmt das MAS Anleihen bei dem schwarmintelligenten Futtersuchverhalten der Honigbienen. Dieses natürliche Schwarmverhalten ist ein gutes Beispiel des zuvor genannten Bottom-Up-Prinzips. Ohne zentrale Kontrollinstanz reagieren die Bienen effizient auf die kontinuierlichen Änderungen in ihrer dynamischen Umgebung, indem eine Vielzahl von Kundschafterinnen die Umgebung des Bienenstocks explorieren und Arbeiterinnen vermittelt über den Bienentanz (unter bestimmten Voraussetzungen) die gefundenen und angepriesenen Futterquellen ausbeuten. Die lokale Interaktion zwischen den Bienen allein bestimmt das Gesamtverhalten des Schwarms und sichert die Erreichung des übergeordneten Gesamtziels, der Sammlung von ausreichend Futter.

Die vorliegende Arbeit liefert Beiträge zu zwei Gebieten. Zum Einen wird das bereits skizzierte Routingverfahren basierend auf einem schwarmintelligenten MAS beschrieben und evaluiert. Das VRGS verfolgt dabei im Einzelnen die folgenden vier Ziele:

1. Das Verfahren arbeitet verteilt, d.h. es unterlässt die Sammlung und Verwendung von globalen Daten, und erlaubt (daher) eine *hohe Skalierbarkeit in Raum und Zeit.*
2. Jedes Fahrzeug soll *rechtzeitig vor jeder Kreuzung* eine aktualisierte Routenempfehlung basierend auf aktuellen Verkehrsinformationen erhalten. Jedem Fahrzeug wird dabei genau der nachfolgende Straßenabschnitt als Empfehlung gegeben (Next Hop), welcher auf dem gegenwärtig (vermeintlich) günstigsten Pfad (Least Cost Path, LCP) zum Ziel liegt. Das Fahrzeug kann sich so rechtzeitig einordnen und der Empfehlung entsprechend abbiegen.
3. Eine *Aktualisierungsfrequenz* für große Systeme im Sekundenbereich ist möglich.
4. Das Verfahren ist *kompetitiv*, d.h. die Fahrzeuge erreichen ihr Ziel gleich schnell, besser schneller, als mit verbreiteten zentralen Systemen.

Das in dieser Arbeit vorgestellte VRGS ist die erste Umsetzung dieser spezifischen Ziele.

Zur Evaluierung wurde im Zuge dieser Arbeit, das stellt den zweiten Beitrag dar, ein Simulations- und Evaluationsumgebung geschaffen. Diese Umgebung, genannt Generic Routing Framework (GRF), stellt eine Middleware dar, die es erlaubt, Routingprotokolle mit unterschiedlichen Verkehrssimulatoren zu evaluieren, ohne die Routingprotokolle an die unterschiedlichen Entwicklungsparadigma der Simulatoren anzupassen. Stattdessen wurden Adapter zur Anbindung dreier Simulatoren an das GRF entwickelt. Allerdings stellte sich heraus, dass nur einer dieser Simulatoren zum jetzigen Zeitpunkt geeignet ist, im Rahmen dieser Arbeit die Evaluation von Routingprotokollen durchzuführen.

Der nächste Abschnitt gibt eine Übersicht über das BeeJamA-Verfahren, der übernächste Abschnitt beschreibt den weiteren Aufbau der Arbeit.

1.3. Übersicht des BeeJamA-Ansatzes

Die Abbildung 1.1 vermittelt einen ersten Eindruck der Funktionsweise des BeeJamA-VRGS. Dargestellt ist ein Straßennetz, mit ausgezeichnetem Start- und Zielknoten. Das Fahrzeug

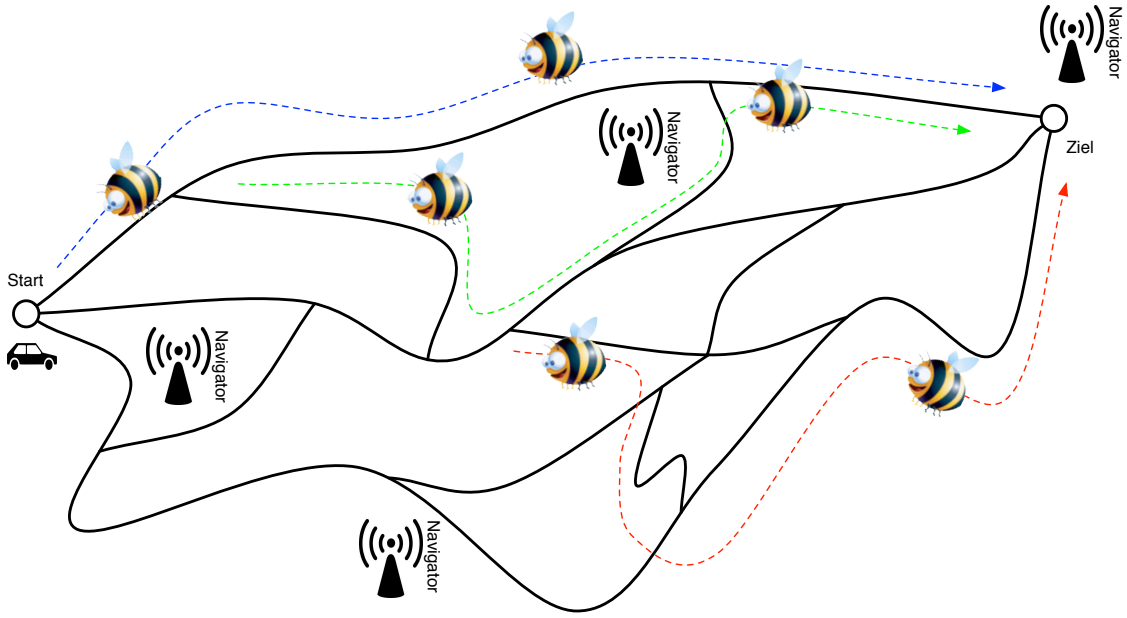


Abbildung 1.1.: Grundprinzip des BeeJamA-VRGS

ist möglichst „kostengünstig“ unter Berücksichtigung der dynamischen Streckenwahl übriger (nicht eingezeichneter) Verkehrsteilnehmer durch das Netz zum Ziel zu leiten. Die VRGS-Infrastruktur besteht aus sogenannten *Navigatoren*, hier zunächst visualisiert als vier Road Side Units (RSU), d.h. als am Straßenrand fest installierte, mit Funkhardware ausgestattete, lokale Infrastrukturkomponenten. Navigatoren teilen das Netz in die bereits genannten *Areas* auf, die ihren jeweiligen Einflussbereich darstellen. Aufgabe der Navigatoren ist es, möglichst aktuelle Verkehrsdaten innerhalb ihrer Area zu sammeln und zu verwalten. Diese aktuellen Daten werden mittels vom Navigator ausgehender Agenten in einer lokalen Umgebung verbreitet, so dass inzidente Navigatoren in die Lage versetzt werden, Fahrzeuge zum Ziel zu leiten. Das Verhalten dieser Agenten ist inspiriert durch das Schwarmverhalten der Honigbienen bei der Futtersuche. Die in dieser Abbildung als stilisierte Bienen dargestellten Agenten, verbreiten ausgehend vom Zielknoten des Fahrzeugs kontinuierlich Richtungsanweisungen und aktuelle Kosten zum Ziels.

Angenommen die schnellste Route zum Ziel für das Fahrzeug sei zunächst der obere Pfad. Die Agenten verbreiten diese Informationen und der nahe des Startknotens gelegene Navigator empfiehlt dem Fahrzeug auf Anfrage, der blauen Richtungsmarkierung zu folgen. Weiter angenommen, dass der Zustand im Straßennetz sich schnell ändert, so dass die blaue Route bald schon nicht mehr den schnellsten Weg darstellt. Das Fahrzeug stellt aber an jeder Kreuzung eine neue Anfrage an den jeweils lokalen Navigator, so dass dieser anhand aktuellster Verkehrsdaten mit einer neuen Routingempfehlung antworten kann. Voraussetzung hierfür ist, dass die Agenten schnell und häufig genug die jeweiligen Informationen verbreiten. Eine zentraler Flaschenhals zur Sammlung globaler Verkehrsdaten ist demnach strengstens zu vermeiden, um potentiellen Verarbeitungsengpässen vorzubeugen.

Hat sich bspw. auf der oberen Route ein Stau gebildet, kann das Fahrzeug so bereits an der ersten Kreuzung auf die grüne wechseln. Angenommen, dass auch diese Route sich verteuert, so würde später auf die rote Route ausgewichen und das Ziel so schließlich erreicht. Stehen

die Informationen nicht schnell genug zur Verfügung, müssen Fahrzeuge unter Umständen suboptimale Routen nutzen.

Neben diesen Aspekten werden im weiteren Verlauf noch zusätzliche Konzepte zur Bildung von Hierarchie und Pfadreservierungen eingeführt.

Aber schon an diesem einführenden Beispiel werden zwei Punkte deutlich: 1) BeeJamA kennt kein TIC, denn kein Navigator kumuliert globale Daten. Es werden nur lokale Informationen benötigt. 2) Die möglichst schnelle Verbreitung der aktuellen Verkehrsdaten beeinflusst den Durchsatz des Verkehrsnetzes.

1.4. Gliederung und Aufbau

Nachdem in den beiden vorherigen Abschnitten eine Motivation des Themengebietes und die Zielsetzung dieser Arbeit dargelegt wurde, sollen im nächsten Abschnitt die benötigten graphentheoretischen Begriffe eingeführt, sowie einige Erklärungen zu dem vorkommenden Pseudocode gegeben werden. Die Notwendigkeit ergibt sich aus den in der Literatur unterschiedlich gebräuchlichen Definitionen und Vorgehensweisen. Die spätere Beschreibung der Routingprotokolle benötigt aber eine exakte Basis.

Anschließend werden in Kapitel 2 verwandte Arbeiten diskutiert und eine Abgrenzung und Einordnung dieser Arbeit vorgenommen. Rubriziert werden kann diese Arbeit in das Themengebiet der *Intelligent Transportation Systems* (ITS), welches dort näher vorgestellt wird. Auch ein vorläufiger Vergleich unmittelbar konkurrierender Ansätze wird dort gegeben, ein ausführlicherer folgt nach detaillierter Vorstellung des BeeJamA-Protokolls.

Im Rahmen dieser Arbeit wurde eine Evaluationumgebung, genannt *Generic Routing Framework* (GRF) geschaffen, welche in Kapitel 3 beschrieben wird. Dort werden ebenso die verwendeten Verkehrsimulatoren und die notwendigen Grundlagen des Straßenverkehrs diskutiert.

In dem anschließenden Kapitel 4 wird die klassische Offline-Optimierung erläutert, wie sie in der Verkehrsplanung angewandt wird. Wenngleich diese Konzepte nicht unmittelbar auf das in dieser Arbeit diskutierte Problem der Online-Routenführung zu übertragen sind, werden dort wesentlichen Grundlagen und Begriffe eingeführt. Zudem erlaubt das Verständnis der Optimierungsansätze eine Einschätzung der Leistungsfähigkeit der Online-Verfahren. Auch liefern diese klassischen Vorarbeiten wichtige Impulse für Ideen zur Weiterentwicklung des geschilderten BeeJamA-Protokolls.

Kapitel 5 ist den Grundlagen verteilter Systeme gewidmet. Die Begriffe *verteilte Systeme* und *verteilte Algorithmen* werden konkretisiert, klassische Routingprotokolle (für Computernetze) erläutert und schließlich die Begriffe *Multi-Agenten Systeme* und *Schwarmintelligenz* auf diesen Fundamenten diskutiert. Es wird verdeutlicht, wie die genannten Begriffe zusammenhängen und wo genuine Unterschiede existieren.

Das Kapitel 6 führt schließlich das BeeJamA-Protokoll detailliert ein. Dazu wird zunächst die notwendige ITS-Architektur vorgestellt, um das BeeJamA-Protokoll im Straßenverkehr umzusetzen. Weiter wird des Protokolls natürliches Vorbild, das Futtersuchverhalten der Honigbienen, dargestellt und auf die Domäne des Straßenverkehrs übertragen. Anschließend wird eine Basisvariante des BeeJamA-Protokolls beschrieben. Zusätzliche Erweiterungen, wie bspw. Pfadreservierungen, werden ebenfalls vorgestellt. Eine Diskussion, eine Einordnung und ein detaillierter Vergleich konkurrierender Ansätze beschließen das Kapitel.

Das anschließende Kapitel 7 ist der simulativen Evaluation des BeeJamA-Protokolls mittels

des GRF gewidmet. Zunächst wird der Aufbau, sowie die Durchführung der Simulationen erläutert. Eine folgende Darstellung und Diskussion der Ergebnisse bilden den Schwerpunkt des Kapitels.

Das letzte Kapitel 8 rundet die Arbeit durch einen Ausblick und eine Zusammenfassung ab.

1.5. Notationen

Dieser Abschnitt führt grundlegende graphentheoretische Notationen ein und beschreibt kurz die Besonderheiten in imperativen Elementen des verwendeten Pseudocodes. Für verteilte Algorithmen wird in Anhang A basierend auf sogenannten Timed I/O Automata ein deklarativer Spezifikations-Pseudocode für verteilte Systeme eingeführt, der allerdings ebenfalls die hier genannten imperativen Elemente nutzt.

1.5.1. Graphen

Straßen- und Kommunikationsnetze werden üblicherweise als Graphen modelliert. Dieses Kapitel führt kursorisch die nachfolgend benötigten Begriffe von Grund auf ein, da in der Literatur vielfältige, teilweise widersprüchliche, Definitionen existieren.

Sei $G = (V, E)$ ein gerichteter Graph mit V als Menge der Knoten und E als Menge der Kanten. Per Konvention werden in dieser Arbeit bei der Darstellung von gerichteten Graphen zu Gunsten der Lesbarkeit teilweise auf die Kantenrichtungen verzichtet. Stattdessen wird nur eine einzelne, bidirektionale Kante abgebildet. Auf Ausnahmen wird gesondert hingewiesen.

Der Knoten $w \in V$ heißt *inzidenter Nachfolger* von $v \in V$, wenn $(v, w) \in E$, *inzidenter Vorgänger*, wenn $(w, v) \in E$ und *inzidenter Nachbar*, wenn w inzidenter Nachfolger oder Vorgänger ist. Es wird auch die Notation $v \to w$, $w \leftarrow v$ und $v \leftrightarrow w$ verwendet, mit der Bedeutung, dass w inzidenter Nachfolger bzw. Vorgänger von v ist bzw. dass beide inzidente Nachbarn sind. Die Menge der *inzidenten Nachfolger* eines Knotens v ist gegeben durch $S_v = \{w | v \to w\}$, analog die Menge der *inzidenten Vorgänger* durch $P_v = \{w | v \leftarrow w\}$ und schließlich die Menge der *inzidenten Nachbarn* $N_v = S_v \cup P_v$. Auf den Zusatz „inzident“ wird verzichtet, sofern nicht zwingend zur Unterscheidung notwendig[1]. Der *Eingangsgrad* eines Knotens v wird mit $d_v^+ = |P_v|$ bezeichnet, der *Ausgangsgrad* mit $d_v^- = |S_v|$.

Ein *Pfad* $\pi_{v_0,v_n} = (v_0, \ldots, v_n)$ in G zwischen zwei Knoten v_0 und v_n ist eine endliche Folge von $n \in \mathbb{N}$ Nachfolgern $v_0 \to \ldots \to v_n$ mit $n > 0$. Die Anzahl der Nachfolger n wird als die Länge des Pfades bezeichnet. Existiert ein Pfad π zwischen v_0 und v_n, dann wird auch notiert $v_0 \overset{\pi}{\rightsquigarrow} v_n$. Existiert ein Pfad der Länge m zwischen v_0 und v_n, wird notiert $v_0 \overset{m}{\rightsquigarrow} v_n$. Damit ist $v \to w$ ein Pfad der Länge 1 zwischen v und w. Für einen Pfad $\pi = (v_0, \ldots, v_n)$ ist die Folge $(v_i, \ldots, v_j)$, $i < j$, $0 \leq i, j \leq n$ ein *Teil-* oder *Subpfad*.

Ein Pfad $\pi = (v_0, \ldots, n_n)$ heißt *Kreis*, wenn $v_0 = v_n$. Ein Pfad ohne Kreis als Teilpfad wird *kreisfrei* oder *einfacher Pfad* genannt. Ein Kreis $\pi = (v_0, \ldots, v_n)$ heißt *einfacher Kreis*, wenn $\nexists i, j \in \{1, n-1\}$: $v_i = v_j$, also wenn kein (weiterer) Kreis als Teilpfad enthalten ist. Häufig, so auch in dieser Arbeit, werden die beiden Begriffe Kreis und einfacher Kreis synonym verwendet, außer die Unterscheidung ist explizit notwendig. Der Begriff bezeichnet daher in

[1] In der Literatur bezeichnen Nachbarn mitunter alle im Graph erreichbaren Knoten, statt nur wie hier die inzidenten. Daher wird häufig eine Unterscheidung zwischen den Begriffen „inzidente Nachbarn“ und „Nachbarn“ eingeführt, wohingegen die Begriffe in dieser Arbeit synonym verwendet werden.

der Regel eine kreisfreie Folge von Knoten[2].

Die Menge der n-Nachfolgerumgebung eines Knotens v ist gegeben durch $S_v^n = \{w | v \overset{n}{\rightsquigarrow} w\}$, analog die n-Vorgängerumgebung durch $P_v^n = \{w | w \overset{n}{\rightsquigarrow} v\}$ und schließlich die n-Umgebung durch $N_v^n = S_v^n \cup D_v^n$.

Nun sei ein gewichteter Graph $G_s = (V, E, \omega)$ betrachtet. Die Gewichtungsfunktion $\omega\colon E \to \mathbb{N}$ weist jeder Kante ein *statisches Gewicht* zu. Die Kosten eines Pfades $\pi = (v_0, \ldots, v_n)$ in G_s sind dann gegeben durch

$$\delta(\pi) = \sum_{i=0}^{n-1} \omega\left((v_i, v_{i+1})\right). \tag{1.1}$$

Als günstigste Pfade zwischen zwei Knoten werden die Pfade bezeichnet, die die geringsten Kosten aufweisen. Für statisch gewichtete Graphen ist ein günstigster Pfad stets ein einfacher Pfad, da ein Kreis die Kosten erhöhen würde. Sei Π_{vw} die Menge aller Pfade in G zwischen v und w, dann ist ein einfacher Pfad $\check{\pi}_{vw}$ ein günstigster Pfad oder *Least Cost Path* (LCP) zwischen v und w, genau dann wenn:

$$\check{\pi}_{vw} \in \{\pi_{vw} | \operatorname*{arg\,min}_{\pi_{vw} \in \Pi_{vw}} \delta(\pi_{vw})\}. \tag{1.2}$$

Die Länge des kürzesten Pfades zwischen zwei Knoten wird als *Distanz* bezeichnet. Die maximale Distanz eines Knotens wird als *Exzentrizität* bezeichnet.

Im Kontext des Routings sind die Kantengewichte allerdings keineswegs stets statisch. Es sei daher der dynamisch-gewichtete Graph $G_d = (V, E, \omega)$ betrachtet, wobei $\omega\colon E \times \mathbb{N} \to \mathbb{N}$, eine Kante $e \in E$ und einen diskreten Zeitparameter $t \in \mathbb{N}$ auf das Gewicht $\omega(e, t) \in \mathbb{N}$ abbildet. Für einen fixierten Zeitpunkt t, werden die *dynamischen Kosten* eines Pfades in G_d analog zu dem statischen Fall bestimmt:

$$\delta(\pi, t) = \sum_{i=0}^{n-1} \omega((v_i, v_{i+1}), t). \tag{1.3}$$

Die dynamischen Gewichte repräsentieren beim Routing i.d.R. die Verweildauer auf der jeweiligen Kante. So lassen die Kosten eines Pfades zu einem fixen Zeitpunkt außer Acht, dass eine Kante (mit Ausnahme der Ersten) nicht zum Zeitpunkt t erreicht wird, sondern erst später. Die exakte Ankunftszeit hängt von den Kosten der vorherigen Kanten des Pfades ab. Ausgedrückt wird dies rekursiv durch die *dynamisch-konsekutiven Kosten*:

$$\kappa(\pi, t) = \sum_{i=0}^{n-1} \omega((v_i, v_{i+1}), t + \kappa((v_0, \ldots, v_i), t)), \tag{1.4}$$

mit $\kappa((v_0, v_0), t) = 0$. Der Zeitpunkt t repräsentiert dann die Startzeit des Fahrzeugs an Knoten v_0. Ein Pfad $\check{\pi}_{vw}$ ist LCP bzgl. dynamischer bzw. dynamisch-konsekutiver Kosten, genau dann wenn gilt:

$$\check{\pi}_{vw} \in \{\pi_{vw} | \operatorname*{arg\,min}_{\pi_{vw} \in \Pi_{vw}} \delta(\pi_{vw}, t)\} \tag{1.5}$$

[2]Diese Begrifflichkeiten werden in der Literatur höchst unterschiedlich verwendet. So wird auch Walk (hier: Pfad), Pfad (hier: einfacher Pfad), Zyklus (hier: Kreis) und Kreis (hier: einfacher Kreis) verwendet.

bzw.

$$\breve{\pi}_{vw} \in \{\pi_{vw} | \underset{\pi_{vw} \in \Pi_{vw}}{\arg\min}\, \kappa(\pi_{vw}, t)\}. \tag{1.6}$$

Es kann zusätzlich unterschieden werden, wann der LCP bestimmt wird: vorab oder nachträglich. Im ersten Fall werden aktuelle oder prognostizierte zukünftige Kantengewichte verwendet. Solch ein LCP sei als *Ex-Ante-LCP* zum Zeitpunkt t bezeichnet. Im zweiten Fall kann auf Aufzeichnungen der tatsächlich aufgetretenen Kantengewichte zurückgegriffen werden. Diese LCPs seien als *Ex-Post-LCPs* bezeichnet[3]. Routingprotokolle leiten Fahrzeuge (bzw. Datenpakete) anhand von Ex-Ante-Wissen weiter, in der Regel über Ex-Ante-LCPs. Welcher Pfad hingegen optimal gewesen wäre, ließe sich erst im Nachhinein bestimmen, nachdem exakt bekannt geworden ist, von welchem Knoten wann wie viele Fahrzeuge (bzw. Datenpakete) zu welchem Ziel aufgebrochen sind. Dieses Wissen ist in realistischen Situationen vorab jedoch nicht vorhanden, so dass ein Ex-Post-LCP nicht bestimmt werden kann.

Zusätzlich seien einige Eigenschaften von Graphen definiert. $G' = (V', E')$ ist ein *Subgraph* von G, wenn gilt $V' \subseteq V \wedge E' \subseteq E$. G heißt *Supergraph* von G'. Der Subgraph G' von G bzgl. der Knotenmenge $V' \subseteq V$ wird als *induzierter* Graph bezeichnet, notiert als $G[V']$, wenn gilt $E' = E \cap \left(\bigcup_{v,w \in V'} \{(v, w)\} \right)$, wenn also nur die Kanten zwischen den übernommen Knoten vorhanden sind.

Die Länge des längsten einfachen Pfades in G wird als *Umfang* von G bezeichnet. Die maximale Exzentrizität von G wird als *Durchmesser*, die Minimale als *Radius* bezeichnet. G heißt *stark zusammenhängend*, wenn gilt $\forall v, w \in V, v \neq w$: $v \rightsquigarrow w$. G enthält keine *Mehrfachkanten*, wenn gilt: $\nexists e = (v, w), e' = (v', w') \in E$: $v = v' \wedge w = w'$. G heißt *schleifenfrei*, wenn gilt: $\nexists (v, w) \in E$: $v = w$.

Im weiteren Verlauf werden Algorithmen zur Lösung zweier Probleme benötigt. Zum Einen müssen starke Zusammenhangskomponente, zum Anderen LCPs bestimmt werden. Beide Probleme lassen sich effizient lösen. Das erste Problem in $\mathcal{O}(|V| + |E|)$ vermöge des Tarjan-Algorithmus [138].

Das LCP-Problem kann in mehrere Problemvarianten unterteilt werden. Die Single-Source-Variante (ein Startknoten, alle übrigen Knoten sind Zielknoten) bzw. Single-Destination-Variante (ein Zielknoten, alle übrigen Knoten sind Startknoten) für positive Gewichte löst der Dijkstra-Algorithmus in $\mathcal{O}(|E| + |V| \log |V|)$ (nach [51], und in $\mathcal{O}(|V|^2)$ durch den ursprünglichen Ansatz in [35]). Der Dijkstra-Algorithmus ist der asymptotisch schnellste Algorithmus für dieses Problem. Treten negative Gewichte auf, können LCPs mithilfe des Bellmann-Ford-Algorithmus in $\mathcal{O}(|V||E|)$ gefunden werden. Häufig muss aber nur die Single-Pair-Variante (ein Start- und ein Zielknoten und positive Gewichte) gelöst werden. Der heuristische A^*-Algorithmus [59] kann dieses Problem in $\mathcal{O}(|V| \log |V|)$ lösen, sobald eine sogenannte *monotone* Heuristik eingesetzt wird (weiter unten wird für den Straßenverkehr eine solche angegeben). Die dritte hier erwähnte Variante sei die All-Pairs-Variante (zwischen allen Knoten). Der bekannteste Lösungsalgorithmus ist der Floyd-Warshall-Algorithmus [48] (keine negativen Kreise sind erlaubt) mit einer Laufzeit von $\mathcal{O}(|V|^3)$. Der zweite Algorithmus, Johnson's Algorithmus [71], weist eine Zeitkomplexität von $\mathcal{O}(|V|^2 \log V + |V||E|)$ auf, ist eine

[3] Lateinisch „ex ante“ bzw. „ex post“ für „aus vorher“ bzw. „aus danach“. Abgrenzend zum axiomatischen A-Priori-Wissen wird zum Ausdruck gebracht, dass die Konklusion auf Mutmaßungen bzw. unsicheren Annahmen basiert. Durch Erfahrung bzw. Beobachtung korrigiertes Ex-Ante-Wissen wird dann als Ex-Post-Wissen bezeichnet.

Kombination aus dem Bellmann-Ford- und Dijkstra-Algorithmus und wird in der Literatur insbesondere für spärliche Graphen empfohlen.

Damit sind die notwendigen graphtheoretischen Begriffe eingeführt. Im Folgenden werden Straßennetze als schleifenfreie, dynamisch-gewichtete Graphen ohne Mehrfachkanten modelliert. Kanten werden in diesen Zusammenhang üblicherweise als *Links* bezeichnet. Knoten repräsentieren Kreuzungen und Links dazwischenliegende Straßenabschnitte. Die verwendeten Routingprotokolle werden Ex-Ante-LCPs bestimmen und es werden (aufgrund der unrealistischen Voraussetzungen allerdings unpraktische) Verfahren zur Bestimmung von Ex-Post-LCPs vorgestellt.

1.5.2. Pseudocode

Im Wesentlichen wird üblicher imperativer Pseudocode zur Darstellung von Algorithmen verwendet. Zur Vermeidung von Uneindeutigkeiten seien an dieser Stelle die wichtigsten Elemente festgehalten:

1. Als Variablen werden Integer und die einfachen Datenstrukturen Liste, Map und Menge verwendet. Die leere Liste wird durch (), die leere Map durch [] und die leere Menge durch $\emptyset$ symbolisiert. Ist a eine einfache Datenstrutktur, bezeichnet $len(a)$ die Anzahl der Elemente.

2. Variablenzuweisungen werden mit $\leftarrow$ notiert, mittels $=$ erfolgt Prüfung auf Gleichheit.

3. Ist l eine Liste, bezeichnet $l[0]$ das erste Element und $l[x..y]$, die Teiliste vom x-ten bis zum y-ten Element und $l[x..] = l[x...len(l) - 1]$. Hinzufügen als letztes Element wird mit $\vdash$ notiert.

4. Ist m eine Map, ist $m[k]$ das zum Key k gehörende Element.

5. Für Mengen werden die Operatoren $(\cup, \cap, \setminus)$ und die Element-Relation $(\in, \notin)$ mit der üblichen Semantik verwendet. Hinzufügen eines Elementes wird durch $\vdash$ notiert. Auf Mengen wird zugegriffen wie auf eine Liste. Für eine Menge A bezeichnet so $A[0]$ das erste Element; es wird demnach, entgegen der rein mathematischen Definition, im Pseudocode eine stabile Ordnung der Menge während der Ausführung vorausgesetzt. Bspw. sei $S_i = \{a, b\}$ die Menge der Nachfolger von Knoten i. Dann gölte stets $S_i[0] = a$ und $S_i[1] = b$. Auf Mengen kann somit wie auf ein Array zugegriffen werden.

6. Aus den vorgenannten Variablentypen können zusammengesetzte Datenstrukturen entstehen, dabei wird objektorientierte Schreibweise verwendet. Bspw. sei G die zusammengesetzte Datenstruktur für einen Graph, bestehend aus zwei Mengen $G.V$ (Menge der Knoten) und $G.E$ (Menge der Kanten, bestehend aus 2-Tupeln, modelliert als zweielementige Liste).

7. Es werden Quantoren $(\forall, \exists)$ und die Junktoren $(\wedge, \vee, \neg)$ der Prädikatenlogik mit üblicher Bedeutung verwendet.

Dieser imperative Pseudocode wird in Anhang A um einen deklarativen Ansatz zur Spezifikation von verteilten Algorithmen ergänzt.

Kapitel 2

Verwandte Arbeiten

Das Thema und Anwendungsgebiet dieser Arbeit lässt sich dem Bereich der *Intelligent Transportation Systems* (ITS) zuordnen. Der weitläufige Begriff ITS, im Deutschen mitunter mit Verkehrstelematik übersetzt, bezeichnet die Anwendung von Informations- und Kommunikationstechnologien in der Anwendungsdomäne beliebiger Verkehrs- und Transportmodi. Der motorisierte Straßenverkehr steht dabei sicherlich im Forschungsfokus, jedoch ist auch der Flug-/Schiffs- und öffentliche Nahverkehr Gegenstand der Forschung. Im weiteren Verlauf der Arbeit sei im Hinblick auf die Zielsetzung derselbigen, die Einschränkung auf den Verkehrsmodus des motorisierten Straßenverkehrs vorgenommen. Hauptziele von ITS-Anwendungen im Straßenverkehr sind zum Einen die Verbesserung der Sicherheit und zum Anderen die Erhöhung der Effizienz. Wichtige Arbeitsfelder umfassen:

Erkennung: Ein wesentlicher Aspekt ist die automatisierte Erkennung von verkehrsrelevanten Zuständen und Umgebungen. Zum Einen aus der Sicht eines einzelnen Fahrzeugs (mikroskopisch), zum Anderen aus der Sicht des Gesamtsystems (makroskopisch). Mikroskopische Erkennung erfasst und erkennt mit am Fahrzeug angebrachten Sensoren die lokale Umgebung, wie Verkehrsschilder [19, 44, 96], anwendbare Vorfahrtsregelungen und die spurgenaue Fahrzeugposition [41, 151]. Makroskopisch gilt es großflächige Verkehrszustände [80, 83], wie bspw. Stau, zu erkennen.

Infrastruktur und Kommunikation: In ITS-Anwendungen stehen häufig einzelne Komponenten des Gesamtsystems zwecks Informationsaustausch in Kontakt. Hierzu wurden unterschiedliche infrastrukturelle Konzepte und Kommunikationstypen entwickelt. Unterschieden werden kann im Wesentlichen zwischen einer Peer-to-Peer-Kommunikation zwischen Fahrzeugen in der lokalen Umgebung, bezeichnet als Vehicle-to-Vehicle (V2V) [13], und einer Kommunikation via infrastruktureller Komponenten, bezeichnet als Vehicle-to-Infrastructure (V2I) [11]. Die infrastrukturellen Komponenten können dabei entweder am Straßenrand als reale Hardware installiert sein (sogenannte Road Side Units (RSU)) oder es findet eine Kommunikation zu im Internet erreichbaren Diensten statt (häufig ist dann von Vehicle-to-Cloud (V2C) [103] die Rede). Neben Fragen der Standardisierung wird dabei auch untersucht, wie die eigentliche (Funk-)Kommunikation zwischen den, zum Teil beweglichen, Komponenten (wie den Fahrzeugen) optimiert werden kann. Der technische Aspekt der Kommunikation zwischen Fahrzeugen wird häufig unter dem Begriff Vehicular Ad-hoc Networks (VANET) [60, 168] zusammengefasst. Dabei bezeichnet

V2V das prinzipielle Kommunikationsparadigma, ein VANET die technische, konkrete Umsetzung. Verwandt sind die Bereiche der allgemeineren Mobile Ad-hoc Networks (MANET [7]; statt Fahrzeugen stehen beliebige mobile Komponenten in Kontakt) und Wireless Sensor Networks (WSN [86]; einfache, häufig zum Stromsparen gezwungene, Sensoren bilden ein Funknetz).

Automatisierung: Aufbauend auf der Erkennung der lokalen Umgebung und der Kommunikation, nimmt die Automatisierung in diesem Bereich immer weiter zu. Paradebeispiele hierfür sind selbstfahrende, autonome Fahrzeuge [90], die ohne menschliche Interaktion auskommen. Ein bekanntes Beispiel ist das „Google Driverless Car"-Projekt, welches Mitte 2012 verkündete, dass eine Flotte von autonomen Fahrzeugen eine Gesamtdistanz von 500.000 Kilometern auf öffentlichen Straßen der USA unfallfrei zurückgelegt hat [56].

Sicherheit: Typische ITS-Anwendungen aus diesem Bereich detektieren, kommunizieren und warnen vor vorausliegenden Gefahren- und Unfallstellen, überwachen mittels geeigneter Sensoren die Umgebung, sowie entscheidende Aktionen und Reaktionen des Fahrers und greifen gegebenenfalls korrigierend ein.

Effizienz: Individual- sowie kommerzieller Güterverkehr im Straßenverkehr haben über die letzten Jahrzehnte kontinuierlich zugenommen und werden, wie einleitend in Kapitel 1 bereits bemerkt, dies voraussichtlich auch weiterhin tun. Diese massive Steigerung – mitunter bis hin zur Überlast – ist ein Hauptgrund der Effizienzminderung der jeweiligen Verkehrswege. Andere Gründe sind instand zu setzende Infrastruktur, Unfälle und Wetterbedingungen. Nach Angaben des U.S. Department of Transportation entstehen in den USA 40% der Staus durch schlichte Überlast, 25% durch Unfälle, 15% durch widrige Wetterumstände und 10% durch Baustellen [144]. Da Ansätze zur Erhöhung der Sicherheit die Unfallzahlen senken, existiert folglich auch eine Querverbindung zur Effizienzsteigerung. Auch die Automatisierung kann positive Auswirkungen auf die Unfallzahlen haben. Gänzlich gemein haben diese Ursachen, dass sie sich auf absehbare Zeit nicht vollständig vermeiden lassen. Umfängliche effizienzsteigernde Maßnahmen müssen demnach adaptiv genug sein, um die unterschiedlichen Stauursachen zu antizipieren. Das Prinzip der Effizienzsteigerung bezeichnet dann die optimierte Ausnutzung der gegebenen Infrastruktur zur Befriedigung des Verkehrsaufkommens. Das in dieser Arbeit vorgestellte Schwarmintelligenzverfahren fällt in diese Kategorie der Mechanismen zur Effizienzsteigerung.

Der folgende Abschnitt gibt eine Überblick über verwandte Arbeiten, die originär dem Staumanagement zur Effizienzsteigerung gewidmet sind.

2.1. Verwandte Maßnahmen zum Staumanagement

Unter dem Begriff *Staumanagement* wird im ITS-Bereich die effizienzsteigernde Maßnahme zur Verbesserung der Fahrzeiten der Verkehrsteilnehmer durch die Reduzierung negativer Auswirkungen von Überlast und Stau verstanden. Es können grundsätzlich zwei Kategorien unterschieden werden (siehe auch Abbildung 2.1):

Infrastruktur-basierte Maßnahmen: Hierbei wird die Infrastruktur in das Staumanagement eingebunden. Neben baulichen Maßnahmen kann darunter die an das Verkehrsgeschehen angepasste Steuerung von Ampeln, Zuflussregelungsanlagen (z.B. zur Regelung

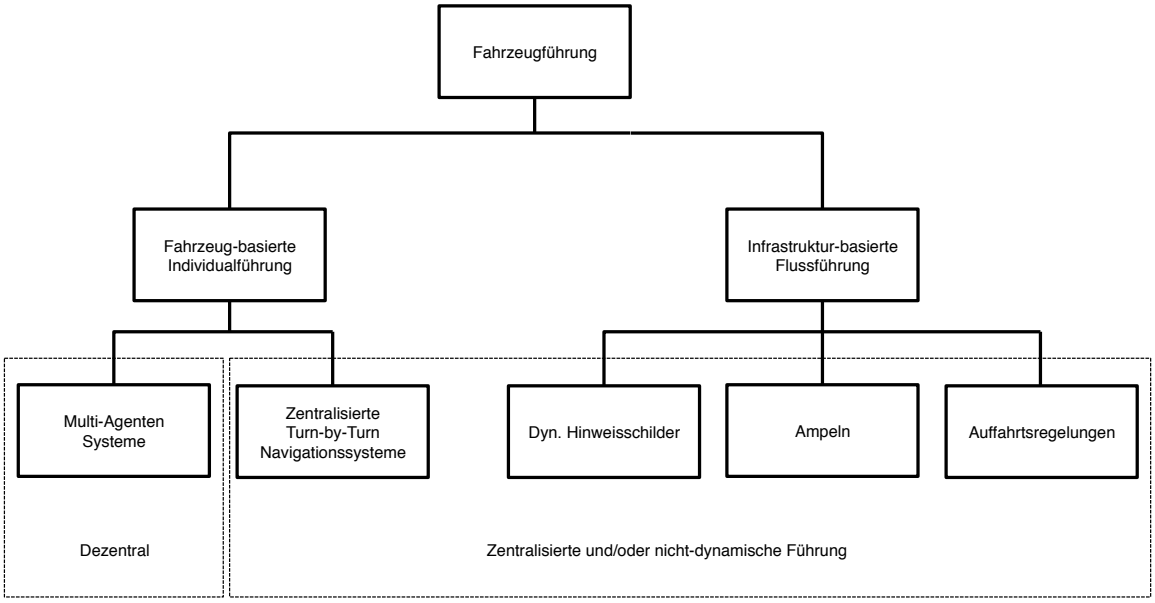


Abbildung 2.1.: Kategorisierung von Maßnahmen zur Fahrzeugführung

der Auffahrten auf Autobahnen) und Wechselverkehrszeichen (elektronisch steuerbare Verkehrszeichen) subsumiert werden.

Fahrzeug-basierte Maßnahmen: Hierbei werden Fahrzeuge mit Informationssystemen ausgestattet, um Fahrer mit aktuellen Verkehrsinformationen zu versorgen. So kann der Fahrer manuell oder das Navigationssystem (als On-Board Unit) automatisch diese Informationen zur Routenfindung verwenden. Der Beitrag dieser Arbeit fällt in diesen Bereich.

Die Grenzen der beiden Kategorien sind aber als fließend zu verstehen. Denn zum Einen existieren Infrastruktur-basierte Maßnahmen, die auf Kommunikation mit Fahrzeugen angewiesen sind, um relevante Zustände zu erkennen (und entsprechende Gegenmaßnahmen zu ergreifen). Zum Anderen benötigen Fahrzeug-basierte Maßnahmen nicht selten Kommunikationsinfrastruktur außerhalb des Fahrzeugs. Das in dieser Arbeit vorgestellte BeeJamA-Routingprotokoll bspw. kommuniziert mit regionalen Infrastrukturkomponenten, sogenannten Navigatoren, welche wiederum untereinander kommunizieren. Der wesentliche Unterschied liegt in der Individualität der Fahrzeug-basierten Maßnahmen, welche speziell auf ein einzelnes, diskretes Fahrzeug (bzw. einen Fahrer und dessen Fahrtziel) abgestimmt sind und daher in dem jeweiligen Fahrzeug eine individuelle Routenempfehlung darstellen können.

Infrastrukturelle Maßnahmen hingegen dienen der Lenkung von (nicht-individualisierten) Verkehrsströmen, weswegen eine Anzeige der Maßnahme (z.B. anhand von Ampeln und Verkehrszeichen) außerhalb des Fahrzeugs ausreichend ist.

Staus und Überlasten sind prinzipiell ein regionales Phänomen, es bieten sich daher dezentrale Lösungsstrategien an. Ein gängiges Entwicklungsparadigma in diesem Bereich sind daher Multi-Agentensysteme (MAS), in denen stationäre oder mobile intelligente Agenten autonom und

dezentral, d.h. ohne notwendige Interaktion mit allen übrigen Systementitäten, Entscheidungen treffen. Das BeeJamA-Routingprotokoll ist hierfür ein Beispiel: Mobile Agenten zur Verbreitung von Transitzeiten (benötigte Fahrzeit über einen Link) bewegen sich durch ein zum Straßennetz kongruentes logisches Kommunikationsnetz. Eine umfangreiche und aktuelle Übersichtsarbeit zu MAS für das Staumanagement ist etwa [32].

Im Folgenden seien einige relevante Maßnahmen beider Kategorien vorgestellt, häufig wird dabei das System explizit als MAS beschrieben.

2.1.1. Infrastruktur-basierte Maßnahmen

Die einfachste denkbare Infrastruktur-basierte Maßnahme wäre schlicht der Neubau von Straßen. Jedoch unterliegt das immer weiter steigenden Randbedingungen in Hinblick auf die zur Verfügung stehenden Flächen und den politisch notwendigen Konsens. Ferner ist das Braess-Paradoxon [17] zu beachten. Es besagt vereinfacht gesagt, dass unter unglücklichen Umständen der Neubau einer Straße zu einer Fahrzeitverlängerung für die beteiligten Verkehrsteilnehmer führen kann. Verkehrsplaner setzen daher simulative Verfahren ein, um solche Effekte bei Infrastrukturerweiterungen möglichst zu vermeiden und vorab die qualitative und quantitativen Effektivität einer zusätzlichen Straßen zu eruieren.

Es existieren aber auch eine Reihe von Agenten-Ansätzen zur Effizienzsteigerung des vorhandenen Straßennetzes:

- In [64] wird ein verteilter Schwarmintelligenzansatz zur Optimierung von Ampelschaltungen skizziert. Fahrzeuge werden als mobile Ameisen-Agenten modelliert, welche auf der intendierten Strecke den Ampeln mitteilen, wann das Agenten emittierende Fahrzeuge voraussichtlich eintrifft. Dadurch entsteht ein sogenanntes Swarm Voting-Konzept, bei dem den Ampeln Ankunftshäufigkeiten der Fahrzeuge bekannt sind. Ein zentraler evolutionärer Algorithmus optimiert sodann die Ampelschaltung mit dem Ziel möglichst geringe Wartezeit an den Ampeln entstehen zu lassen.

- In [4] wird ein Reinforcement Learning (RL) Algorithmus vorgestellt, in dem ein einzelner Ampelagent aus einem Netz von Ampeln über die Zeit Signalstrategien aus der Anzahl der Fahrzeuge vor diesen Ampeln ableitet. Im Vergleich zu einfachen dynamischen und statischen Signalstrategien verkürzt dieser Ansatz die Wartezeit.

- In [70] wird ebenfalls ein Reinforcement Learning-Ansatz vorgestellt, bei welchem statt innerstädtischen Ampeln Zufahrtsregulierungsanlagen (Ramp Meter) für Autobahnen und Wechselverkehrszeichen (Variable Message Sign, VMS) reguliert werden. Die genannten Entitäten werden wieder als RL-Agent modelliert, welche Verkehrsinformationen (z.B. Dichte und Geschwindigkeit) über lokale Sensoren sammeln und entsprechend die Zufahrt über die Ramp Meter regulieren bzw. die Geschwindigkeit und die Anzahl der Spuren pro Richtung über die VMS.

- In [120] wird ein Ansatz kommunizierender stationärer Ampelagenten beschrieben, welche aus vergangenen und aktuellen Verkehrsdaten lernen und zukünftige Verkehrszustände vorhersagen und zwar anhand simpler statischer Gleichungssysteme. (Dies ist dem konkreten Fall möglich, da ein leicht vorherzusagenes Verkehrsmuster verwendet wird. In der Realität müssten solche Annahmen eliminiert und bspw. durch Simulationen zur Vorhersage ersetzt werden).

In der genannten Übersichtsarbeit [32] und dem umfangreichen Referenzwerk der Arbeit [111] finden sich viele weitere Arbeiten zum Thema Steuerung von Infrastrukturkomponenten zum Staumanagement. Das Grundprinzip ist jedoch im Wesentlichen stets dasselbe: Mit der zu steuernden Infrastrukturkomponente wird ein Agent assoziiert, welche über ein Kommunikationsnetz mit den anderen Agenten kommunizieren können. Jeder einzelne Agent sammelt mittels lokalen Sensoren relevante Verkehrsdaten und tauscht diese ggf. mit den restlichen Agenten aus. Je nach Konzept werden diese Daten dann dezentral verwendet (z.B. für Lern-Algorithmen), um die jeweilige Steuerungsaufgabe zu erfüllen oder aber zentral gesammelt und eine Optimierung anhand von globalen Daten durchgeführt. Insbesondere das Ramp Metering ist aber nicht unumstritten [5, 21], da häufig Rückstaus an den Auffahrten zu beobachten sind, welche weitere Behinderungen im rückwärtigen Raum auslösen können. Diese Effekte werden häufig bei der Untersuchung der Effektivität dieser Methoden außer Acht gelassen.

2.1.2. Fahrzeug-basierte Maßnahmen

Mit dem Aufkommen von Navigationssystemen (*Personal Navigation Assistants*, PNAs) in Fahrzeugen ergab sich die Möglichkeit, Fahrzeuge individuell, d.h. basierend auf dem Ziel des Fahrers, zu leiten. Nachteilig bei klassischen PNA-Lösungen ist die zugrundeliegende statische Gewichtung der Links (Kanten) des Straßennetzes. Offenkundig ist die Auslastung des Straßennetzes jedoch einer hohen Dynamik unterworfen. Aktuellere VRGS, bestehend aus einem PNA als On-Board Unit und zusätzlichen Systemkomponenten außerhalb des Fahrzeugs zur Sammlung kontinuierlich aktualisierter Link-Daten wie Auslastung, aktuelle Fahrtzeiten und Wetterkonditionen, beziehen daher auch solche dynamischen Informationen in die Routenfindung mit ein. Bisherige kommerziell erhältliche VRGS arbeiten dabei allerdings zentral, da konzeptbedingt zunächst in einem *Traffic Information Center* (TIC) alle relevanten Informationen gesammelt werden. Anschließend ergeben sich zwei Möglichkeiten:

1. Die akkumulierten Daten werden an die Fahrzeuge übermittelt. Dazu kann Rundfunk eingesetzt werden (wie bei TMC), wodurch aber alle Fahrzeuge dieselben Informationen erhalten. Zwar steigt der Kommunikationsaufwand (die sogenannte Nachrichtenkomplexität) nicht mit der Anzahl der Fahrzeuge, dafür muss allerdings das gesamte Straßennetz an alle PNAs übermittelt werden, da keinerlei Unterscheidung zwischen den Fahrzeugen vorgenommen werden kann.

2. Alternativ kann das PNA eine Anfrage mit Ziel des Fahrzeugs, eine sog. *Routinganfrage*, an das TIC senden. Hierzu werden typischerweise bidirektionale Internetverbindungen genutzt. Beispiele hierfür sind TomTom HD Traffic [140] und Waze [153]. Das TIC hat sodann zwei Optionen. Zum Einen kann es selbst eine (Teil-)Route berechnen und nur die Antwort zurücksenden. So wird die Nachrichtenkomplexität zuungunsten des Berechnungsaufwandes im TIC reduziert. Zum Anderen kann das TIC dem anfragenden PNA Link-Aktualisierungen für einen heuristisch bestimmten Ausschnitt des gesamten Straßennetzes zurücksenden und das PNA berechnet lokal anhand dieser aktualisierten Daten die Route neu. Es steigt die Nachrichtenkomplexität, dafür wird die Routenberechnung vom TIC auf die peripheren PNA verschoben.

In beiden Fällen ergibt sich ein Flaschenhals im TIC, prinzipbedingt ist die Skalierbarkeit dort begrenzt.

Dezentrale VRGS hingegen, welche potentiell den dräuenden Flaschenhals zumindest abmildern können, sind bisher nur vereinzelt erforscht worden, in praktischer Anwendung befindet sich davon bisher keines. Neben dem hier beschriebenen BeeJamA-Routingprotokoll sind drei Verfahren in der Literatur bekannt, die hier zunächst kursorisch aufgezählt seien. Im Anschluss der Darstellung des BeeJamA-Protokolls werden die Unterschiede noch einmal genauer diskutiert. Die drei Verfahren im Einzelnen:

1. In [139] wird das verteilte Schwarmintelligenzverfahren H-ABC (für Hierarical Ant Based Control), basierend auf dem natürlichen Vorbild der Ameisen, vorgestellt, dass ursprünglich auf einem ähnlichen Ansatz für Computernetze basiert. Das Straßennetz wird in sogenannte Sektoren unterteilt. (Das BeeJamA-Protokoll führt eine ähnliche regionale Unterteilung durch.) Innerhalb dieser Sektoren werden, vergleichbar zu dem AntNet-Routingprotokoll [34] für Computernetze, sogenannte, von den einzelnen Netzknoten startende, zielgerichtete Forward- und Backward-Ants eingesetzt, um Routingtabellen laufend zu aktualiseren. Zwischen den Sektoren werden sogenannte Exploration-Ants eingesetzt, um die Qualität von Verbindungen zwischen den Sektoren zu erkunden. Insgesamt ermitteln die Ants den günstigsten Pfad zwischen zwei Sektoren und zwischen Knotenpaaren innerhalb der Sektoren. Fahrzeuge werden sodann über diese Pfade zum Ziel geleitet. Allerdings wird nur ein einziger Pfad, nämlich der Günstigste, zwischen zwei Sektoren ermittelt. Ein Sektor repräsentiert dabei eine ganze Stadt, d.h. unabhängig von der Position innerhalb dieser Stadt des zu routenden Fahrzeugs, werden alle Fahrzeuge über die gleiche Verbindung zum Ziel geleitet. Ferner können Fahrzeuge den Zielsektor nicht verlassen (falls sich folglich vor dem Ziel Staus ergeben, können diese nicht mehr weiträumig umfahren werden) und das System kann nur langsam auf Veränderungen im Verkehrsgeschehen reagieren.

2. Ein weiterer (namenloser) Ameisen-basierter Ansatz ist in [24, 163] beschrieben, der im Folgenden als D-MAS bezeichnet sei (da die Autoren sich auf ein Konzept namens Delegated MAS beziehen). Hierbei starten die Agenten nicht kontinuierlich von allen Kreuzungen, sondern von der aktuellen Position des Fahrzeugs aus. Die Agenten explorieren mittels Flooding das gesamte Netz bis das individuelle Ziel des Fahrzeugs gefunden wurde. Dieser Vorgang wird bei Bedarf, bspw. an jeder Kreuzung, wiederholt. Auf ihrem Weg sammeln die Ants Fahrzeiten, welche durch die antizipierte Anzahl von Fahrzeugen bei der erwarteten Ankunft des Fahrzeugs approximiert werden. Dazu wird ein Reservierungsmechanismus eingesetzt, nach dem Fahrzeuge sich bei Straßenabschnitten weit im Voraus anmelden müssen, bevor sie diese passieren. Dadurch kann abgeschätzt werden, wie viele Fahrzeuge zu einem bestimmten Zeitpunkt sich auf einem Straßenabschnitt befinden und daraus eine Fahrzeit geschätzt werden. Der Ansatz arbeitet völlig ohne Hierarchie, Agenten durchlaufen daher stets das gesamte Netz, weswegen von einem großen Overhead auszugehen ist.

3. In [23] wird ein Routingprotokoll für Wireless Sensor Networks (WSN) auf die VRGS-Domäne übertragen. Jede Kreuzung stellt einen Knoten in dem WSN dar. Fahrzeuge senden eine Routinganfrage zu jedem direkt vorausliegenden WSN-Kreuzungsknoten. Von dort aus wird das gesamte WSN mittels Flooding nach dem Zielknoten abgesucht. Die eingesetzten Agenten kumulieren auf ihrem Weg die Fahrzeiten. Erreichen Agenten das angestrebte Ziel, werden sie auf gleichem Weg wieder zurückgeschickt. Das Fahrzeug hat dann verschiedene Pfade zur Auswahl und kann bspw. den mit der kürzesten Fahrtzeit

wählen. Technisch ist das Verfahren sehr ähnlich zu dem zuvor beschriebenen, mit dem Unterschied, dass keine Reservierungen durchgeführt werden.

Insgesamt gilt, dass die drei genannten Verfahren nur sehr oberflächlich beschrieben und unzureichend evaluiert worden sind. Details der Verfahren wurden nicht veröffentlicht, die Beschreibungen bleiben an vielen Stellen sehr vage und Implementierungen sind nicht verfügbar. Das erstgenannte Verfahren wurde vor, die beiden anderen nach BeeJamA erstmalig veröffentlicht. Um die Nachteile der Verfahren und Unterschiede, insbesondere zu BeeJamA, exakt herausarbeiten zu können, müssen zunächst notwendige Grundlagen und das BeeJamA-Protokoll an sich eingeführt werden. Abschnitt 6.13 geht daher noch einmal und vor allem detaillierter auf diesen Vergleichsaspekt zwischen den Protokollen ein.

2.2. Weitere Themengebiete dieser Arbeit

Neben dem Staumanagement und der Fahrzeugführung sind auch weitere Themengebiete von Bedeutung, entweder in direktem Bezug zum Inhalt dieser Arbeit oder zur besseren Einordnung des vorgestellten Verfahrens und dessen Ergebnisse. Eine detaillierte Einführung erfolgt in dem jeweils das Thema behandelnde Kapitel, hier sei der Übersicht halber eine Zusammenfassung dargetan:

Verteilte Systeme und Algorithmen: Das vorgestellte VRGS stellt ein verteiltes System dar, der zugrundeliegende Routingansatz ist ein verteilter Algorithmus. Schon dadurch wird die fundamentale Bedeutung dieser Begriffe für den Kontext dieser Arbeit deutlich. Ein verteiltes System besteht aus zur Kooperation gezwungener, nebenläufiger Komponenten mit jeweils lokalen, eventuell untereinander unterschiedlichen oder gar widersprüchlichen Weltsichten. Die Erforschung verteilter Systeme und Algorithmen begann im Kontext von Computernetzen. Das ARPANET [46], eingeführt Ende der 1960er, stellte einen ersten großen Meilenstein dar. Auch theoretische Betrachtungen im Umfeld von Betriebssystemen, bspw. zu dem Philosophenproblem [63] zur verteilten Ressourcenallokation, stellten befruchtende Beiträge dar. Die besondere Bedeutung für diese Arbeit erhalten diese beiden Themen durch die grundlegenden Beiträge für die folgenden. Routingverfahren, Multi-Agenten Systeme und Wireless Sensor Networks stellen jeweils verteilte Systeme oder Algorithmen dar. Eine einführende Monographie zu diesem Thema findet sich in [91].

Routing in Computernetzen: In größeren Computernetzen stehen typischerweise mehrere Pfade zwischen zwei Knoten zur Verfügung. Die Frage, welche Pfade zum jeweiligen Zeitpunkt zur effizienten Kommunikation verwendet werden sollen, kam dabei schon früh auf. Da Computernetze verteilte Systeme darstellen, sind die zur Lösung dieser Frage verwendeten Verfahren natürlicherweise verteilte Algorithmen, die zur Laufzeit kontinuierlich neue Pfade explorieren. Die ersten Verfahren stellten mehr oder minder statische Ansätze dar, die in größeren Netzen an Komplexitätsgrenzen [94] stießen. Die Forschung konzentrierte sich darauf, diese Mängel auszuräumen und gleichzeitig die Effizienz zu steigern im Sinne von erhöhtem Durchsatz und verringerter Varianz. Es entstanden die klassischen Routingalgorithmen [61, 98, 115], wie sie Dekaden im Internet eingesetzt wurden und werden. Später wurden auf der Suche nach Ansätzen mit höherer Dynamik Verfahren auf Basis von Multi-Agenten Systemen und Schwarmintelligenzprinzipien vorgeschlagen [34, 156].

Multi-Agenten Systeme und Schwarmintelligenz: In den 1990er Jahren begann die starke Ausbreitung der Multi-Agent Systems (MAS, vgl. [167]). Grundlegende Idee ist, Systeme durch kommunizierende Komponenten mit begrenzter Weltsicht zu beschreiben, deren zielgerichtete Interaktion durch möglichst einfache Regeln determiniert sind, aber gleichzeitig flexibel genug, um auf dynamische Änderungen zu reagieren. Schwarmintelligenz [15], oder auch kollektive Intelligenz, wenngleich ein umstrittener Begriff, bezeichnet häufig die Eigenschaft eines MAS, bestehend aus sehr großen Anzahl von Agenten und einer sehr reduzierten Menge von Kommunikations- und Interaktionsregeln, iterativ zu einem gewünschten Zielzustand, häufig ein Gleichgewicht, zu gelangen. Die Abgrenzung der Begrifflichkeiten ist nicht immer leicht und die Grenzen sind fließend (dieser Punkt wird in Abschnitt 5.6 noch einmal aufgegriffen). Wesentliche Entwicklungen auf diesem Gebiet, stellten die ersten Veröffentlichungen zu dem Thema der sog. Ameisen-Algorithmen [37, 39] dar. Die nächste Dekade der Forschung in diesem Bereich wurde sicherlich stark durch diese ersten Ansätze beeinflusst. So entstanden auch Ameisen-basierte Routingalgorithmen für Computer- [34] und, wie im vorherigen Abschnitt dargestellt, Verkehrsnetze [23, 139, 163].

Transportoptimierung: Schon früh kam die Frage auf, wie alltägliche Probleme der Transportlogistik mithilfe von Optimierungsverfahren angegangen werden können. Dominierend dabei waren kombinatorische Fragestellungen, die durch zentrale Methoden gelöst wurden [10, 20, 107]. Stellvertretend sei das Vehicle Routing Problem (VRP) [143] genannt. Auch wenn der Name es vermuten lassen könnte, unterscheidet sich dieses Problem von dem dieser Arbeit deutlich. Das VRP dreht sich um die Fragestellung, wie Routen von Transportfahrzeugen von zwischen Depots (Lager von Gütern) und Kunden optimiert werden können – in Hinblick auf die Anzahl der notwendigen Fahrzeuge, der benötigten Zeit für die Gesamtheit aller Lieferungen, der verstrichenen Zeit zwischen Bestellung und Lieferung u.v.m. Die Minimierung der Fahrzeiten aller Fahrzeuge, durch Reduzierung der Nachteile von Stau hingegen spielt i.d.R. keine Rolle. Stattdessen wird die vorhandene Verkehrsbelastung als gegeben hingenommen. Zentrale Ansätze der Optimierung wurden aber auch häufig auf das originäre Thema dieser Arbeit, die Fahrzeitreduzierung, übertragen [52, 73, 105].

Verkehrsplanung und -simulation: Ebenfalls früh wurden Fragen diskutiert, wie Verkehrssysteme gestaltet sein müssen, um die antizipierte Nachfrage bedienen zu können. Fragestellungen umfassten auch, wie Modelle beschaffen sein müssen, um realen Straßenverkehr zu beschreiben oder gar nachzubilden. Dies stellt eine notwendige Bedingung für die Verkehrssimulation dar, die in Abschnitt 3.3.2 noch ausführlich dargestellt wird.

Der nächste Abschnitt geht auf die eigenen Vorarbeiten im Themengebiet dieser Arbeit ein.

2.3. Eigene Vorarbeiten

Am Lehrstuhl 3 der Fakultät Informatik der TU Dortmund wird etwa seit dem Jahr 2004 Forschung zu schwarmbasierten Routingprotokollen betrieben. Erstes Ergebnis war das BeeHive-Protokoll [45, 156] zum Routing in Computernetzen. Das Protokoll adaptierte das dynamische Verhalten von Bienenschwärmen bei der Futtersuche und erzielte bessere Resultate in Hinsicht auf höheren Datendurchsatz und eine wesentlich bessere Skalierbarkeit als klassische

Protokolle wie OSPF [98] und andere Schwarmintelligenzansätze wie AntNet [34]. Anschließend wurde mit BeeAdHoc [155] eine Erweiterungen für MANETs vorgeschlagen und mit BeeSensor [121] eine speziell angepasste Variante für WSN vorgelegt. Mit dieser Arbeit wird das Konzept auf die VRGS-Domäne übertragen. Das resultierende Protokoll wurde *BeeJamA* (für Bee Jam Avoidance) genannt. Ursprünglich wurden die ersten Vorarbeiten im Rahmen von zwei studentischen Projektgruppen erstellt und die ersten vorläufigen Ergebnisse veröffentlicht [154, 159, 160]. In folgenden Veröffentlichungen wurde das Konzept kontinuierlich weiterentwickelt [126, 128, 129, 157, 158, 162].

In die Ergebnisse dieser Arbeit flossen Ergebnisse aus am Lehrstuhl erstellten Abschlussarbeiten ein: Anbindung des Aimsun-Simulators [65], Anbindung des SUMO-Simulators [72], Anbindung des MATSim-Simulators [77], verteilte Implementierung des BeeJamA-Protokolls [166] und Implementierung des Generischen Routing Frameworks [88, 113].

Das Protokoll in der heutigen Form wurde in den folgenden Journal- und Konferenzbeiträgen entwickelt und dargelegt:

- Horst Wedde und Sebastian Senge, *BeeJamA: A Distributed, Self-Adaptive Vehicle Routing Guidance Approach*, IEEE Transactions on Intelligent Transportation Systems, Dezember 2013 [161]

- Sebastian Senge und Horst Wedde, *Marginal Cost Pricing and Multi-Criteria Routing in a Distributed Swarm-Intelligence Approach for Online Vehicle Guidance*. Proceedings of the IEEE Intelligent Transportation Systems Conference 2013, Den Haag, Niederlande, Oktober 2013 [130]

- Sebastian Senge, *Assessment of path reservation in distributed real-time vehicle guidance*. Proceedings of the IEEE Intelligent Vehicles Symposium 2013, Gold Coast, Australien, Juni 2013 [125]

- Horst F. Wedde und Sebastian Senge, *2-Way Evaluation of the Distributed BeeJamA Vehicle Routing Approach*. Proceedings of the IEEE Intelligent Vehicles Symposium 2012, Alcalá de Henares, Spanien, Juni 2012 [127]

Kapitel 3

Ein Generisches Routing Framework

Die Erprobung der Wirksamkeit neuartiger VRGS vor der Markteinführung erfordert eine simulative Überprüfung, möglichst unter realitätsnahen Bedingungen. Wesentliches Werkzeug hierzu sind Verkehrssimulatoren (im Folgenden kurz als *Simulatoren* bezeichnet). Ursprünglich entwickelt wurden solche Simulatoren zu verkehrsplanerischen Zwecken, als Werkzeug bei der Fahrzeugentwicklung (z.B. als Ersatz für Crashtests) und zur Erforschung von Phänomenen des Straßenverkehrs, z.B. der Staubildung. Simulative VRGS-Erprobung ist demgegenüber ein junges Anwendungsgebiet und bringt neue Anforderung mit sich: Fahrzeuge müssen online, d.h. zur Laufzeit der Simulation, gesteuert werden können und zwar in dem Sinne, dass jedem einzelnen Fahrzeug individuell vorgeschrieben werden kann, ob und wie abgebogen werden soll. Die Benachrichtigung jedes einzelnen Fahrzeugs muss dabei rechtzeitig vor jeder Kreuzung geschehen, damit sich das Fahrzeug ggf. noch entsprechend einordnen kann.

Die bisherigen Anforderungen an Verkehrssimulatoren waren diesbezüglich geringer. Oft reichte es aus, aus einer Datei einen vorgegebenen, statischen Pfad pro Fahrzeug (offline) auszulesen und das Fahrzeug danach, während der Simulation, dementsprechend zu leiten. Die Programmierschnittstellen zur Online-Fahrzeug-Führung im Sinne eines Routings sind daher bei Simulatoren, falls überhaupt, nur begrenzt und teilweise fehlerhaft vorhanden. Dies ist einer der Gründe, warum Veröffentlichungen in diesem Bereich häufig unterschiedliche, oder gar selbstentwickelte, nicht veröffentlichte Simulatoren einsetzen. Eine Vergleichbarkeit der Ergebnisse untereinander wird dadurch erheblich erschwert, da die Simulatoreigenschaften, wie noch gezeigt wird, deutlichen Einfluss auf die Fahrzeiten der Fahrzeuge haben.

Bei den verfügbaren Simulatoren sind die Programmierschnittstellen (falls vorhanden) aus unterschiedlichen Programmiersprachen und mittels verschiedener Paradigma ansprechbar, was eine (mehrfache) Portierung eines bereits entworfenen Routingprotokolls notwendig macht. Eine Vereinheitlichung der Schnittstellen ist daher ein sinnvolles Unterfangen. Wenngleich diese Arbeit das Problem nicht grundsätzlich beseitigen kann, so wird mit dem in diesem Kapitel beschriebenen *Generischen Routing Framework* (GRF) zumindest ein Beitrag geleistet, die Vergleichbarkeit zukünftiger Entwicklungen zu vereinfachen. Bei dem GRF handelt es sich um eine Middleware, die Routingprotokolle und Simulatoren voneinander trennt und so ermöglicht, ein Protokoll ohne Änderungen mit unterschiedlichen Simulatoren zu evaluieren. Der Vorteil aus Sicht des Routingprotokolls (bzw. dessen Entwickler) ist, dass trotz Änderung

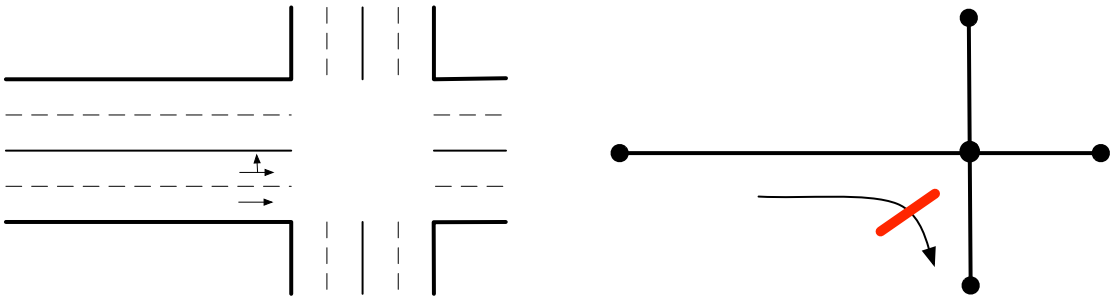


Abbildung 3.1.: Straßen als Graphen

des Simulators eine gleichbleibende Umgebung vorhanden ist und anfallende Sekundäraufgaben wie Logging und Auswertung zentralisiert werden können.

Im nachfolgenden Abschnitt 3.1 folgt zunächst eine Beschreibung der relevanten Eingaben des GRF: Straßennetz und sogenannte Verkehrspläne zur Modellierung der *Verkehrsnachfrage*. Anschließend wird in Abschnitt 3.2 die Architektur und Funktionalität des GRF erläutert. Es folgt eine kursorische Einführung in die Verkehrsdynamik und -simulation, sowie eine Übersicht der bisher unterstützten Verkehrssimulatoren in Abschnitt 3.3 und der Routingprotokolle in Abschnitt 3.4. Abschließend wird anhand einiger Ergebnisse von mit dem GRF durchgeführten Simulationen in Abschnitt 3.5 zum Einen aufgezeigt, welche deutlichen Auswirkungen unterschiedliche Simulatoren haben und zum Anderen, dass für die spätere Evaluation des BeeJamA-Protokolls, von den bisher eingebundenen Simulatoren, nur einer in Frage kommt.

3.1. GRF-Eingaben

Das GRF benötigt zweierlei Eingaben:

Straßennetz: Diese können in Form von Graphenbeschreibungen $G = (V, L)$ bspw. von dem OpenStreetMap-Projekt [104] bezogen werden.

Verkehrspläne: Der Verkehrsplan (kurz: Plan) eines Fahrzeugs α ist ein 3-Tupel (s, d, t), mit $s \in V$ als Start-, $d \in V$ als Zielknoten und t als Startzeitpunkt. Das Tupel (s, d) wird auch als *Origin-Destination*-, oder kurz, OD-Paar bezeichnet. Die in dieser Arbeit zur Modellierung einer Verkehrsnachfrage verwendeten Verkehrspläne, werden allesamt synthetisch, mit zufällig gleichverteilten $s, d \in V$ und t aus einem gegebenen Zeitintervall (welches die vorab festgelegte Simulationsdauer umfasst), erzeugt.

Prinzipiell wären realistischere Verkehrspläne wünschenswert, jedoch ist deren Verfügbarkeit nur bedingt gegeben. Zwar besteht die Möglichkeit, behördlich erhobene Auslastungsdaten zu erhalten, diese enthalten aber ausschließlich Überfahrtsmessungen, bspw. von Induktionsschleifen. Es existieren zwar Verfahren (siehe z.B. [109] für eine Übersicht) zur approximativen Rekonstruktion von Start-Ziel-Zusammenhängen (OD-Matrizen) aus Beobachtungen, jedoch bedarf es hierbei meistens weiterer sozioökonomischer Informationen. So müsste z.B. bekannt sein, wo Wohngebiete ausgewiesen sind, da dadurch die Wahrscheinlichkeit steigt, dass dort morgens ein Fahrzeug aufbricht (z.B. in Richtung eines Gewerbegebiets) und abends zurückkehrt. Da solche Information nur selten mit den verfügbaren Straßennetzen verknüpft sind, ist die Verwendung rein synthetischer Pläne eine übliche Vorgehensweise im ITS-Umfeld.

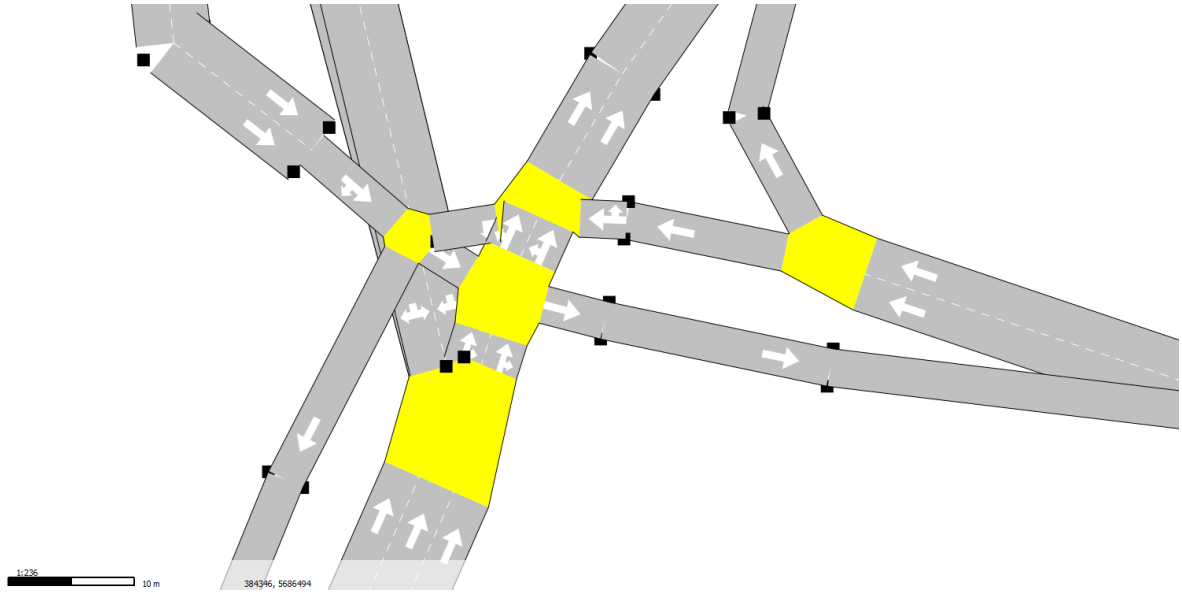


Abbildung 3.2.: Komplexe Kreuzungen

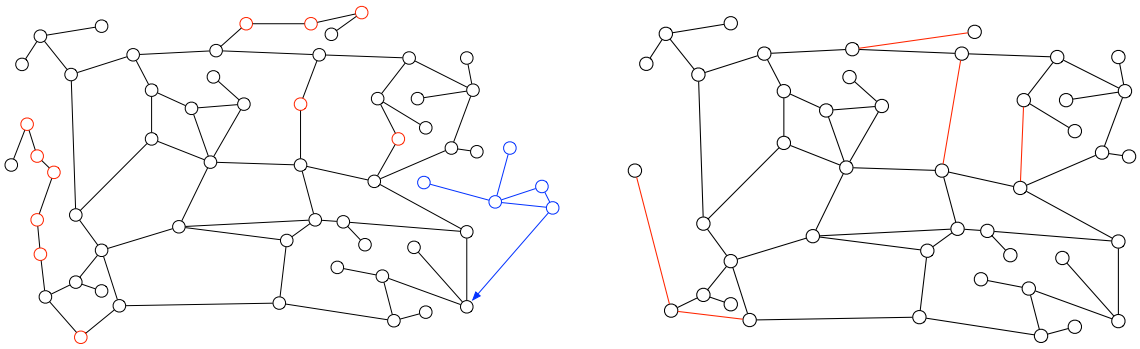
Die später vorgestellten Verkehrssimulatoren unterscheiden sich in ihrem simulierten Detailgrad. Dies hat Auswirkungen für die Repräsentation des Straßennetzes. Ein wesentliches Unterscheidungsmerkmal der Simulatoren besteht nämlich in der Berücksichtigung von Spuren und dem zugehörigen Abbiegeverhalten. Beispielhaft zeigt Abbildung 3.1 eine bidirektionale Kreuzung, mit jeweils zwei Spuren pro Richtung. Dazu wird ein korrespondierender Graph dargestellt[1]. Als zusätzliche Information ist das Abbiegeverbot der von links nach rechts führenden Spur annotiert. Nicht jeder Simulator jedoch nutzt diese Information, weswegen bei jenen ein Abbiegen nach rechts (von links kommend) unrealistischerweise möglich wäre.

Zudem stellt die Abbildung 3.1 ein gewisses Idealbild einer Kreuzung dar. In den verfügbaren Daten bestehen einzelne Kreuzungen aus einer Vielzahl von Knoten, wodurch sich häufig recht komplexe Gebilde ergeben, deren Zusammenhang mit dem infrastrukturellen Gegenbild nicht unmittelbar erkennbar ist. Abbildung 3.2 und Abbildung 3.3 (links) zeigen Beispiele hierfür. Im ersten Beispiel sind überlappende Links und Aufspaltungen einer Kreuzung in mehrere Knoten zu erkennen. Im zweiten Beispiel besteht die Kreuzung z.B. aus drei Knoten mit mehreren (sehr kurzen) Links zwischen diesen. Häufig entstehen so Knoten, die nur unidirektional verbunden sind, obwohl die Gesamtkreuzung bidirektional ist. Es existieren Tools, um solche Knoten zu vereinen (Abbildung 3.3 (rechts)), dies funktioniert aber nur unter sehr speziellen Voraussetzungen, so dass keine zusätzlichen Abbiegemöglichkeiten ermöglicht werden oder gänzliche neue Wege geschaffen werden.

Relevanter im Kontext des Routings sind jedoch die vielen sogenannten Intermediate-Knoten (kurz: Intermediates), die rein zur geometrischen Formgebung, bspw. von Kurven, vorhanden sind. Da die Komplexität von Routingprotokollen von der Anzahl Knoten und Kanten abhängt, ist es sinnvoll, diese Intermediates zu eliminieren. Allerdings nur aus Sicht der Routingprotokolle, die Simulatoren hingegen benötigen weiterhin den vollständigen Graphen: Erstens zur Visualisierung des vollständigen Graphen, zweitens, da durch das Löschen von

[1] Wie in Abschnitt 1.5.1 erwähnt, repräsentieren, wenn nicht anders festgehalten, ungerichtet dargestellte Links einen bidirektionalen Link.

Abbildung 3.3.: Vereinfachung von Graphen



(a) (b)

Abbildung 3.4.: Ursprüngliches Straßennetz und Routinggraph

Knoten Links zusammengefasst würden, die ggf. unterschiedliche Parameter aufweisen können. Würde bspw. das arithmetische Mittel der Anzahl der Spuren gebildet, können sich unterschiedliche Simulationsergebnisse im Vergleich zu dem unveränderten Zustand ergeben. Da davon Abstand genommen werden soll, erzeugt das GRF aus dem, als physikalischen Graphen bezeichneten, Eingabe-Straßennetz den sogenannten *Routinggraph*. Dieser besteht nur aus den Knoten, die Kreuzungen repräsentieren und den verbindenden Links. Abbildung 3.4a zeigt einen physikalischen Graphen.

Der Routinggraph wird wie folgt erzeugt (vgl. Algorithmus 1). Zunächst werden die starken Zusammenhangskomponenten bestimmt und nur die größte weiter betrachtet. Hintergrund ist, dass an den Rändern der, aus einem größeren Straßennetzkontext ausgeschnittenen, Straßennetze mitunter einige Knoten vorhanden sind, die vom „Hauptnetz“ nicht erreicht werden können. In dieser verbleibenden Hauptkomponente werden anschließend die Knoten identifiziert, die keine Kreuzung darstellen (rot dargestellt) und die Links entsprechend zusammengefasst. Abbildung 3.5a zeigt die Fälle in denen eine Zusammenfassung vorgenommen wird. Von den dort abgebildeten vier Möglichkeiten, können in zwei Fällen Ersetzungen durchgeführt werden, ohne das neue Fahrtrichtungen eingeführt werden. Die zweite Abbildung 3.5b zeigt den Fall, in welchem statt einem Intermediate, mehrere vorhanden sind. Ausgehend von einem beliebigen dieser Knoten, sucht der Algorithmus einmal in Vorwärts- und einmal in Rückwärtsrichtung bis zu einem Knoten, der mehrere Abbiegemöglichkeiten aufweist und ersetzt die Intermediates entsprechend. Die Abbildung 3.4b zeigt den resultierenden Routinggraph zu dem physikalischen Graph aus Abbildung 3.4a.

Automaton 1 Konvertierung des Graphen

```
1: function prepareGraph(G)
2:     G_R ← empty graph
3:     intermediates← ∅
4:     largest_component ← select largest component of tarjan(G)
5:     for each node n ∈ largest_component do
6:         if |N_n| = 2 ∨ d_n^+ = d_n^- then
7:             intermediates ⊢ n
8:     G_R.V = largest_components \ intermediates
9:     for n, m ∈ G_R.V do
10:        if (n, m) ∈ G.E then
11:            G_R.E ⊢ (n, m)
12:    for n ∈ intermediates do
13:        m = S_n[0]
14:        forward_path = (n)
15:        for m ∈ intermediates do
16:            forward_path ⊢ m
17:            if S_m[0] ∈ forward_path then
18:                m ← S_m[1]
19:            else
20:                m ← S_m[0]
21:        end ← m
22:        m = S_n[1]
23:        backward_path = (n)
24:        for m ∈ intermediates do
25:            backward_path ⊢ m
26:            if P_m[0] ∈ backward_path then
27:                m ← S_m[1]
28:            else
29:                m ← S_m[0]
30:        start ← m
31:        intermediates ← intermediates \{ backward_path, forward_path}
32:        path ⊢ start ⊢ (reverse backward_path \{n}) ⊢ forward_path ⊢ end
33:        e ← (start, end)
34:        e.path ← path
35:        G_R.E ⊢ e
36:        if (path[0], path[1]) ∈ G.E ∧ (path[0], path[1]) ∈ G.E then
37:            e' ← (end, start)
38:            e'.path ← reverse path
39:            G_R.E ⊢ e'
40: end function
```

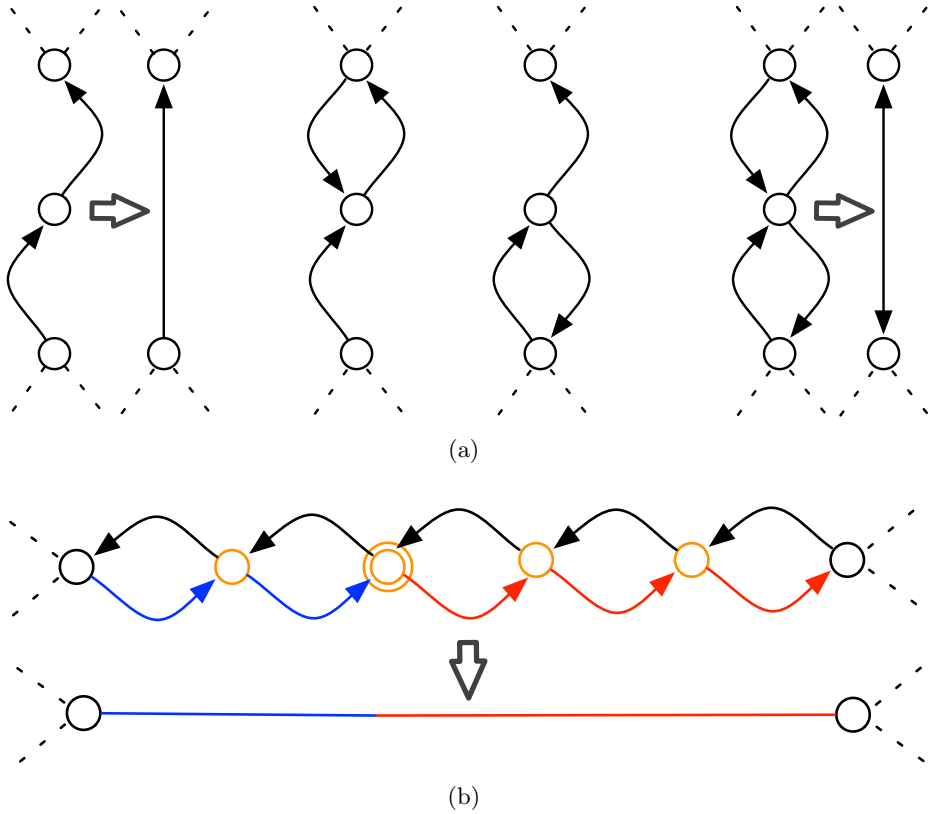


Abbildung 3.5.: Identifizierung und Konvertierung von Intermediate-Knoten

3.2. Architektur und Funktionalität

Das GRF ist eine Middelware-Komponente zwischen Routingprotokollen und Simulatoren. Diese Architektur ist in Abbildung 3.6 dargestellt. Die obere Schicht bilden die Routingprotokolle, die untere die Simulatoren. Die Middlewarefunktionalitäten des GRF werden durch die folgenden Module bereitgestellt:

Init: Dient der Initialisierung der Protokolle und Simulatoren. Dazu müssen Straßennetze und Verkehrspläne in unterschiedliche Formate umgewandelt werden. Es sind hierzu Tools entstanden, die aus OpenStreetMap-Daten die notwendigen Dateiformate erzeugen.

Token: Das Token-Modul kapselt alle relevanten Daten der Fahrzeuge, wie Start- und Zielknoten, zurückgelegten Pfad und aktuelle Position der Token.

Graph: Dieses Modul verwaltet die Graphen des Straßennetzes. Dabei muss eine transparente Vermittlung zwischen Simulator (physikalischer Graph) und Routingprotokolle (Routinggraph) stattfinden. Aus dem Simulationsfeedback werden die aktuellen Auslastungen der Links bestimmt.

Visualisierung: Das GRF bietet eine rudimentäre Visualisierung für relevante Debug-Informationen. Zwar bieten die Simulatoren ebenfalls Visualisierungen, insbesondere auch zur

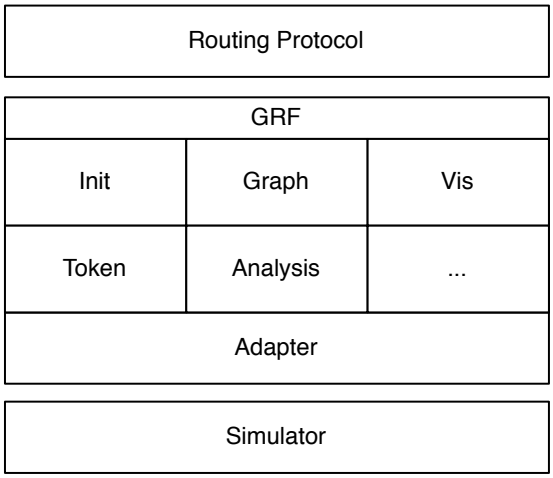


Abbildung 3.6.: Struktur des Generic Routing Framework

Darstellung des fließenden Verkehrs, jedoch nicht für Protokoll-spezifische Daten. Umgesetzt sind die nötigsten Funktionen, um die Funktionsweise der Protokolle nachvollziehen zu können.

Auswertung: Dieses Modul bereitet während und nach einer Simulation die erhobenen Werte der Links und Fahrzeuge statistisch auf.

Verteilung: Das BeeJamA-Protokoll wurde zunächst prototypisch, obschon dezentral ausgelegt, zentral implementiert, d.h. mit Zugriff auf einen gemeinsamen Speicher. Anschließend wurde eine rein dezentrale Implementierung umgesetzt, bei der die einzelnen Navigatoren auf unterschiedlichen Computer laufen können und die Kommunikation mittels TCP/IP oder UDP abgewickelt wird (statt per Methodenaufruf wie in der zentralen Implementierung). Die dabei entstandene Bibliothek *Generic Communication Library* kann dann dazu verwendet werden, dezentrale Protokolle auch leicht dezentral zu implementieren.

Adapter: Pro unterstütztem Simulator muss ein Adapter erstellt werden, der die Funktionalitäten des GRF auf Simulator-spezifische Schnittstellenaufrufe in die richtigen Programmiersprache umsetzt. Gegenwärtig existieren Adapter für drei Simulatoren (MATSim, SUMO, Aimsun), die im nächsten Abschnitt näher erläutert werden.

Das grobe Ablaufkonzept des GRF wird in Abbildung 3.7 dargestellt. Ausgehend von den Eingaben startet das GRF die entsprechende Simulation. Der gestartete Simulator stellt anschließend (vermöge des Adapters) jedesmal, wenn ein Fahrzeug eine Kreuzung erreicht, beim GRF die Anfrage nach einem *Next Hop*, d.h. nach dem als nächstes anzusteuernden Knoten auf dem Weg zum Ziel. Diese wird vom GRF an das zuständige Routingprotokoll des Fahrzeugs weitergeleitet und die Antwort zurückgegeben. Gleichzeitig aktualisiert das GRF kontinuierlich den internen Routinggraph mit den geänderten Positionen der Fahrzeuge und die Transitzeiten der Links. Die Routingprotokolle berechnen, basierend auf diesen Kosten, die Pfade bzw. den Next Hop. Dieser Prozess wird wiederholt, bis alle Fahrzeuge ihr individuelles Ziel erreicht haben.

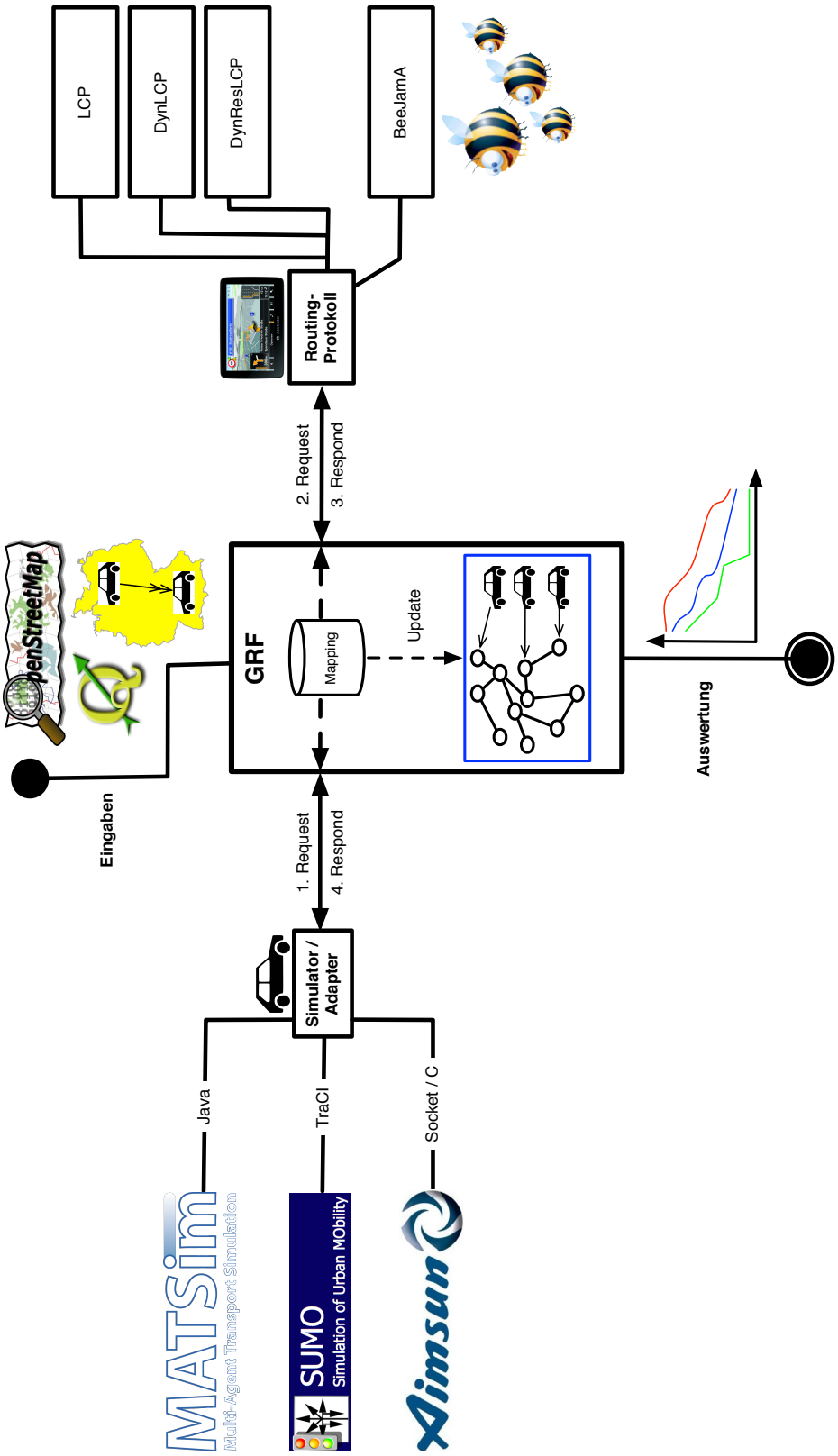


Abbildung 3.7.: Funktionalität des GRF

3.3. Verkehrssimulation

Dieser Abschnitt widmet sich zunächst den Grundlagen der Verkehrsdynamik, um einige wesentliche Begriffe einzuführen. Anschließend werden die drei Simulatoren vorgestellt, für die gegenwärtig GRF-Adapter existieren.

3.3.1. Grundlagen der Verkehrsdynamik

Unter der Dynamik des Straßenverkehrs sei im Folgenden die temporale Ortsveränderung von Teilnehmern des Straßenverkehrs verstanden. Eine einführende Monographie zu dem Thema stellt [145] dar, dieser Abschnitt orientiert sich an den dortigen Ausführungen.

Dabei bezeichnet ein Querschnittsmerkmal eine Verkehrseigenschaft, die an einem fixen Ort x zu einem Zeitpunkt t gemessen werden kann. Technische Messvorrichtungen hierfür sind bspw. Induktionsschleifendetektoren, die orthogonal zur Fahrtrichtung angebracht sind, sowie Videoüberwachungssysteme. Querschnittsmerkmale sind folglich konkrete Messungen des einzelnen Fahrzeugs, sie werden daher *mikroskopische* Daten genannt.

Dabei wird das aktuell gemessene Fahrzeug mit α bezeichnet. Die Fahrtrichtung des Fahrzeugs wird auch als Downstream-Richtung (stromabwärts) bezeichnet. Die entgegengesetzte als Upstream-Richtung (stromaufwärts). Dabei ist stets die gleiche Fahrbahn von α gemeint, die Upstream-Richtung entspricht demnach nicht der entgegengesetzten Fahrbahn (z.B. auf der Autobahn) oder Gegenspur (z.B. innerorts). Die Fahrzeuge hinter α auf der selben Fahrbahn (also in Upstream-Richtung) werden absteigend durchnummeriert. Das direkt als nächstes den Querschnitt passierende Fahrzeug wird daher mit $\alpha - 1$ bezeichnet, das direkt zuvor passierende als $\alpha + 1$.

Das elementarste zu detektierende Merkmal ist die Überfahrt eines Fahrzeugs über einen Querschnitt der Straße an Ort x zum Zeitpunkt t. Bei doppelten Induktionsschleifendetektoren sind zwei Induktionsschleifen in kurzem Abstand angebracht, sodass bei Annahme einer konstanten Geschwindigkeit zwischen den beiden Detektoren leicht die Geschwindigkeit v_α des Fahrzeuges α bestimmt werden kann. Die Zeitpunkte der Querschnittspassage von α und $\alpha - 1$ seien mit t_α respektive $t_{\alpha-1}$ bezeichnet. Die (Brutto-)*Zeitlücke* dieser Fahrzeuge beträgt dann

$$\Delta t_\alpha = t_\alpha - t_{\alpha-1}, \tag{3.1}$$

der (Brutto-)*Abstand*

$$\Delta d_\alpha = v_{\alpha-1} \Delta t_\alpha. \tag{3.2}$$

Aus diesen mikroskopischen lassen sich *makroskopische* Daten aggregieren, d.h. gemittelte Daten über mehrere Fahrzeuge, um Aussagen über Fahrzeugströme zu erhalten. Die elementarste makroskopische Größe ist der Fluss über den Querschnitt. Werden ΔN Fahrzeuge $\alpha_1, \ldots, \alpha_N$ in einem Zeitintervall $[t, t + \Delta t]$ detektiert, dann ist der *Verkehrsfluss*, kurz Fluss, Q als Überquerung pro Zeiteinheit (i.d.R. Fahrzeuge pro Stunde, Fz/h) gegeben durch

$$Q(x,t) = \frac{\Delta N}{\Delta t} = \frac{\Delta N}{\sum_{\alpha=\alpha_1}^{\alpha_N} \Delta t_\alpha}, \tag{3.3}$$

und die *makroskopische Geschwindigkeit* V (i.d.R. in Kilometer pro Stunde, km/h) durch

$$V(x,t) = \frac{\sum_{\alpha=\alpha_1}^{\alpha_N} v_\alpha}{\Delta N}. \tag{3.4}$$

Die Anzahl der Fahrzeuge pro Strecke, genannt *Verkehrsdichte* D, komplettiert die drei grundlegenden makroskopischen Merkmale (i.d.R. in Fahrzeuge pro Kilometer, Fz/km):

$$D(x,t) = \frac{\Delta N}{\sum_{\alpha=\alpha_1}^{\alpha_N} \Delta d_\alpha}. \tag{3.5}$$

Im späteren Verlauf der Arbeit wird zudem die *Auslastung* L, ein Link-spezifisches Merkmal, benötigt:

$$L = \frac{\text{Anzahl der Fahrzeuge auf dem Link}}{\text{Maximal mögliche Anzahl der Fahrzeuge auf dem Link}}. \tag{3.6}$$

Der Nenner ist i.d.R. vorgegeben, da Simulatoren bspw. eine konstante und homogene Fahrzeuglänge vorgeben.

Die drei fundamentalen, makroskopischen Merkmale Geschwindigkeit, Dichte und Fluss bilden den folgenden Zusammenhang (wobei die zuvor empirisch bestimmten Merkmale als Zufallsvariablen aufgefasst werden):

Satz 3.1 (Wardrop-Formel)**.** Für einen Querschnittsort x, einen Zeitpunkt t und den zugehörigen Verkehrsfluss Q, die Verkehrsdichte D und der makroskopischen Geschwindigkeit V gilt:

$$D = \frac{Q}{V}\left(\frac{1}{1 + cv(V)cv(Q)^{-1}\rho(v_\alpha, \Delta t_\alpha)}\right), \tag{3.7}$$

wobei $cv(\cdot)$ den Variationskoeffizienten und $\rho(\cdot)$ den Korrelationskoeffizienten bezeichnet.

Beweis. (nach [145, S. 18]) Ausgehend von Gleichung 3.5 gilt:

$$\begin{aligned}
D &= E(d_\alpha)^{-1} = E(v_{\alpha-1}\Delta t_\alpha)^{-1} \\
&\overset{(1)}{=} E(v_\alpha \Delta t_\alpha)^{-1} \\
&\overset{(2)}{=} (E(v_\alpha)E(\Delta t_\alpha) + cov(v_\alpha, \Delta t_\alpha))^{-1} \\
&= \left(\frac{V}{Q} + cov(v_\alpha, \Delta t_\alpha)\right)^{-1} \\
&= \frac{Q}{V}\left(\frac{1}{1 + \frac{Q}{V}cov(v_\alpha, \Delta t_\alpha)}\right) \\
&\overset{(3)}{=} \frac{Q}{V}\left(\frac{1}{1 + cv(V)cv(Q)^{-1}\rho(v_\alpha, \Delta t_\alpha)}\right)
\end{aligned} \tag{3.8}$$

Es wird bei (1) ausgenutzt, dass $v_{\alpha-1} \approx v_\alpha$ aufgrund der Annahme gleicher Geschwindigkeit aufeinander folgender Fahrzeug sogar in scharfer Version $v_{\alpha-1} = v_\alpha$ gilt.

ad (1) $v_{\alpha-1} = v_\alpha$
ad (2) Ausnutzung des Verschiebungssatzes der Kovarianz $cov(\cdot)$
ad (3) Ausnutzung der Variations- und Korrelationskoeffizientendefinition □

In der Praxis wird aber häufig der Faktor in der Klammer weggelassen und die folgende einfachere Beziehung verwendet:

Definition 3.2 (Fundamentalgleichung des Verkehrsflusses)**.**

$$Q(x,t) = D(x,t) \cdot V(x,t) \tag{3.9}$$

Diese Vereinfachung ist der Spezialfall des *freien Verkehrs*, der dadurch gekennzeichnet ist, dass Fahrer ihre Geschwindigkeit aufgrund geringer Dichte vollkommen losgelöst von dem Verhalten anderer Fahrer wählen können. Folglich können die Zeitlücken als zufällig angenommen werden und sind mithin unkorreliert mit den Fahrzeugsggeschwindigkeiten. Somit ist $\rho(v_\alpha, \Delta t_\alpha)$ in der Realität nahe Null und die Klammer ergibt 1. Im *gebundenen Verkehr*, bei dem Fahrer ihre Geschwindigkeit an die anderer Fahrer anpassen müssen, sinken die Geschwindigkeiten bei steigender Dichte bis hin zum Stillstand. Dadurch wachsen die Zeitlücken für $V \to 0$ in Unendliche, da $\alpha - 1$ den Querschnitt niemals passieren wird. Dadurch wird der Korrelationskoeffizient negativ (steigende Bruttolücken, sinkende Geschwindigkeiten), der Nenner wird kleiner 1 und die Dichte somit für den Fall des gebundenen Verkehrs durch die Fundamentalgleichung im Vergleich zu der Wardrop-Formel unterschätzt. In der Praxis hat sich dieses einfachere Modell jedoch durchgesetzt.

Aus dieser Fundamentalgleichung lassen sich aufschlussreiche Visualisierungen ableiten. Zuvor wird jedoch noch der Begriff der *Kapazität* benötigt, der zweierlei Bedeutung haben kann:

Designkapazität: Beschreibt den Fluss für den eine Straße baulich (Anzahl Spuren, Breite, Maximalgeschwindigkeit, ...) konzipiert wurde, um die üblicherweise erwartete Nachfrage abzudecken. In der Regel werden Reserven vorgesehen, um Nachfragespitzen befriedigen zu können.

Maximalkapazität: Beschreibt den maximalen Fluss (abhängig u.a. von der Maximalgeschwindigkeit und der Fahrzeuglänge) der pro Zeiteinheit den Link passieren kann.

In dieser Arbeit ist im Folgenden, wenn nicht anders angemerkt, stets die Maximalkapazität gemeint.

Mittels dieser Begriffe kann das sogenannte *Fundamentaldiagramm des Verkehrsflusses* erstellt werden. Dieses Diagramm stellt einen idealisierten Zusammenhang zwischen den beiden Größen Fluss und Dichte dar. Dabei wird angenommen, es handle sich um identische Fahrzeuge mit äquidistantem, der jeweiligen Geschwindigkeit perfekt angepasstem, Abstand – es wird also ein sogenanntes Verkehrsgleichgewicht angenommen. Daneben werden die Zusammenhänge zwischen Geschwindigkeit/Dichte und Fluss/Geschwindkeit mitunter, so auch hier, ebenfalls als Fundamentaldiagramme bezeichnet (vgl. Abbildung 3.8). In dem Evaluationsteil werden später empirische Varianten dieser (idealisierten) Fundamentaldiagramme verwendet.

Der in der genannten Abbildung dargestellte D-V-Zusammenhang, wird in der sogenannten Drei-Phasen-Theorie des Staus [76] namensgebend in drei Phase unterteilt: 1) Geringe Dichte führt zu hohen (makroskopischen) Geschwindigkeiten, weswegen dieser Zustand als „frei“ bzw. „ungebunden“ bezeichnet wird. 2) In einer Transitionsphase, bei dem Verkehrs als „gebunden“ bezeichnet wird, reduziert sich die Geschwindigkeit aufgrund zunehmender Dichte. 3) Schließlich entsteht Stau. Wie die Einteilung dieser drei Phasen exakte vorgenommen wird, ist prinzipiell willkürlich und daher nicht stets einheitlich.

Die übrigen beiden Diagramme sind ähnlich strukturiert. Im V-Q-Diagramm ist erkenntlich, dass der höchste Fluss bei einer mittleren Geschwindigkeit entsteht. Bei geringerer Geschwindigkeit herrscht Stau, bei höherer nimmt der nötige Sicherheitsabstand zu (bis schließlich, rein theoretisch, im Beobachtungszeitraum kein Fahrzeug mehr über den Querschnittsdetektor fährt).

Im D-Q-Diagramm ist bei mittlerer Dichte der Fluss maximal. Bei zu geringerer Dichte, ist die Kapazität des Links nicht ausgeschöpft. Bei zu hoher Dichte herrscht Stau.

Wie intuitiv einsichtig, entsteht Stau demnach dann, wenn die (Design-)Kapazität von Verkehrswegen überschritten wird. Ursächlich dazu beitragen können drei Faktoren:

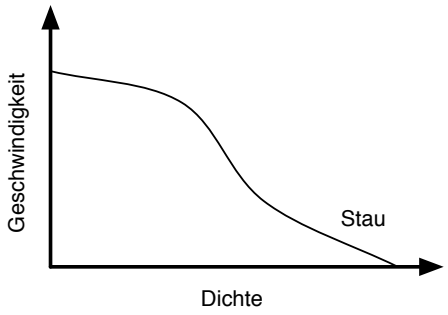
Menschliches oder technisches Versagen: Menschliche Reaktionszeiten, individuelles Können und Fahrausbildungsstand bedingen die Anzahl von Unfällen, ebenso wie technische Mängel von Fahrzeugen oder der Infrastruktur. Aus alltäglicher Erfahrung ist bekannt, dass Unfälle und beispielsweise daraus resultierende Autobahnvollsperrungen, teils erheblichen Stau induzieren können. Neben der primären Sicherheitsverbesserung ist Forschung in diesem Bereich somit auch der Staureduzierung dienlich.

Zu hohe Verkehrsnachfrage: Ist die Verkehrsnachfrage schlichtweg so groß, dass es keine Möglichkeit gibt, selbst durch geschicktes Umleiten, die Fahrzeugströme unter der infrastrukturellen Designkapazität zu halten, entsteht Stau. Beispielsweise ist dieses Phänomen häufig bei Großveranstaltungen zu beobachten. Eine Umleitung zum Veranstaltungsort bzw. von dort weg ist nicht möglich, da alle Fahrzeuge den Veranstaltungsort erreichen bzw. wieder verlassen müssen. Die Kapazität umliegender Verkehrswege wird dabei regelmäßig überschritten. Als Konsequenz entsteht Stau, der höchstens durch Ausbau von Infrastruktur oder alternative Transportmitteln gemildert werden könnte.

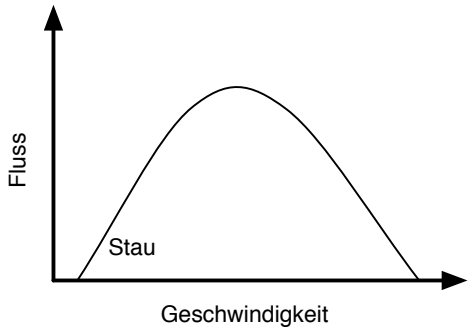
Unkoordinierte Routenführung: Werden bestehende Umleitungsmöglichkeiten, deren Benutzung die Reisezeit nicht verlängern, nicht verwendet, kann es zu vermeidbaren Staus kommen, die durch mangelnde Koordination entstehen. Entweder kennen Fahrzeuge die Umleitungen gar nicht oder antizipieren zu hohe Kosten in Form von Fahrzeitverlängerungen bzgl. dieser Umleitungen. Durch verstärkte Koordination ließe sich in diesem Falle Stau vermeiden.

Der letzte Punkt ist Gegenstand dieser Arbeit, die ersten beiden Punkte werden durch diese Arbeit hingegen nicht angegangen.

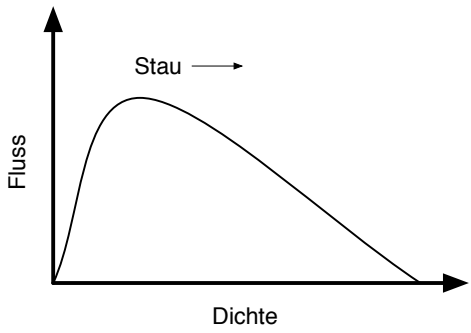
Abschließend soll in diesem Abschnitt der Begriff der *Link Performance Function* (LPF) eingeführt werden. Dabei handelt es sich im Wesentlichen um ein Link-spezifisches, empirisch ermitteltes Q-V-Diagramm. Statt der makroskopischen Geschwindigkeit wird allerdings die *Transitzeit* τ, d.h. die benötigte Zeit eines Fahrzeugs für eine Linkpassage, verwendet. Dadurch drückt eine LPF aus, wie lange ein Fahrzeug bei einem bestimmten gegenwärtigen Fluss über einen Link für eine Passage derselben benötigt. Die prinzipielle Aussage einer LPF: Je



(a) DV



(b) VQ



(c) DQ

Abbildung 3.8.: Fundamentaldiagramme

ausgelasteter ein Link, desto höher die Transitzeit. Dieser funktionale Zusammenhang wird nachher eingesetzt, um bei bekannter zukünftiger Linkauslastung eine Transitzeit-Abschätzung abzuleiten. Dadurch kann eingeschätzt werden, ob es sinnvoll ist, ein Fahrzeug über einen bestimmten Pfad zum Ziel zu leiten.

Für einen Link l mit Länge len_l und der makroskopischen Geschwindigkeit V zu einem fixen Zeitpunkt ist die Transitzeit definiert durch $\tau_l = \frac{len_l}{V}$. Eine minimale Transitzeit wird als *Freiflusszeit* oder *Free Flow Time* (FFT), nachfolgend auch als *FF-Transitzeit*, bezeichnet und ist gegeben durch $\tau_l^0 = \frac{len_l}{v_l^{max}}$, mit v_l^{max} als (erlaubter oder maximal möglicher) Höchstgeschwindigkeit (oder auch *Freiflussgeschwindigkeit*) auf l.

Um eine LPF η für einen Link zu bestimmen, muss ein funktionaler Zusammenhang zwischen Fluss und Transitzeit ermittelt werden. Für amerikanische Straßenverhältnisse hat das U.S. Bureau of Public Roads (BPR) solche Zusammenhänge für typische Straßenklassen approximativ ermittelt. Das Ergebnis ist die, in diesem Kontext häufig verwendete, sogenannte *BPR-Funktion*.

Definition 3.3 (BPR-Funktion)**.** Für einen Link l mit FFT τ_l^0, Designkapazität c_l und gegenwärtigen Fluss $Q_l > 0$ gilt:

$$\eta_1(Q_l) = \tau_l^0 \left(1 - \alpha \left(\frac{Q_l}{c_l}\right)^{\beta}\right), \tag{3.10}$$

mit $\alpha > 0$ und $\beta > 0$ als zwei Parametern zur Anpassung an unterschiedliche Linktypen.

So muss nicht für jeden Link ein neuer funktionaler Zusammenhang bestimmt werden, sondern nur zwei Parameter entsprechend gewählt werden. Ein Tabellenwerk mit Parameterwerten für α, β zu verschiedenen Straßentypen findet sich in [67]. Für die spätere Verwendung der BPR-Funktion ist folgende Proposition relevant:

Proposition 3.4. Die BPR-Funktion nach Gleichung 3.10 ist *konvex*, aufgrund der stets positiven zweiten Ableitung, $\forall Q_l \in \mathbb{R}_+, \alpha > 0, \beta > 0$: $\eta_1''(Q_l) > 0$.

3.3.2. Arten der Verkehrssimulation

Verkehrssimulation beschreibt den Vorgang anhand eines *Verkehrsmodells*, die Dynamik von Verkehr zu simulieren. Eine gängige Einordnungsmöglichkeit dieser Modelle ist die Level-of-Detail-Kategorisierung, bei der nach drei, im Detailgrad aufsteigende, Hauptkategorien unterschieden wird:

Makroskopisch: Hierbei werden nur Fahrzeugströme beschrieben, einzelne Fahrzeuge als Simulationsentität existieren nicht. Diese Ströme werden mittels akkumulierten Informationen, z.B. der makroskopischen Dichte, beschrieben. Dadurch können diese Ströme als Fluide aufgefasst werden, weswegen Verkehrsmodelle mitunter auf physikalischen Fluidmodellen basieren. Solche Verkehrsmodelle beschreiben dann die temporale Veränderung des Verkehrs durch (Fahrzeug-)Dichteveränderungen auf den Links.

Mikroskopisch: Hierbei werden einzelne, diskrete Fahrzeuge simuliert. Je nach Präzision des Verkehrsmodells kann zeit- und ortsdiskret oder -kontinulierich die Position jedes

Fahrzeuges zu einem bestimmten Zeitpunkt angegeben werden. Die Genauigkeit und Auflösung dieser Position hängt vom Verkehrsmodell ab.

Submikroskopisch: Hierbei werden sogar einzelne Bauteile der Fahrzeuge (in der Regel mittels Differentialgleichungen) beschrieben und simuliert.

Generell gilt, je höher der Detailgrad, desto rechenintensiver das Verfahren. Makroskopische Verfahren eignen sich, wenn nur Linkauslastungen relevant sind. Ein einzelnes Fahrzeug wird dabei nicht als Simulationsentität berücksichtigt. Für das in dieser Arbeit vorgestellte Verfahren ist das zu ungenau, da jedem Fahrzeug an jeder Kreuzung eine individuelle Routingentscheidung mitgeteilt werden soll (siehe Ziel 2 in Abschnitt 1.2), wozu die aktuelle Position benötigt wird. Mikroskopische Verfahren liefern genau diese Information und werden im weiteren Verlauf dieses Kapitels diskutiert. Submikroskopische Modelle werden z.B. während der Entwicklung eines Fahrzeugs und für (virtuelle) Crash-Tests benötigt und sind für die Belange dieser Arbeit zu genau (und daher zu rechenaufwendig).

Mikroskopische Modelle können, nach steigender Komplexität und Realitätsgrad, weiter unterteilt werden:

Queue-basiert: Bei diesen Modellen werden Links als Queues interpretiert und eintreffende Fahrzeuge in diesen platziert. Pro Zeiteinheit darf ein gewisser Anteil an Fahrzeugen den Link passieren, die restlichen verbleiben in der Warteschlange. Dadurch ist über den aktuellen Ort eines Fahrzeuges nur der gegenwärtige Link bekannt, nicht aber die exakte Position darauf.

Zellulare Automaten: Bei diesen Modellen werden Links in Zellen eingeteilt und Regelsysteme eingesetzt, die die Belegung der Zellen in dem aktuellen Zeitschritt aus dem Zustand des vorherigen Zeitschrittes bestimmen. Es handelt sich demnach um orts- und zeitdiskrete Modelle, eine Zelle ist zu einem bestimmten Zeitpunkt entweder leer oder belegt. Für jedes Fahrzeug ist somit die Aufenthaltszelle zu jedem Zeitpunkt exakt bekannt.

Fahrzeugfolgemodelle: Darüber hinaus existieren orts- und zeitkontinuierliche Modelle, sogenannte Fahrzeugfolgemodelle. Dazu werden (partielle) Differentialgleichungen verwendet. Diese Modelle erreichen die größte Genauigkeit, benötigen aber auch am meisten Rechenkraft.

Queue-basierte Modelle eignen sich aufgrund der geringsten (Rechen)-Anforderungen für große Netze und werden häufig zur Evaluation neuer verkehrsplanerischen Konzepte angewendet, da dies mitunter mit einer häufigen Simulation desselben Netzes, aber geänderten Parametern einhergeht. Das andere Extrem, die mikroskopischen Fahrzeugfolgemodelle, werden häufig in kommerziellen Systemen verwendet und von Behörden genutzt, um gezielt einzelne infrastrukturelle Maßnahmen wie neue oder geänderte innerstädtische Kreuzungen oder Autobahnabschnitte samt Auffahrten bewerten zu können.

Für das GRF wurden, wie zuvor erwähnt, Adapter für drei Simulatoren umgesetzt. MATSim ist Queue-basiert, SUMO und Aimsun basieren auf Fahrzeugfolgemodellen. Zusätzlich können die Verkehrsmodelle dahingehend unterschieden werden, ob und welche Lateral- und Longitudinal-Modelle eingesetzt werden. Ersteres bezeichnet das seitliche Verhalten, bei mikroskopischen Verkehrsmodellen insbesondere das Spurwechselverhalten. Letzteres bezeichnet das Verhalten in Vorwärtsrichtung. In der folgenden Beschreibung der drei GRF-Simulatoren wird darauf entsprechend eingegangen, um die späteren Simulationsergebnisse besser einschätzen zu können.

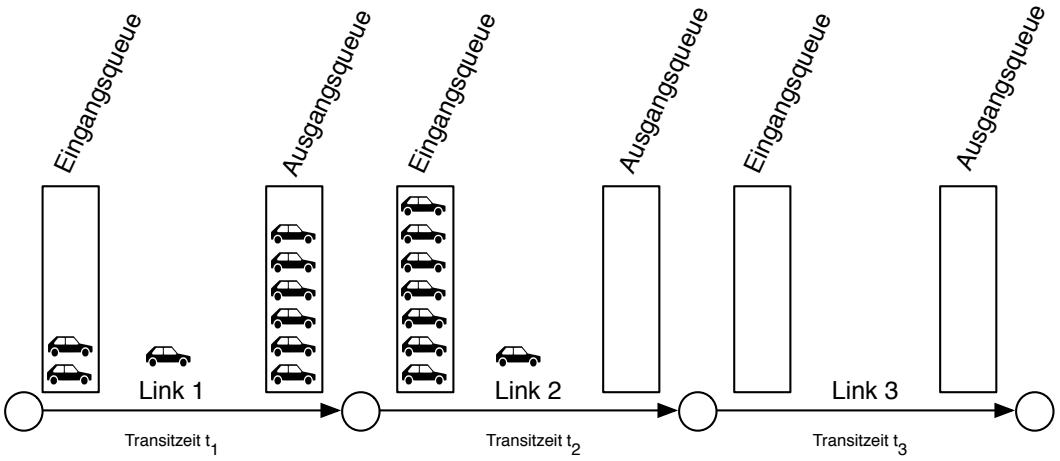


Abbildung 3.9.: MATSim Funktionsweise

3.3.3. MATSim

MATSim [6, 93] ist ein mikroskopischer Open-Source Verkehrssimulator, basierend auf einem Queue-Modell. Große Netze simulieren zu können, ist ein primäres Ziel. Neben dem reinen Verkehrsgeschehen, können auch sozioökonomische Faktoren mit den Verkehrsteilnehmern verknüpft werden. So können Fragen derart nachgegangen werden, wie finanzkräftig die Nutzer bestimmter Links sind (z.B. für Fragen rund um das Thema Maut interessant). Für diese Arbeit spielen diese zusätzlichen Funktionen jedoch keine Rolle.

Das Verkehrsmodell verknüpft jeden Link mit einer Eingangs- und einer Ausgangsqueue. Zusätzlich ist für jede Kante aus den eingelesenen Daten die Länge, Maximalgeschwindigkeit, Straßentyp und Anzahl der Spuren bekannt. Aus den beiden erstgenannten Attributen ergibt sich durch Division die Freiflussgeschwindigkeit. Der Straßentyp gibt an, um welche Ausbaugrad es sich handelt. Unterschieden werden zwischen kleinen, mittleren und großen Straßen. Damit verknüpft wird eine Spurkapazität von 600, 800 und 1000 Fz/h. Mittels Multiplikation mit der Anzahl der Spuren ergibt sich so für jeden Link eine (Gesamt-)Kapazität. MATSim interpretiert diese Kapazität als *maximale Kapazität*, d.h. mehr Fahrzeuge können pro Stunde den Link nicht passieren. Anders ausgedrückt, die minimale Zeitlücke ergibt sich durch Division von Kapazitätsperiode (Zeiteinheit auf den sich die Kapazität bezieht) und (maximaler) Kapazität.

Die prinzipielle Funktionsweise von MATSim ist in Abbildung 3.9 dargestellt. Startende Fahrzeuge werden, sofern Platz vorhanden ist, zu der Eingangsqueue hinzugefügt. Ist kein Platz, kann das Fahrzeug nicht rechtzeitig starten und muss warten bis die Eingangsqueue freie Restkapazität aufweist. In der Frequenz der minimalen Zeitlücke wird geprüft, ob die Ausgangsqueue desselben Links noch mindestens einen freien Platz hat. Ist dies der Fall, passiert das Fahrzeug den Link und zwar in Freiflussgeschwindigkeit. Anschließend wird das Fahrzeug der entsprechenden Ausgangsqueue hinzugefügt. Einen Zeitschritt später werden so viele Fahrzeuge wie möglich in die Eingangsqueue des nächsten Links kopiert. Erreicht ein Fahrzeug das Ziel (genau genommen, wird das Fahrzeug aus der Ausgangsqueue des Ziellinks in die Eingangsqueue eines beliebigen nachfolgenden Links kopiert), wird es aus dem System entfernt.

Da MATSim keine Lateralmodelle simuliert, damit kein Überholen stattfindet, erfüllt es die FIFO-Eigenschaft (d.h. kein später den Link befahrendes Fahrzeug verlässt den Link eher).

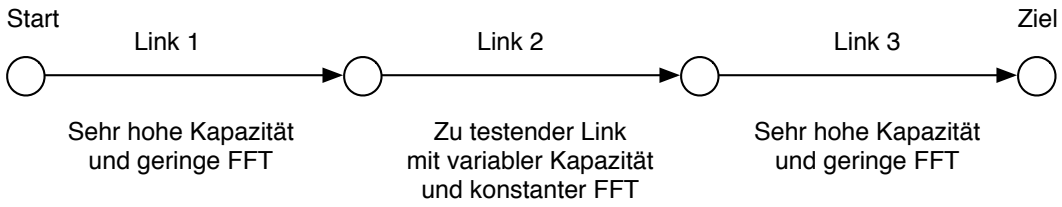


Abbildung 3.10.: Simulationsaufbau zur Link Performance

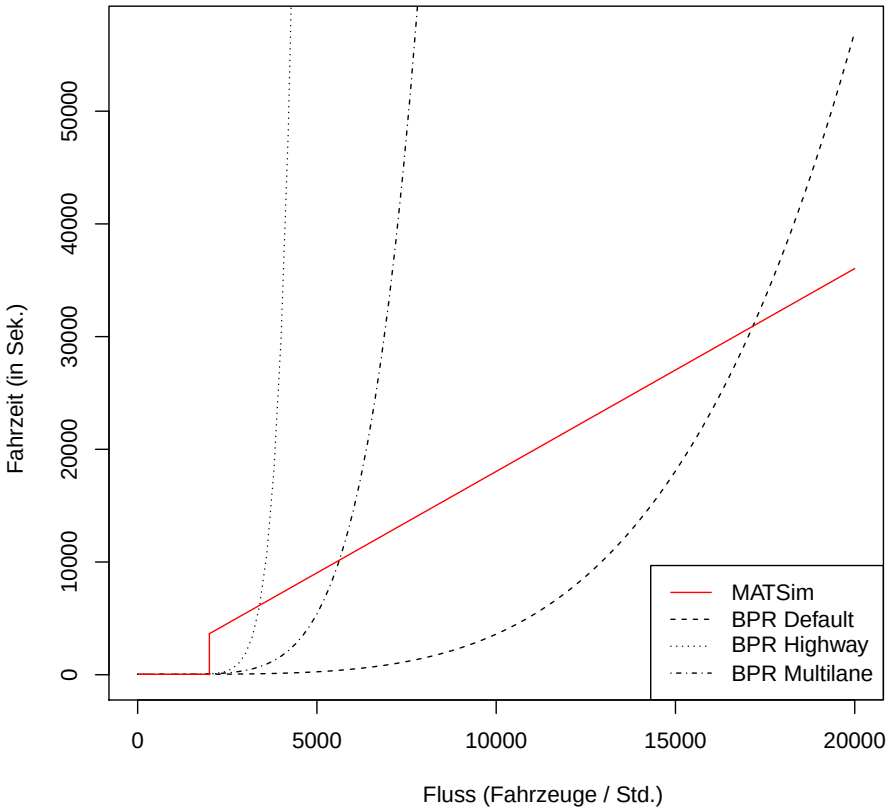


Abbildung 3.11.: MATSim Link Performance Function im Vergleich zur BPR-Funktion

Für MATSim wurde im Rahmen dieser Arbeit eine spezifische LPF abgeleitet, statt die generische BPR-Funktion zu nutzen. Diese MATSim-LPF wird später für ein Reservierungskonzept in BeeJamA verwendet, um zukünftige Transitzeiten abzuschätzen. Ermittelt wurde die LPF mittels des Simulationsaufbaus in Abbildung 3.10. Die Fahrzeuge werden vom Start- zum Zielknoten geleitet, wobei der 1km lange Link 2 eine „Engstelle“ bildet. Als Parameter wurde ceteris paribus die Kapazität dieses Links verändert (bei nacheinander getesteten, unterschiedlichen Maximalgeschwindigkeiten). Anhand der empirischen Daten war schnell ersichtlich, dass das MATSim-Verkehrsmodell zu einer LPF führt, die aus zwei (linearen) Teilstücken zusammengesetzt ist. Durch weitergehende Analyse des Source Codes konnte die

exakte LPF rekonstruiert werden.

Definition 3.5 (MATSim-LPF)**.** Für jeden Fluss $Q_l > 0$ des Links l gilt in MATSim für die Transitzeit:

$$\eta_2(Q_l) = \begin{cases} \tau_l^0, & Q_l < \hat{c}_l \\ \tau_l^0 + \left((Q_l - 1)\dfrac{3600}{\hat{c}_l}\right), & sonst, \end{cases} \tag{3.11}$$

wobei $\hat{c}_l$ die Kapazität und τ_l^0 die FFT des Links l darstellt.

Dabei ist festzuhalten, dass diese Funktion so nicht im Source Code vorhanden ist. Vielmehr entsteht dieses Verhalten aus dem Zusammenwirken des zuvor beschriebenen Queuemodells. Die Abbildung 3.11 zeigt die MATSim-LPF. Dazu ist die BPR-Funktion mit drei Parametrisierungen dargestellt[2]. Wie aus dieser Darstellung ersichtlich wird, überapproximiert die MATSim-LPF die Fahrzeit im Vergleich zu den BPR-Varianten zunächst, unterapproxmiert diese jedoch bei höheren Flüssen. Allein danach ist zu vermuten, dass MATSim andere Simulationsergebnisse liefert als Simulatoren mit komplexeren Verkehrsmodellen. Nachfolgende simulative Untersuchungen werden diesen Umstand näher beleuchten.

3.3.4. Aimsun

Aimsun [9, 147] ist ein kommerzieller, proprietärer Verkehrssimulator, der ein zeit- und ortsdiskretes Fahrzeugfolgemodell umsetzt. Nach Angaben des Herstellers TSS basiert Aimsun auf dem Gipps-Verkehrsmodell [55]. Dieses Modell geht davon aus, dass nachfolgende Fahrzeuge stets kollisionsfrei fahren. Entsprechend wird die Geschwindigkeit derart reguliert, dass ein Mindestabstand niemals unterschritten wird. Dazu wird eine „sichere Geschwindigkeit“ $v_{\text{safe}}(s_\alpha, v_{\alpha-1})$ angestrebt, die abhängig ist vom Abstand s_α zum vorausfahrenden Führungsfahrzeug $\alpha - 1$ und dessen Geschwindigkeit $v_{\alpha-1}$. Zusätzlich fließt die als konstant angenommene Verzögerung b des Fahrzeugs und die als ebenso konstant angenommene Reaktionszeit Δt des Fahrers ein. Das Modell garantiert Kollisionsfreiheit für die folgende Definition der sicheren Geschwindigkeit unter allen Umständen (bei einer zeitlichen Auflösung von t Zeiteinheiten und einem minimalen Abstand s_0):

$$v_{\text{safe}}(s_\alpha, v_{\alpha-1}, t) = -b\Delta t\sqrt{b^2t^2 + v_{\alpha-1}^2 + 2b(s_\alpha - s_0)}. \tag{3.12}$$

Das Gipps-Verkehrsmodell ist dann durch die folgende iterierte Abbildung gegeben:

$$v_\alpha(t + \Delta t) = \min\{v_\alpha + a\Delta t, v_0, v_{\text{safe}}(s_\alpha, v_{\alpha-1}, t)\}, \tag{3.13}$$

wobei a eine konstante Beschleunigung des Fahrzeugs angibt und v_0 die Wunschgeschwindigkeit des Fahrzeugs darstellt. Das Gipps-Modell beschleunigt demnach, bis die Wunschgeschwindigkeit erreicht ist und verzögert, wenn die aktuelle über der sicheren Geschwindigkeit liegt.

Im Gegensatz zu dem Queue-Modell von MATSim weist Aimsun durch dieses Modell eine deutlich detailliertere Sichtweise auf: die Fahrzeuge sind zu diskreten Zeitpunkten exakten diskreten Positionen zugeordnet (statt nur einem Link). Auch werden mehrere Spuren unterstützt. Neben dem Longitudinal-Modell, entsteht somit zeitgleich die Notwendigkeit, Spurwechsel zu

[2] Die Parameter stammen aus [67]. Warum ein Highway schneller ansteigende Transitzeiten aufweisen soll als die dort angegeben *Multilane*-Parametrierung ließ sich nicht klären. Eventuell handelt es sich um einen Fehler in dem Ausgangsdokument.

ermöglichen. Als Lateral-Modell wird dabei ein Entscheidungsprozess verwendet, der, sobald ein Fahrzeug einen Spurwechselwunsch anmeldet, prüft, ob und wohin ein Spurwechsel möglich ist. Ein Wechselwunsch liegt vor, wenn von der aktuellen Spur nicht auf den gewünschten Nachfolgelink abgebogen werden kann, oder wenn $v_\alpha < v_0$ (und demnach überholt werden soll), sowie, wenn von einer vorausliegenden Auffahrt Fahrzeuge auf die aktuelle Spur (z.B. einer Autobahn) auffahren wollen. Zusätzlich werden Ampeln und Vorfahrtsschilder unterstützt.

Die Einbindung von Aimsun in das GRF war problembehaftet. Zwar verfügt Aimsun über eine (unzureichend dokumentierte) C-Schnittstelle, welche aber nicht dafür ausgelegt ist, einzelne Fahrzeuge darüber zu steuern. So lassen sich konzeptbedingt nur ganze Fahrzeugklassen (Pkw, Bus, ...) steuern. Einzelnen Fahrzeugen können nur Hinweise gegeben werden, welcher nächste Link angesteuert werden soll. Diese Hinweise werden jedoch nicht selten von Aimsun mit dem vor Simulationsstart berechneten LCP überschrieben.

Trotz dieser Probleme wäre Aimsun aufgrund des detaillierten Verkehrmodells prinzipiell als ein geeigneter Simulator einzustufen. Es stellte sich jedoch heraus, dass an Kreuzungen häufig Deadlocks entstehen – vier einzelne, aus unterschiedlichen Richtungen gleichzeitig ankommende Fahrzeuge blockieren sich gegenseitig unendlich lange. Je dichter der Verkehr, desto häufiger tritt dieser Effekt auf. Abgemildert werden konnte dieser Effekt durch die Verwendung von Vorfahrtsschildern, ganz vermieden jedoch nicht. Der Abschnitt 3.5 vergleicht MATSim und Aimsun für einige einfache Straßennetze, dort wird dieser Punkt weiter aufgegriffen.

3.3.5. SUMO

SUMO [135] ist ein Open Source Verkehrssimulator, der vom Deutschen Zentrum für Luft- und Raumfahrt entwickelt wird. Es wird eine Vielzahl von Verkehrsmodellen unterstützt, einige davon waren zum Zeitpunkt des Testens jedoch nicht voll funktionstüchtig. Das Hauptmodell stellt das Modell nach Krauß [81] dar.

Das Ansprechen und Steuern der einzelnen Fahrzeuge zur Laufzeit erfolgt über die TCP/IP-basierte TraCI-Schnittstelle. Allerdings führte die fehlerhafte Implementierung dieser Zwischenschicht zu erheblichen Problemen. Warten Fahrzeuge zu lange an einer verstopften Kreuzung, wird ein sogenanntes „Teleporting“ aktiviert, bei dem die Fahrzeuge einfach hinter die verstopfte Kreuzung gesetzt werden. Jedoch führte genau dieses Teleporting zu verlorenen Fahrzeugen, so dass der Simulator und das GRF unsychonisiert bzgl. der Aufenthaltsorte der Fahrzeuge wurden. Gänzlich vermeiden ließ sich das Teleporting jedoch auch nicht, da bei hoher Verkehrsdichte rasch Verstopfungen auftraten, die zu sehr langen Wartezeiten für Fahrzeuge auf niedrigpriorisierten Links führten. Niedrigpriorisiert ist ein Link beispielsweise dann, wenn eine kleine Straße (z.B. in Anzahl von Spuren und Höchstgeschwindigkeit) auf eine größere Straße trifft. Da die Vorfahrt über diese Priorisierung geregelt wird, kommt es schnell vor, dass Fahrzeuge auf diesen niedrigpriorisierten Links erst dann auf die höherpriorisierten auffahren dürfen, wenn auf dem Vorrangslink längere Zeit keine Fahrzeuge fuhren. Auch trat das Problem auf, dass Fahrzeuge nicht rechtzeitig Spuren freigaben, so dass auffahrende Fahrzeuge (z.B. auf Autobahnen), ohne vorher zum Stillstand zu kommen, den Spurwechsel auf den nächsten Link nicht vornehmen konnten. Prinzipiell können mittels zusätzlichen Tools heuristisch Ampeln und Autobahnauffahrten gesetzt werden. Zuverlässig funktioniert dies jedoch gegenwärtig nicht. Komplexere Simulation ließen sich so nicht bewerkstelligen. SUMO, obwohl es sich aufgrund der Tatsache, dass es mehrere Verkehrsmodelle unterstützt und so ausgezeichnet zur VRGS-Evaluation geeignet gewesen wäre, lässt sich gegenwärtig nicht für automatisch importierte OpenStreetMap-Daten für große Simulationen nutzen und wird in

dieser Arbeit daher nicht weiter betrachtet.

3.4. Routingprotokolle

Für das GRF wurden verschiedene Routingprotokolle umgesetzt:

LCP: Zentrale und statische LCP-Pfadfindung, entsprechend eines klassischen Navigationssystem, das vor Fahrtantritt eine Route berechnet. Es liegen globale Daten in Form eines Graphen vor, dessen Linkgewichte den FF-Transitzeiten entsprechen. Eine Online-Neuberechnung während der Fahrt findet nicht statt.

DynLCP: Hierbei handelt es sich um die *dynamische* Variante des LCP-Protokolls. Dynamisch bedeutet in diesem Fall, dass online zur Fahrtzeit neue Routen zum Ziel berechnet werden, basierend auf aktualisierten Verkehrsdaten. Der Ablauf dabei ist weiterhin zentralisiert. Die Verkehrsdaten (z.B. Floating Car Data) werden zentral im TIC gesammelt und dem Routingprotokoll verfügbar gemacht. Dazu werden die Linkkosten entweder an die PNA übertragen, welche ihrerseits dann den aktualiserten LCP berechnen, oder es wird eine Anfrage an das TIC gestellt und nur die Antwort zurückgeschickt. TMC und TomTom HD Traffic sind kommerziell verfügbare Systeme, die einen dynamischen LCP umsetzen.

Dem Protokoll muss als Parameter eine Aktualisierungsfrequenz angegeben werden: z.B. bedeutet DynLCP 30min, dass alle 30 Minuten neue, aktualisierte Linkkosten zur Verfügung stehen. Dieser Wert ist global zu verstehen, d.h. alle 30 Minuten stehen am TIC neue Linkkosten für alle Links bereit. Ein Fahrzeug profitiert dann bei der nächsten Routenberechnung davon. Berechnet hingegen ein Fahrzeug 10 Minuten nach der letzten globalen Aktualisierung einen neuen Pfad, dann basiert diese Berechnung auf 10 Minuten „alten“ Informationen, auch wenn aus der individuellen Sicht des Fahrzeugs bereits 30 Minuten seit der letzten Routenneuberechnung vergangen sind.

BeeJamA: Das BeeJamA-Protokoll gemäß Kapitel 6.

Die LCP-Protokolle basieren allesamt auf dem A*-Algorithmus [59]. Wie in Abschnitt 1.5.1 bereits erwähnt, handelt es sich um einen heuristischen Single-Pair-LCP-Algorithmus. Es sei ein LCP π_{ab} in $G = (V, L)$ gesucht. A* setzt eine Heuristik $h(i)$ ein, um einen betrachteten Knoten $i \in V$ frühzeitig (ggf. als $i \notin \pi_{ab}$) bewerten zu können. Für jede Heuristik ist A* vollständig, d.h. eine existente Lösung wird stets gefunden. Auch ist A* optimal effizient, d.h. kein anderer Algorithmus kann mittels derselben Heuristik schneller die Lösung finden. Der Dijkstra-Algorithmus ist ein Spezialfall mit $h(i) = 0, \forall i \in V$ (wodurch stets alle benachbarten Knoten betrachtet werden müssen). Im Allgemeinen ist die Worst Case Komplexität des Algorithmus jedoch exponentiell bzgl. Laufzeit und Speicherverbrauch. Die Komplexität fällt jedoch auf $\mathcal{O}(|V| \log |V|)$, sobald eine sogenannte monotone Heuristik eingesetzt werden kann. Eine Heuristik $h(i)$ heißt in diesem Zusammenhang *monoton* genau dann, wenn die tatsächlichen Kosten $f(i)$ von i zu b niemals überapproximiert werden, $f(i) \geq h(i)$, und wenn die folgende Dreiecksungleichung gilt: die geschätzten Kosten von i sind nicht größer als die tatsächlichen Kosten zu einem Nachfolger $s \in S_i$ plus die geschätzten Kosten von s zu b, $h(i) \leq \omega_{is} + h(s)$. Ist die Heuristik monoton, ist der gefundene Pfad π_{ab} zusätzlich optimal (d.h. es existiert kein anderer Pfad mit geringeren Kosten).

Im Zusammenhang des Routings wird häufig die Euklidische Distanz zwischen Start und Ziel (in Metern) als monotone Heuristik verwendet. Für ein VRGS ist das nicht ausreichend,

da stattdessen relevant ist, wie *lange* – und nicht wie *weit* – ein Fahrer vom Ziel entfernt ist. Es wird kein (im wahrsten Sinne des Wortes) kürzester Pfad gesucht, sondern ein LCP bzgl. der Fahrzeit. Ein passende Heuristik erhält man aber leicht durch Division der Euklidischen Distanz mit der maximal auftreten Geschwindigkeit v_{max}:

$$h(i) = \frac{\sqrt{(x_i - x_m)^2 + (y_i - y_m)^2}}{v_{max}}. \tag{3.14}$$

Dabei ist $v_{max} > 0$ in der Regel bei einer Simulation durch die Eingaben vorgegeben, da die simulierten Fahrzeuge die zwingend in den Eingabedateien anzugegebene Link-Höchstgeschwindigkeit gemäß der eingesetzten Verkehrsmodelle nicht überschreiten. Unlimitierte Höchstgeschwindigkeiten treten daher nicht auf. Die Heuristik überapproximiert die Fahrzeit niemals, denn es gibt keinen kürzeren Weg (in Metern) als die Luftlinie und diese Strecke kann nicht schneller zurückgelegt werden als mit v_{max}. Die Dreiecksungleichung ist ebenfalls erfüllt, weswegen $h(i)$ nach Gleichung 3.14 monoton ist.

Der Algorithmus führt eine Open- und eine Closed-Liste (siehe Algorithmus 2), wobei nur Knoten in der erstgenannten Liste weiter betrachtet werden müssen. Anhand der Heuristik kann entschieden werden, ob ein Knoten von der Open- in die Closed-Liste wechseln muss, wodurch, sofern $h(i)$ monoton, der Vorteil gegenüber des Dijkstra-Algorithmus erzielt wird (bei dem alle nachfolgenden Knoten betrachtet werden). In Algorithmus 3 ist die monotone Heuristik angebgen, wobei $AllowedTurnings_{n,l}$ die Menge der erlaubten Abbiegemöglichkeiten an Knoten n angibt, wenn ein Fahrzeug zuvor an l war. Dort ist auch die DynLCP-Funktion abgebildet. Die Variable t_{now} repräsentiert die aktuelle Uhrzeit und $updateInterval$ eine Konstante, die den Zeitraum zwischen zwei Aktualisierungen angibt.

Protokolldurchdringungen In den späteren simulativen Protokollevaluationen werden auch partielle Marktdurchdringungen (market penetration), kurz: *Durchdringungen*, der Protokolle betrachtet. Dabei verwenden nicht alle Fahrzeuge dasselbe VRGS, sondern nur gewisse Teilmengen. Dadurch lassen sich zwei Szenarien modellieren:

1. Kein Protokoll wird von allen Verkehrsteilnehmern verwendet werden, es wird immer eine Marktsegmentierung geben.

2. Nicht alle Fahrer werden sich an die Routingempfehlungen halten.

Eine typische Aufteilung könnte bspw. sein, dass 30% der Fahrzeuge mit BeeJamA ausgestattet sind, die übrigen 70% mit einem DynLCP-Protokoll.

Die Aktualisierung des Routinggraphs erfolgt in der Form, dass regelmäßig die Transitzeiten übernommen werden. Jedes dynamische Protokoll registriert dazu das gewünschte Aktualisierungsinterval. Wird z.B. das DynLCP-Protokoll mit 15 und 30 minütiger Aktualisierung in einer Simulation verwendet, werden global alle 15 respektive 30 Minuten die Links des Routinggraphs aktualisiert. Für einen Link des Routinggraphen werden dazu die Transitzeiten der zugehörigen Links des physikalischen Graphen aufsummiert und als Gewicht im Routinggraph gespeichert. Für jedes Aktualisierungsintervall wird das ermittelte Gewicht separat gespeichert, so dass kein Protokoll neuere Informationen bekommen kann, als das Aktualisierungsintervall es zulässt.

Automaton 2 A^*-Algorithmus

```
1: function ASTAR(src, dst)
2:     closed ← ∅
3:     open ← {src}
4:     came_from ← []
5:     g[src] ← 0
6:     f[src] ← g[src] + h(src, dst)
7:     while open ≠ ∅ do
8:         i ← arg min_{o∈open} f[o]
9:         if i = dst then
10:            return path(came_from, dst)
11:        open← open \{i}
12:        closed ← closed ∪{i}
13:        for s ∈ neighbors(i, came_from[len(came_from) − 1]) do
14:            g_tmp ← g[i] + ω_is
15:            if s ∈ closed then
16:                if g_tmp ≥ g[s] then
17:                    continue
18:            if s ∉ open ∨ g_tmp < g[s] then
19:                came_from[s] ← i
20:                g[i] ← g_tmp
21:                f[s] ← g[s] + h(s, dst)
22:                if s ∉ open then
23:                    open ← open ∪{s}
24:    return "no path found"
25: end function
26: function PATH(came_from, current_node)
27:    if current_node ∈ came_from then
28:        p ← path(came_from, came_from[current_node])
29:        return p ∪ {current_node}
30:    else
31:        return current_node
32: end function
```

3.5. Vergleich von MATSim und Aimsun

Im Folgenden seien die beiden Simulatoren MATSim und Aimsun verglichen. Als zugrundeliegendes Straßennetz wurde ein Grid mit 625 Knoten gewählt, da in einem Grid mithilfe von Vorfahrtsschildern sichergestellt werden konnte, dass Aimsun keine Deadlocks erzeugt – zumindest nicht zu früh. Gänzlich vermeiden lässt es sich scheinbar nicht, denn bei hohen Verkehrsaufkommen kann reproduzierbar eine Deadlock-Situation erzeugt werden. In Abbildung 3.12 sind die Auswirkung einer steigenden Anzahl von Fahrzeugen auf die durchschnittliche Fahrzeit dargestellt. Verwendung finden das LCP und das DynLCP 15min Protokoll bei 100% Durchdringung. Unter MATSim werden die Protokolle auch jeweils einmal mit „wartend“ gekennzeichnet. Diese Unterscheidung bezieht sich auf die Bestimmung der Fahrzeit.

Automaton 3 LCP-Protokolle

```
1: function H(src, dst)
2:     return √((src.x − dst.x)² + (src.y − dst.y)²)/v_max
3: end function
4: function NEIGHBORS(n, l)
5:     return S_n ∩ AllowedTurnings_{n,l}
6: end function
7: function LCP(token)
8:     return astar(token.pos, token.dst)
9: end function
10: function DynLCP(token)
11:     if token.cachedPath = ∅ ∨ token.lastUpdate - t_now ≥ updateInterval then
12:         token.cachedPath ← LCP(token)
13:         token.lastUpdate ← t_now
14:     nextHop ← token.cachedPath[0]
15:     token.cachedPath← token.cachedPath[1]
16:     return nextHop
17: end function
```

MATSim lässt Fahrzeuge nur auf Startlinks auffahren, wenn genügend Platz ist. In Stausituationen kann es sehr lange dauern bis solch eine Situation auftritt – im Gegensatz zur Realität, wo üblicherweise ein Einfädeln möglich ist. Daher werden bei MATSim-Simulationen mittels des GRFs als Fahrzeit nur die Zeit ab dem Starten gewertet, nicht die Wartezeit bis zur initialen Auffahrt. Aimsun hingegen bezieht die Wartezeit ein. Um die Fahrzeiten vergleichbarer zu machen, wurde bei den markierten Ergebnisse auch unter MATSim diese Wartezeit mit eingebunden. Wie zu erkennen, ergibt dies zwar Unterschiede, welche aber die strukturelle Diskrepanzen nicht verändern.

Durch die dargestellten Ergebnisse können zwei zentrale Feststellungen gemacht werden:

1. Die beiden Simulatoren zeigen deutliche Unterschiede in den Ergebnissen. So steigen die Fahrzeiten unter MATSim erst bei deutlich größerer Nachfrage an als unter Aimsun. Das deutlich komplexere Verkehrsmodell von Aimsun führt zu realistischerem Verhalten und dadurch zu längeren Fahrzeiten im Vergleich zu MATSim.

2. Erklärungsbedürftigerweise schneidet das DynLCP-Protokoll schlechter ab als das LCP-Protokoll. Denn intuitiv könnte erwartet werden, dass ein dynamisches Protokoll zu einer Verbesserung führt. Wäre dies nicht der Fall, böte ein dynamisches VRGS, entgegen aller Erfahrungen, keine Vorteile. Prinzipiell ist somit ersteinmal festzuhalten, dass es offensichtlich Eingabeinstanzen gibt, bei denen ein dynamisches Protokoll kontraproduktiv ist. Andererseits werden bei den späteren Simulationen auf realistischeren Straßennetzen als einem Grid solche Effekte nicht beobachtbar sein. Daher stellt sich die Frage, wo die Besonderheit des verwendeten Grid-Netzes liegt. Durch Nachverfolgung einzelner Fahrzeuge konnte festgestellt werden, dass eine Art *Pendeleffekt* eintritt. Der mittlere Bereich des Grids ist stets vollständig ausgelastet. Am Rand ergeben sich hin und wieder Lücken und die Fahrzeuge versuchen diese auszunutzen. Die Fahrzeuge, die als erstes an dieser Lücke ankommen, können diesen Vorteil auch nutzen. Die später Ankommenden hingegen geraten in einen Stau. Zeitgleich sinkt die Auslastung an anderer Stelle, da

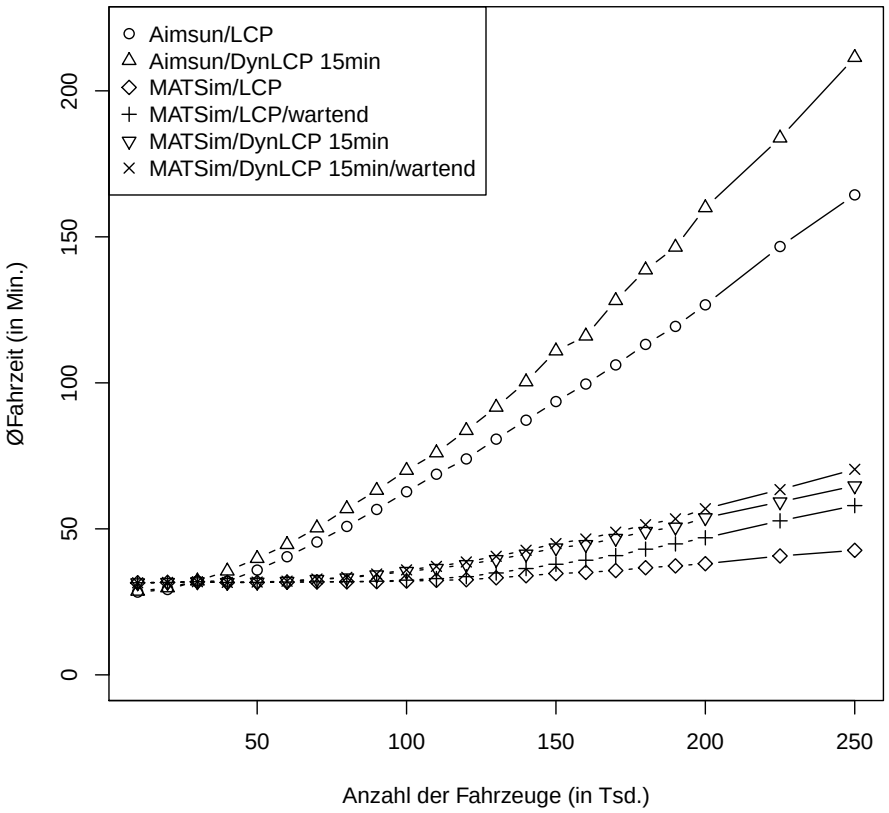


Abbildung 3.12.: Vergleich zwischen MATSim und Aimsun

viele Fahrzeuge versuchen die Lücke zu erreichen. Es entsteht demnach an anderer Stelle eine Lücke. Die im Stau stehenden Fahrzeuge versuchen dann wiederum diese neue Lücke zu erreichen. Dieser Prozess setzt sich im Zweifel mehrfach fort. Im Resultat erhöhten sich die zurückgelegten Fahrtstrecken erheblich und somit auch die Fahrzeiten. Die späteren Simulationen werden ergeben, dass auf realistischeren Netzen diese Effekte nicht auftreten. Vermutlich ist dieser Umstand, darin begründet, dass es dort eine größere Anzahl an Ausweichrouten gibt.

Für die Interpretation der späteren Simulationen ist besonders der erste Punkt relevant. Denn daraus kann gefolgert werden, dass in realistischen Situationen weniger Nachfrage bei einer gegebenen durchschnittlichen Fahrzeitobergrenze geroutet werden kann. So können mit MATSim selbst 250.000 Fahrzeuge mit einer maximalen durchschnittlichen Fahrzeit von 50 Minuten geroutet werden, mit Aimsun aber gerade mal ca. 100.000 Fahrzeuge bei dieser Eingabeinstanz. In der Realität (ohne Simulator), können also vermutlich weniger Fahrzeuge geroutet werden, als die späteren MATSim-Ergebnisse andeuten.

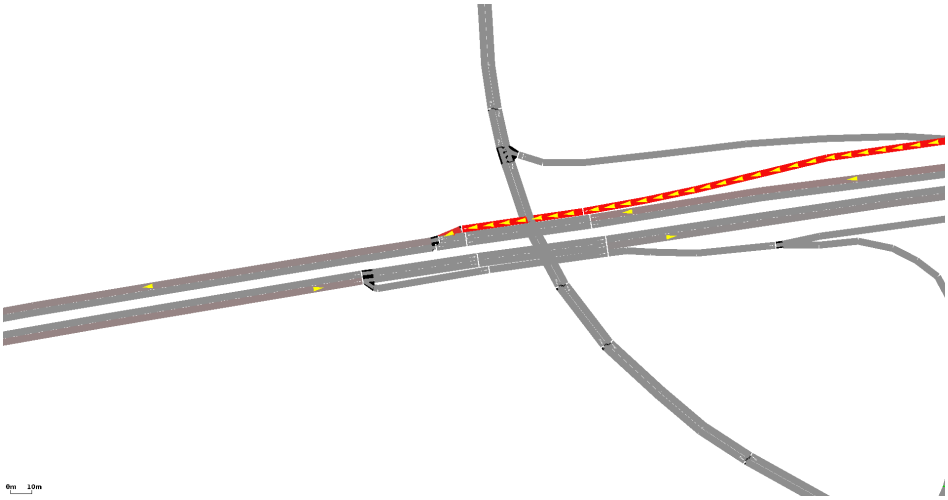


Abbildung 3.13.: Beispiel, indem Fahrzeuge auf einer Autobahn nicht die rechte Spur vor Auffahrten freigeben, wodurch sich ein Rückstau auf der Auffahrt bildet (eine genauere Beschreibung findet sich im Fließtext).

3.6. Diskussion

Das GRF erleichtert die Entwicklung von Protokollen durchaus, jedoch ist die Einsetzbarkeit erheblich eingeschränkt, solange Fahrzeugfolgemodell-Simulatoren derartige Probleme aufweisen mit einer größeren Anzahl von Fahrzeugen umzugehen. Ursprünglich sollte MATSim im Rahmen dieser Arbeit dazu verwendet werden, erste Ergebnisse zu erzielen. Später sollten weitere, präzisere, Simulatoren eingebunden werden, um vergleichende Ergebnisse zu erhalten. Schlussendlich stellten sich die getesteten, übrigen beiden Simulatoren zum gegenwärtigen Entwicklungszeit als unbrauchbar für den Zweck dieser Arbeit heraus, zumindest bei der Verwendung großer, automatisch importierter Netze.

Die Entwickler der Simulatoren empfehlen, die Verkehrsdichte niedriger zu wählen, so dass die Problematiken in Bezug auf gestaute Kreuzungen und Straßen erst gar nicht auftreten. Anders ausgedrückt: In gestauten Zuständen, und genau darum geht es in dieser Arbeit, sind die eingesetzten Verkehrsmodelle eine schlechte Approximation menschlicher Fahrer. Neben Deadlocks ist ein grundsätzliches Problem die mangelnde Vorausschau. Die Verkehrsmodelle beachten zwar vorausfahrende Fahrzeuge, mitunter auch über Linkgrenzen hinweg, jedoch werden Kontextinformationen in Bezug auf „Kreuzungen“ nicht immer mit eingebunden. So wird eine normale, innerstädtische Kreuzung mitunter nicht anders als eine Autobahnauffahrt behandelt. Für den menschlichen Fahrer bestünde jedoch ein enormer Unterschied – für eingesetzten Verkehrsmodelle aber nicht zwingend. Abbildung 3.13 zeigt, wie die Fahrzeuge auf der Autobahn nicht die linke Spur durch einen Spurwechsel freigeben und sich dadurch ein Rückstau auf der Autobahnauffahrt entwickelt. Die von rechts nach links fahrenden Fahrzeuge auf der Autobahn nehmen die Fahrzeuge auf der Auffahrt nicht wahr, weswegen für sie kein Anlass zum Spurwechseln besteht. Die Fahrzeuge auf der Auffahrt beschleunigen aber scheinbar auch nicht ausreichend und Sicherheitsabstände werden (korrekterweise) ebenfalls niemals unterlaufen. Menschliche Fahrer reagieren hingegen viel flexibler und vorausschauender. So wird eventuell etwas abgebremst oder stärker beschleunigt, um eine sich dadurch eine bietende (vielleicht auch nicht immer den Vorschriften entsprechend große) Lücke nutzen zu können.

Gelingt hingegen nur einem einzigen simulierten Fahrzeug nicht sofort die Auffahrt, bremst dies zum Stillstand ab und wartet, bis eine genügend große Lücke auftritt. Da das Fahrzeug aber sodann aus dem Stand wieder heraus beschleunigen muss, wird schlechterdings eine besonders lange Lücke benötigt, bevor das Fahrzeug schließlich auf die Autobahn wechseln kann. Da die einen keinen Platz machen und die anderen dadurch noch größere Lücken brauchen, ist bei entsprechendem Verkehr nahezu keine Auffahrtschance mehr gegeben. So entstehen an vielen Stellen im Netz, an denen Straßen unterschiedlicher Priorität zusammenstoßen, Rückstauungen. Mancherorts kann mit Ampeln und Schildern (sofern sie denn, wie intuitiv erwartet, funktionieren würden) nachgeholfen werden, auf Autobahnen ist ein Abbremsen des fließenden Verkehrs aber nicht realistisch.

Bis Fahrzeugfolgemodelle große Verkehrsnetze realistisch simulieren können, muss daher auf Behelfslösungen wie Queue-basierte, mikroskopische Systeme zurückgegriffen werden.

Kapitel 4

Verkehrsplanerische Routenwahl

Eine zentrale Aufgabe der Verkehrsplanung ist der Entwurf neuer oder die Erweiterung bestehender Straßeninfrastruktur [36]. Die ausreichende Dimensionierung dieser Infrastruktur und die Abschätzung der Konsequenzen für die bestehende Verkehrsnachfrage sind relevante Fragestellungen in diesem Kontext, die i.d.R. zunächst simulativ erörtert werden. Dazu werden Techniken benötigt, um möglichst realistische Routenwahl der Verkehrsteilnehmer aus einer gegebenen Verkehrsnachfrage abzuleiten. Nur so kann eine belastbare Einschätzung erfolgen, welche Auswirkungen geplante infrastrukturelle Änderungen im späteren Realeinsatz tatsächlich zeitigen. Das Ziel möglichst realistischer Verkehrsströme wird angegangen, indem zunächst *Optimalitätskriterien* definiert werden. Mittels geeigneter (zentraler) Optimierungsverfahren werden sodann Verkehrsströme gesucht, die optimal bzgl. dieser Kriterien sind. Die Fragestellung, welche Kriterien zu den realistischsten Ergebnissen führen, ist Gegenstand der Verkehrsplanung, für diese Arbeit aber nicht weiter von Bedeutung. Da hier das Ziel einer fahrzeitminimalen Routenwahl verfolgt wird (und nicht das einer realistischen, die real beobachtbaren Ströme erklärende), reicht es bereits die klassischen Kriterien basierend auf dem Konzept günstigster Pfade zu betrachten. Solche Kriterien können zwar nur approximativ reale Verkehrsströme modellieren, aber es besteht, wie gezeigt wird, ein enger Zusammenhang zum Ziel der Fahrzeitreduzierung eines VRGS. Zwar können die im Folgenden vorgestellten Techniken nicht Eins zu Eins in ein verteiltes VRGS umgesetzt werden, handelt es sich bei der Verkehrsplanung doch um ein Offline-Konzept, wohingegen ein VRGS prinzipbedingt ein Online-Verfahren ist. Dafür zeigt die folgende Diskussion, was im Straßenverkehr potentiell als „optimal“ angesehen werden kann, erlaubt eine kritische Diskussion des BeeJamA-Protokolls und ermöglicht im späteren Verlauf konstruktive Verbesserungsansätze für BeeJamA aufzuzeigen.

Ausgangspunkt für die verkehrsplanerische Routenwahl ist eine häufig empirisch bestimmte Verkehrsnachfrage in Form von OD-Paaren. Die folgende, auch als *Traffic Assignment Problem* (TAP) [26] bezeichnete, *Routenwahl* dient der Festlegung von möglichst realistischen Routen unter Berücksichtigung der restlichen, ebenfalls Nachfrage auf den beschränkten Kapazitäten induzierender, OD-Paare. Dazu werden Verhaltensannahmen der Fahrer getroffen, die eine realitätsnahe Verteilung der Fahrer auf die möglichen Routen ermöglichen sollen.

Die einfachste (und historisch zu erst betrachtete) Routenwahl, besteht in der LCP-Zuweisung bzgl. der FFT [20]: Jedes Fahrzeug nutzt genau die Route mit den minimalen FFT-Transitkosten. Da alle Fahrzeuge mit demselben OD-Paar auch dieselbe, günstigste Route

(bzgl. der FFT) nutzen würden und alle anderen Routen ungenutzt blieben, wird dieses Prinzip auch *All-or-Nothing*-Zuweisung genannt. In der Realität hingegen würden die Fahrzeuge sich gemäß der sich tatsächlich einstellenden Kosten auf verschiedene Routen aufteilen (zumindest sofern der Fluss über die günstigste Route zu groß wird). Diese simpelste Vorgehensweise kann demnach nicht als sonderlich realistisch, noch als sinnvoll im Sinne möglichst kurzer Fahrzeiten, bewertet werden.

Neben der All-or-Nothing-Zuweisung wurden weitere Optimalitätskriterien, sowie Verfahren zur Erreichung selbiger in den letzten Jahrzehnten eingeführt. Mit dem User Equilibrium und dem System Optimum werden in diesem Kapitel die beiden klassischen Konzepte [10, 152] in diesem Kontext erörtert und mit dem Frank-Wolfe-Algorithmus [50] das bekannteste Lösungsverfahren vorgestellt.

Da die zuletzt genannten Optimalitätskriterien, wie gezeigt wird, dem Grundgedanken günstigster Routen entsprechen, gibt es eine enge Verknüpfung zum Routing. Angenommen zu einen bestimmten Zeitpunkt wäre die aktuelle Verkehrsnachfrage bekannt. So könnte eine TAP-Lösung bestimmt werden und Fahrzeuge damit prinzipiell auch online geroutet werden. Nur verlöre die Lösung, aufgrund der hohen Dynamik des offen Systems, schnell ihre Optimalität und müsste kontinuierlich für die geänderte Situation neu bestimmt werden. Für Netze realistischer Größe ist diese zentrale Optimierung, aufgrund der zeitlichen Nebenbedingungen, jedoch nicht mehr möglich, wie nachstehend erläutert wird.

Für dieses Kapitel seien zunächst die folgenden, in diesem Kontext üblichen, Annahmen getätigt:

Annahme 4.1. Jeder Fahrer ist egoistisch und möchte den günstigsten Pfad nutzen.

Es gibt viele Gründe, eine Route gegenüber einer anderen zu bevorzugen. Nicht immer spielt nur die Fahrzeit eine ausschlaggebende Rolle. So kann eventuell die Ästhetik der Route ein weiteres Kriterium sein. Aber es liegen empirische Untersuchungen vor, die die Fahrzeit (wenig überraschend) als das wichtigste Kriterium einer Route für menschliche Fahrer hervorheben [42]. Diese Verhaltensannahme erscheint somit als vertretbar, ist der Wunsch von freiwilligen, signifikanten Umwegen sicherlich nur bei einem marginalen Anteil von Verkehrsteilnehmern verbreitet. Ein auf dieser Annahme basierendes Routing wird auch als *Selfish-Routing* bezeichnet. Es wird gezeigt, dass die Gesamtfahrzeit im Allgemeinen nicht minimal sein kann, wenn ausschließlich solche egoistische Entscheidungen getroffen werden.

Annahme 4.2. Jeder Fahrer hat vollständige, globale Information über den Graphen bzgl. Topologie und Linkkosten.

Wenngleich der Bezug topologischer Informationen des (als mehr oder minder) statisch angenommenen Straßennetzes nicht problematisch ist, sind hingegen globale, stets aktuelle Linkkosten unter realistischen Umständen nicht verfügbar. Die Nichtverfügbarkeit ist ja gerade die Motivation verteilter Ansätze. Für globale Optimierungsansätze ist diese Annahme aber unerlässlich. Für die Einsatzzwecke des TAP zur Offline-Routenwahl stellt diese Annahme auch kein Problem dar, bei einem etwaigen (und zuvor bereits angedeuteten) Einsatz im VRGS-Kontext hingegen schon.

Annahme 4.3. Die gesamte Verkehrsnachfrage ist *a priori* bekannt.

Wie bei der vorherigen Annahme gilt, dass diese Information nur in einem vollständig kontrollierten Offline-Szenario vorliegen. Die Optimierungsansätze benötigen eine vollständige

Übersicht der zukünftig auftretenden Nachfrage. In einem VRGS-Kontext ist diese Annahme sicherlich nicht vollumfänglich erfüllbar, da es jedem Fahrer frei steht, beliebig über Fahrtantritt und -ziel zu entscheiden.

Annahme 4.4. Fahrzeiten auf einem Link werden durch eine Link Performance Funktion (LPF) beschrieben.

Die Transitzeit eines Fahrzeugs auf einem Link ist dadurch exakt durch die LPF beschrieben, wodurch unvorhergesehene Ereignisse (wie Unfälle) oder zeitliche Verluste durch Trödeln (im Sinne des Nagel-Schreckenberg Verkehrsmodells [100]) ausgeschlossen sind.

Es handelt sich bei den genannten Annahmen, ganz offensichtlich um theoretische. In der Realität muss ein VRGS robust und flexibel genug sein, um mit Verstößen gegen diese Annahmen umzugehen. Für die folgende Erörterung unterer Fahrzeitschranken sind sie jedoch zunächst vonnöten.

Es werden nacheinander die folgenden drei, stets realistischer werdende, TAP-Varianten betrachtet:

Flussunabhängig und statisch: Die Kosten eines Links hängen nicht von dem Fluss über selbige ab. Aufgrund tagtäglicher Erfahrung und der Diskussion der BPR-Funktion ist das als unrealistisch zu bewerten (vgl. Abschnitt 3.3.1). Für die zu minimierende Zielfunktion resultiert dies jedoch in dem erheblichen Vorteil der Linearität, womit ein Lineares Programm zur Lösung eingesetzt werden kann.

Flussabhängig und statisch: Dieser Fall erweitert den vorherigen um Kostenfunktionen, die vom Fluss abhängen. Für die Zielfunktion ergibt sich daraus im Allgemeinen Nicht-Linearität. Die BPR-Funktion zeigt, dass eine LPF (und damit die TAP-Zielfunktion) aber zumindest konvex modelliert werden kann. Dadurch können spezielle Optimierungverfahren genutzt werden. Hier wird der Frank-Wolfe-Algorithmus erläutert, der das konvexe Problem auf den zuvor genannten linearen Fall zurückführt.

Flussabhängig und dynamisch: Die nächste Ergänzung fügt den temporalen Aspekt hinzu: ein Fluss belegt eine Kante nur noch für eine bestimmte Zeit und nicht permanent. Der vorherige, flussabhängige und statische Fall wird zur Routenwahl für Rush Hour-Situationen eingesetzt, da dort charakteristischerweise stets weitere Fahrzeuge nachfließen und der einzelne Linkfluss und die Linkkosten daher approximativ als statisch angesehen werden können. Diese dynamische Variante hingegen berücksichtigt die zeitliche Flussänderung eines Links.

Eine ausführliche Monographie zum ersten Fall findet sich etwa in [2], für den zweiten in [107] und für den dritten in [114].

Zunächst seien die beiden statischen Fälle beschrieben.

4.1. Flussunabhängige Routenwahl

Sei $G = (V, L)$ ein Graph. Folgende Begriffe werden benötigt:

1. Ein *Linkfluss* ist eine Funktion $\phi\colon L \to \mathbb{R}$, die jedem Link eine Belegung zuordnet, d.h. im Kontext des Straßenverkehrs eine (kontinuierliche) Anzahl von Fahrzeugen.

2. Als *Fluss* sei das Tupel $\Phi = (\phi_1, \ldots, \phi_{|L|})$ der Linkflüsse in G bezeichnet, wobei jedem Link $l \in L$ genau ein Element $\Phi_i, 1 \leq i \leq |L|$ zugeordnet ist und umgekehrt. Im Folgenden wird mit ϕ_l genau das Element Φ_i bezeichnet, das Link l zugeordnet ist. Sei ferner $\mathbf{F} := \mathbb{R}^{|L|}$ die Menge aller möglichen Flüsse in G.

3. Für Links gelte ein (oberes Fluss-) *Kapazitätslimit* u: $L \to \mathbb{R}_+$.

4. Mit jedem Knoten sei ein *Bedarf*, λ: $V \to \mathbb{R}$, verknüpft. Knoten $o \in O \subseteq V$ besitzen ein (Über-)Angebot, $\lambda(o) < 0$, Knoten $d \in D \subseteq V$ eine Nachfrage, $\lambda(d) > 0$, und alle übrigen Knoten $v \in V \setminus (O \cup D)$ sind ausgeglichen, $\lambda(v) = 0$. Für G gelte eine Balance zwischen Angebot und Nachfrage, sodass jedes Fahrzeug ein Ziel hat:

$$\text{Netzgleichgewicht:} \sum_{v \in V} \lambda(v) = 0 \tag{4.1}$$

5. Für Links gelten die (Transit-)*Kosten* τ: $L \to \mathbb{R}_+$.

Statt $u(l), \tau(l), \lambda(v)$ wird im Folgenden auch u_l, τ_l, λ_v geschrieben.

Das grundlegende Problem der Kostenminimierung bei gegebenem Bedarf wird als *Minimum Cost Flow Problem* (MCFP) bezeichnet und wird klassischerweise als Lineares Programm (LP) [27] dargestellt:

$$\check{\Phi}^* = \min_{\Phi \in F} \quad T(\Phi) = \sum_{l \in L} \phi_l \tau_l \tag{4.2a}$$

unter den Nebenbedingungen

$$\sum_{(v,v') \in L} \phi_{(v,v')} - \sum_{(v',v) \in L} \phi_{(v',v)} = \lambda_v \qquad \forall v \in V \tag{4.2b}$$

$$0 \leq \phi_l \leq u_l \qquad \forall l \in L \tag{4.2c}$$

Die erste Nebenbedingung fordert, dass die Ein- und Ausflüsse an einem Knoten dem Bedarf entsprechen. Die Zweite fordert, dass die Kapazitätsbeschränkungen eingehalten werden. Das LP ist nach unten beschränkt, da die minimalen Flusskosten nicht negativ werden können.

Für ein beschränktes LP existiert genau dann eine eindeutige Lösung, wenn die Nebenbedingungen widerspruchsfrei sind (und somit mindestens eine Lösung existiert) [25, Theorem 29.13]. Da ist hier dann der Fall, wenn das Kapazitätslimit der Links ausreicht, um den Bedarf zu befriedigen. LPs lassen sich mittels Interior-Point-Methoden und dem Ellipsoid-Verfahren prinzipiell in Polynomialzeit lösen, jedoch ist in der Praxis das weitverbreitete Simplex-Verfahren häufig effizienter, auch wenn im Worst Case nur exponentielle Laufzeit garantiert werden kann. Neben allgemeinen Optimierungsansätzen existieren auch effiziente direkte Algorithmen für das MFCP.

Proposition 4.5. Die Spezialisierung des MCFP mit unbegrenzter Kantenkapazität, $u_l = \infty, \forall l \in L$, einer Quelle und einer Senke, entspricht dem LCP-Problem (vgl. Abschnitt 1.5.1), da der gesamte Fluss für das OD-Paar über denselben, günstigsten Pfad geleitet werden kann. Es handelt sich daher um eine All-or-Nothing-Zuweisung.

Somit kann diese Spezialisierung statt mittels eines LP auch durch Anwendung eines LCP-Algorithmus gelöst werden.

Das MCFP kennt keine ausgezeichneten Start-/Zielpaare. Anders ausgedrückt, es ist irrelevant, zu welchem Zielknoten j mit $\lambda_j > 0$ der Fluss von einem Knoten i mit $\lambda_i > 0$ fließt. Sog. *Multi commodity* Flüsse ergänzen die Möglichkeit der Modellierung von OD-Paaren[1]. Die folgenden Begriffe werden zusätzlich benötigt:

- Das Paar $(o, d) \in K := V \times V$ heißt OD-Paar (vgl. Anschnitt 3.1). Sei P_k die Menge aller möglichen Pfade für das OD-Paar $k = (o, d)$. Der Bedarf λ_v des MCFP wird durch λ_{od} ersetzt und gibt die Anzahl der an o startenden Fahrzeuge mit Ziel d an.
- Sei $\mathbf{K}$ die Menge aller Multi Commdity-Linkflüsse mit

$$\mathbf{K} = \underbrace{\mathbb{R}^{|K|} \times \ldots \times \mathbb{R}^{|K|}}_{|L|-\text{viele}},$$

  d.h. pro Link wird nicht mehr ein Linkfluss, sondern pro OD-Paar genau einer, gesucht. Bezeichne ϕ_l^k den Eintrag von $\Phi \in \mathbf{K}$ für OD-Paar k und Link l.

Das LP für das Multi Commodity-MFCP (MC-MFCP) ist dann gegeben durch:

$$\check{\Phi}^*_{MC} = \min_{\Phi \in \mathbf{K}} \quad T(\Phi) = \sum_{k \in K} \sum_{l \in L} \phi_l^k \tau_l \tag{4.3a}$$

unter den Nebenbedingungen

$$\sum_{(v,d) \in L} \phi^k_{(v,d)} - \sum_{(d,v) \in L} \phi^k_{(d,v)} = \lambda_k \qquad \forall k \in K, v \in V \tag{4.3b}$$

$$\sum_{k \in K} \phi_l^k \leq u_l \qquad \forall l \in L \tag{4.3c}$$

$$\phi_l^k \geq 0 \qquad \forall l \in L \tag{4.3d}$$

Die erste Nebenbedingung erzwingt einen Fluss zur Befriedigung der Nachfrage, die beiden weiteren sichern wieder die Einhaltung der Kapazitätsschranken.

Proposition 4.6. Die Spezialisierung des MC-MCFP mit unbegrenzter Kapazität $u_l = \infty, \forall l \in L$, lässt sich durch einem LCP-Algorithmus lösen, da gemäß Proposition 4.5 für jedes OD-Paar (o, d) mit λ_{od} nur der LCP bestimmt werden muss.

Allerdings ist es von erheblichen Nachteil, dass das MCFP flussunabhängig ist und eine statische Situation beschreibt.

4.2. Flussabhängige Routenwahl

Wird die Einschränkung der Flussunabhängigkeit aufgehoben, ist der Fluss ϕ_l ein Parameter der Kostenfunktion. Dadurch ist die Zielfunktion $\phi_l \tau_l(\phi_l)$ im Allgemeinen aber nicht mehr linear.

[1] Historisch stammt der Begriff „Commodity“ aus der Transportlogistik und der Aufgabenstellung die Nachfrage nach unterschiedlichen Gütern zu befriedigen. Ein Gut entspricht hier einem OD-Paar.

Wie in Proposition 3.4 zur BPR-Funktion gezeigt, kann aber ein realistisches $\tau(\phi_l)$ zumindest als konvexe Funktion modelliert werden. Dadurch ist auch die Zielfunktion konvex und es ist gesichert, dass lokale und globale Optima identisch sind. Für diese Voraussetzungen existieren bekannte Optimierungsverfahren, bspw. der Frank-Wolfe-Algorithmus, der im Zusammenhang des TAP häufig Verwendung findet. Eine konvexe Zielfunktion bedeutet allerdings auch, dass die Routenwahl eines Fahrers ggf. die Entscheidung anderer Fahrer beeinflusst, da jedes Fahrzeug (im Gegensatz zum linearen Fall) die Kosten des Links erhöht.

Aufbauend auf dieser Feststellung formulierte Wardrop 1952 seine beiden bekannten, deklarativen Prinzipien [152]:

Erstes Wardrop Prinzip: „The journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route."

Zweites Wardrop Prinzip: „The average journey time is a minimum."

Das erste Prinzip beschreibt ein Nutzergleichgewicht (*User Equlibrium*, UE) bei dem ein Fahrer unilateral keine weitere Minimierung seiner Fahrzeit erreichen kann. Formal ist das UE äquivalent zum Nash-Gleichgewicht [101, 102] (welches von Nash bereits 1950 vorgestellt wurde). Das zweite Prinzip beschreibt eine Pfadwahl, bei der hingegen die Gesamtfahrzeit aller Fahrer nicht weiter minimiert werden kann (*System Optimum*, SO).

Das UE-Konzept modelliert die Pfadwahl aus der Sicht eines einzelnen, unkooperativen Fahrers, der die für ihn günstigste Route (Annahme 4.1) unter Berücksichtigung der Pfadwahl der übrigen Fahrer verwenden möchte. Bestimmt wird ein UE i.d.R. durch iterative Verfahren, der weiter unten vorgestellte Frank-Wolfe-Algorithmus fällt darunter. Liegt ein UE vor, ist die charakterisierende Eigenschaft, dass mit der Änderung der Route eines beliebigen Fahrers auch eine Verschlechterung seiner Reisezeit einhergeht.

Das SO hingegen geht von einer zentralen Kontrollinstanz aus, welche mittels globaler Information einen Fluss entwerfen kann, bei der die Gesamtfahrzeit minimiert wird. In der Realität müssten sich hierzu alle Fahrer kooperativ bzgl. ihrer Routenwahl absprechen, ein unrealistisches Szenario.

Beispiel (Unterschied UE und SO). Abbildung 4.1 verdeutlicht die Unterschiede zwischen UE und SO. Angenommen, drei Fahrzeuge möchten gleichzeitig von Knoten a zu Knoten c. Der Pfad $a \to b \to c$ hat fixe Kosten von 10 Minuten, der direkte Pfad $a \to c$ ist quadratisch abhängig vom Fluss. Im UE wählen alle drei Fahrzeuge den direkten Pfad und benötigen hierzu jeweils 9 Minuten, insgesamt werden 27 Minuten Gesamtfahrzeit benötigt. Der längere Pfad würde nicht gewählt, da die benötigte Zeit um 1 Minute länger wäre. Nähme aber ein Fahrzeug diesen Umweg in Kauf, benötigten die beiden anderen Fahrzeuge auf dem direkten Pfad $a \to c$ jeweils nur noch 4 Minuten. Die Gesamtfahrzeit betrüge $2 \cdot 4min + 1 \cdot 10min = 18min$. Wechselte noch ein Fahrzeug auf den Pfad $a \to b \to c$, stiege die Gesamtfahrzeit als auch die Fahrzeit des zusätzlich Wechselnden wieder. Die Gesamtfahrzeit im SO beträgt damit 18 Minuten bzw. 27 Minuten im UE.

In dem Fall von $\lambda_{ac} = 3$ ist die FFT-LCP-Zuweisung äquivalent zum UE. Wie oben bereits angemerkt, gilt das im Allgemeinen für eine All-or-Nothing-Zuweisung keineswegs. Sei $\lambda_{ac} = 4$. Bei einer vollständigen Zuweisung auf Pfad $a \to c$ ergeben sich Kosten von $4 \cdot 16min = 64min$. Das UE ergibt sich hingegen, wenn ein Fahrzeug auf den oberen Pfad ausweicht, mit Gesamtkosten von $1 \cdot 10min + 3 \cdot 9min = 37min$. Ein zweites Fahrzeug (mit egoistischem Fahrer) würde nicht auf den oberen Pfad wechseln, da sich die individuelle Fahrzeit von $9min$

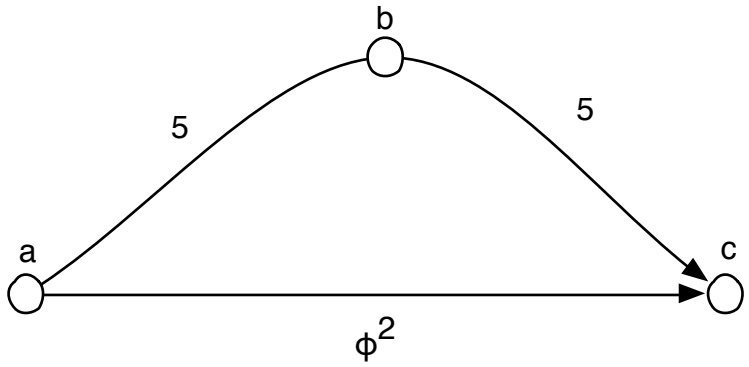


Abbildung 4.1.: Unterschied zwischen UE und SO.

auf $10min$ verschlechtern würde, obwohl Gesamtkosten von $2 \cdot 10min + 2 \cdot 4min = 28min$ einträten. Das stellt gleichzeitig das SO dar, da bei einem weiteren Wechsel Gesamtkosten von $31min$ entstünden. Um das SO zu erreichen, müssten die Fahrer aber Kommunizieren und teilweise suboptimale Routen akzeptieren. □

Ein SO mag global betrachtet demnach sinnvoller sein, realistischer aufgrund des zu erwartenden Egoismus und der nicht stattfindenden Kooperation ist jedoch ein UE, welches in Reinform allerdings kaum auftreten wird. Die UE/SO-Flüsse wurden in diesem Beispiel intuitiv ermittelt, das weiter unten vorgestellte Frank-Wolfe-Verfahren löst dieses Problem allgemein.

In der Verkehrsplanung wird ein UE klassischerweise verwendet, um eine realistische Routenwahl zu modellieren. Ausgehend von einer Verkehrsnachfrage wird ein UE bestimmt und erst mit dieser vermeintlich realistischen Routenwahl die zu evaluierende Infrastruktur simulativ erprobt. Motiviert wird die Realitätsbehauptung dadurch, dass Fahrer, die regelmäßig eine bestimmte Strecke fahren (Pendler in einem sogenannten Day-to-Day-Szenario), mit der Zeit den für sie schnellsten Weg unter Berücksichtigung der durch die anderen Verkehrsteilnehmer induzierten Verzögerungen durch Trial-and-Error lernen und sich schlussendlich nach einigen Iterationen (Tagen) im UE befinden.

Die Annahme, dass Fahrer immer versucht sind, die ihren Eigennutzen zu maximierende Route zu wählen, ist als theoretische Annahme sicherlich gerechtfertigt und beschreibt einen Zustand, in dem kein Fahrer freiwillig Einbußen hinnehmen muss. In der Realität sind solche Gleichgewichte offensichtlich nicht omnipräsent und können sich höchstens angenähert einstellen. Denn zum Einen sind nicht alle Fahrer ortskundig (oder es stellen sich unerwartete Staus ein) und können somit die Entscheidungen der anderen Fahrer nicht mit beliebig hoher Genauigkeit antizipieren. D.h. welche Route die für sie aktuell beste ist, ist ihnen nicht bekannt. In empirischen Untersuchungen [40, 97] konnte aber nachgewiesen werden, dass in der Realität Flüsse beobachtet werden können, die näher am UE als am SO sind.

Auch wenn ein UE die menschliche Routenwahl nicht perfekt modelliert, ist es (inkl. der im Nachfolgenden geschilderten dynamischen Variante) die gängige Methode in der Verkehrsplanung zur Routenwahl. Aus Sicht des Routings von Fahrzeugen ist der UE-Fluss ebenfalls interessant, da ein abweichender Fluss mindestens für einen Teil der Fahrer längere Fahrzeiten bedeuten würde. So kann das UE durchaus als das Ziel eines VRGS angesehen werden. In der Tat basieren alle bekannten verteilte Ansätze auf einem Selfish Routing. Wie gezeigt wird, kann

ein UE allerdings unter bestimmten Voraussetzungen durch eine spezielle Maut auch in ein SO verwandelt werden, so dass dieselben Verfahren zur Erreichung beider Flüsse verwendet werden können. Jedoch sind die gegenwärtig vorgeschlagenen verteilten Ansätze weit davon entfernt, ein UE erreichen zu können. Im Folgenden werden UE und SO näher beschrieben, um damit die Möglichkeit für eine Diskussion der VRGS unter diesem Gesichtspunkt zu ebnen. Aufbauend auf dieser Beschreibung können im Anschluss der Darstellung des BeeJamA-Protokolls auch Erweiterungen vorgeschlagen werden, um dem Ziel einer verteilten Approximation des UE (bzw. SO) näher zu kommen.

4.2.1. Statische Pfadwahl

Die beiden zuvor genannten Konzepte werden üblicherweise mittels Pfadflüssen (statt wie zuvor mit Linkflüssen) formalisiert:

- Der *Pfadfluss* ϕ_p^k gibt die Belegung auf dem Pfad $p \in P_k$ für das OD-Paar $k \in K$ an. Aus gegebenen Pfadflüssen lässt sich ein Linkfluss bestimmen:

$$\phi_l = \sum_{k \in K} \sum_{p \in P_k} \phi_p^k \delta_{l,p}^k \qquad \forall l \in L, \tag{4.4}$$

$$\text{mit } \delta_{l,p}^k = \begin{cases} 1, & \text{falls } l \in p \\ 0, & \text{falls } l \notin p \end{cases} \qquad \forall l \in L \; \forall k, \in K, \forall p \in P_k \tag{4.5}$$

- Für Pfad p sind die *Pfadkosten* gegeben durch $\tau_p = \sum_{l \in p} \tau_l$.

Der Fluss $\Phi \in \mathbf{K}$ ist dann ein UE-Fluss gdw.

$$\phi_p^k \cdot \left(\tau_p - \min_{q \in P_k} \tau_q \right) = 0 \qquad \forall k \in K, p \in P_k \tag{4.6a}$$

$$\phi_p^k \geq 0 \qquad \forall k \in K, p \in P_k \tag{4.6b}$$

$$\sum_{p \in P_k} \phi_p^k = \lambda_k \qquad \forall k \in K \tag{4.6c}$$

Gleichung 4.6a ist eine direkte Umsetzung des ersten Wardrop-Prinzips: Der Fluss $o \overset{p}{\rightsquigarrow} d$ muss Null sein, es sei denn p weist minimale Kosten auf. Gleichung 4.6b fordert zudem wieder Bedarfsbefriedigung, allerdings ausgedrückt mittels Pfadflüssen. Die Gleichung 4.6c fordert nicht-negative Pfadflüsse (wodruch auch keine negativen Linkflüsse entstehen können). Die zusätzliche Forderung der Einhaltung eines Kapazitätslimits u_l entfällt bei der Verwendung konvexer Kostenfunktion, da höhere Linkflüsse zu immer höheren Kosten führen (im Gegensatz zu dem linearen Fall mit konstanten Linkkosten). Das Kapazitätslimit ist somit quasi implizit in der Kostenfunktion enthalten, denn wird der Linkfluss zu hoch, lohnt die Verwendung des Links nicht mehr.

Nach Beckmann [10] lässt sich die charakterisierende Darstellung in Gleichung 4.6 in das folgende, konvexe Minimierungsprogamm umwandeln:

$$\Phi^*_{UE} = \min_{\Phi \in \mathbf{K}} \quad T(\Phi) = \sum_{l \in L} \int_0^{\phi_l} \tau_l(\phi) d\phi \tag{4.7a}$$

unter den Nebenbedingungen

$$\phi_p^k \geq 0 \qquad \forall k \in K, p \in P_k \tag{4.7b}$$

$$\sum_{p \in P_k} \phi_p^k = \lambda_k \qquad \forall k \in K \tag{4.7c}$$

$$\sum_{k \in K} \sum_{p \in P_k} \phi_p^k \delta_{l,p}^k = \phi_l \qquad \forall l \in L \tag{4.7d}$$

Die ersten beiden Nebenbedingungen entsprechen den Gleichungen 4.6b und 4.6c der vorherigen, deklarativen UE-Charakterisierung. Die Dritte, in Gleichung 4.7d, koppelt die Link- an die Pfadflüsse, in der Art, dass eine Lösung $\Phi \in \mathbf{K}$ nur dann gültig ist, wenn die Linkflüsse exakt aus Pfadflüssen ableitbar sind.

Das Programm des SO-Flusses zur Minimierung der Gesamtfahrzeit lässt sich ohne Umwege direkt formulieren:

$$\Phi^*_{SO} = \min_{\Phi \in \mathbf{K}} \quad T(\Phi) = \sum_{l \in L} \phi_l \tau_l(\phi_l) \tag{4.8a}$$

unter den Nebenbedingungen

$$\phi_p^k \geq 0 \qquad \forall k \in K, p \in P_k \tag{4.8b}$$

$$\sum_{p \in P_k} \phi_p^k = d_k \qquad \forall k \in K \tag{4.8c}$$

$$\sum_{k \in K} \sum_{p \in P_k} \phi_p^k \delta_{l,p} = \phi_l \qquad \forall l \in L \tag{4.8d}$$

Die Nebenbedingungen entsprechen denen des UE-Flusses.

4.2.2. Zusammenhang zwischen UE und SO

Zwischen UE und SO bestehen zwei bemerkenswerte Zusammenhänge. Zum Einen ist das Gesamtkostenverhältnis, der sogenannte *Price of Anarchy*, beschränkt und zum Anderen lassen sich die beiden Flüsse mittels der sog. Beckmann-Transformation ineinander umwandeln.

Roughgarden und Taros [119] zeigten 2002, dass Selfish Routing nicht beliebig schlecht im Vergleich zu einem optimalen Routing ist. Bezeichne $c(UE)$ die aggregierten Kosten eines UE und analog $c(SO)$ die Kosten des SO. Beispielweise können die Kosten die (durchschnittliche) Gesamtfahrzeit repräsentieren. Der *Price of Anarchy* (PoA) bezeichnet das Verhältnis dieser beiden Größen und im Allgemeinen gilt:

$$PoA := \frac{c(UE)}{c(SO)} \leq 2 \tag{4.9}$$

Der PoA quantifiziert den Verlust des UE im Vergleich zum SO. Die Ungleichung gibt an, dass die Gesamtfahrzeit höchstens doppelt so hoch ist wie im SO, oder anders ausgedrückt, dass zu den Kosten des UE die doppelte Verkehrsnachfrage optimal geroutet werden kann. Zwar kann $c(UE)$ beliebig groß werden, die Ungleichung bleibt aber erhalten, d.h. $c(SO)$ wächst entsprechend. Von Schulz und Moses [123] wird dies bspw. ob der beschränkten Verschlechterung als nachträgliche Rechtfertigung des UE interpretiert. Die Idee dabei ist, dass das UE leichter, da im Gegensatz zum SO ohne Kooperation auskommend, zu erreichen ist und dennoch nicht eine „erhebliche“ Verschlechterung hinsichtlich der durchschnittlichen Fahrzeiten mit sich bringt. Die allermeisten Routingverfahren, so auch BeeJamA, basieren auf Selfish-Entscheidungen und wenngleich Online-Routing im Allgemeinen kein UE entstehen lassen kann, ist dennoch das egoistische Grundprinzip diesen Überlegungen zufolge gerechtfertigt.

Der zweite wichtige Zusammenhang besteht darin, dass durch gezielte Verteuerung von Links ein UE in ein SO transformiert werden kann [149]. Daher wird dieses Konzept in der Literatur häufig als Begründung für eine Maut-Erhebung genannt. Durch Maut kann demnach prinzipiell ein Anreiz geschaffen werden, dass Fahrer nicht den für sie persönlich besten Pfad wählen, sondern den, der insgesamt für alle Fahrer am besten ist. Die grundsätzliche Idee dabei ist, jedem Fahrzeug eine *Strafmaut* in Höhe der durch dieses Fahrzeug entstandene Verzögerung aufzuerlegen. Ein einzelnes, zusätzliches Fahrzeug auf einem Link erzeugt eine Verzögerung von $\tau_l'(\phi_l) = d\tau_l(\phi_l)/d\phi$. Da in den bisher betrachteten, statischen Flussproblemen, alle Fahrzeuge auf einer Kante die Verzögerung erfahren, erzeugt ein weiteres Fahrzeug eine Gesamtverzögerung von $\phi_l\tau_l'(\phi_l)$ auf dem Link l. Die Erhebung dieser Zusatzkosten wird als *Marginal Cost Pricing* (oder Congestion Pricing) bezeichnet. Durch Einsetzen der Gesamtkosten $\tau_l^{MCP}(\phi_l) = \tau_l(\phi_l) + \phi_l\tau_l'(\phi_l)$ in Gleichung 4.7a des UE, erhält man Gleichung 4.8a des SO:

$$\sum_{l\in L}\int_0^{\phi_l} \tau_l^{MCP}(\phi)d\phi = \sum_{l\in L}\int_0^{\phi_l}\left(\tau_l(\phi) + \phi\frac{d\tau_l(\phi)}{d\phi}\right)d\phi \tag{4.10}$$

$$= \sum_{l\in L}\left(\int_0^{\phi_l}\tau_l(\phi)d\phi + \int_0^{\phi_l}\phi\frac{d\tau_l(\phi)}{d\phi}d\phi\right) \tag{4.11}$$

$$= \sum_{l\in L}\left(\int_0^{\phi_l}\tau_l(\phi)d\phi + \left([\tau_l(\phi)\cdot\phi]_0^{\phi_l} - \int_0^{\phi_l}\tau_l(\phi)d\phi\right)\right) \tag{4.12}$$

$$= \sum_{l\in L}\phi_l\tau_l(\phi_l) \tag{4.13}$$

Proposition 4.7 (Beckmann Transformation)**.** Mittels Beckmann Transformation $\tau^{MCP}(\phi) = \tau(\phi) + \phi\tau'(\phi)$ der ursprünglichen Kosten $\tau(\phi)$ kann ein UE in ein SO umgewandelt werden, da ein UE bzgl. der Gesamtkosten (Transitzeiten + Maut) äquivalent zu einem SO bzgl. der Transitzeiten und umgekehrt ist. UE- und SO-Flüsse können daher mit demselben Algorithmus bestimmt werden.

Beispiel (Braess-Paradoxon). An einem Beispiel, dem sog. Braess-Paradoxon [17] von 1968, seien die bisherigen Erläuterungen diskutiert. Das Beispiel zeigt, dass der Neubau einer Straße negative Auswirkungen auf die Gesamtfahrzeit haben kann und dass im UE die Fahrzeiten aller Fahrer schlechter sein können als im SO. Das Braess-Paradoxon tritt in zufälligen Graphen [148], wie in realen Straßennetzen auf [14, 47, 79].

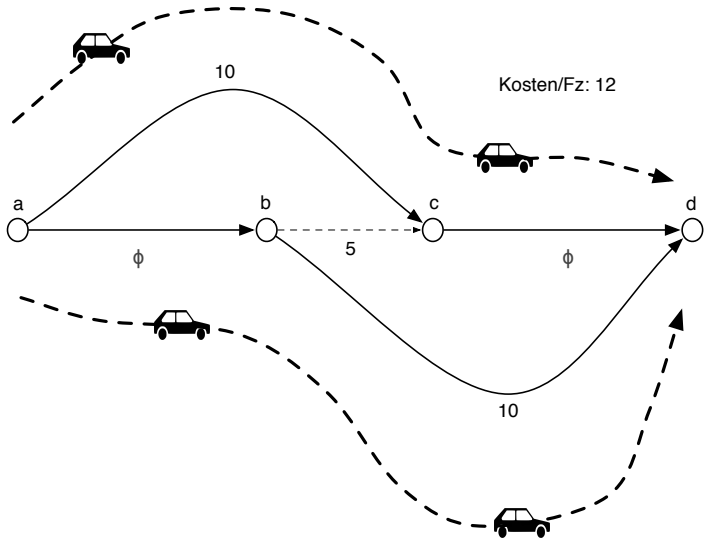


Abbildung 4.2.: Braess-Paradoxon

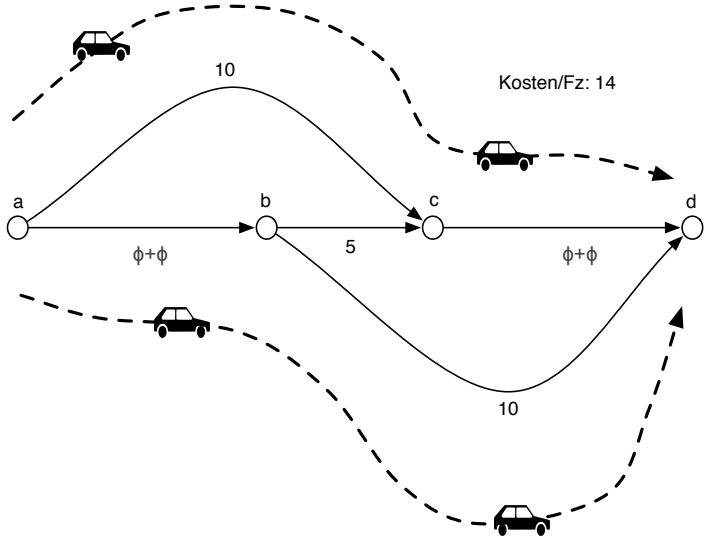


Abbildung 4.3.: Marginal Cost Pricing

Angenommen, die Nachfrage für das Netz in Abbildung 4.2 sei $\lambda_{ad} = 4$ und $(b, c) \notin L$. Dann existieren die beiden Pfade $T = (a, c, d)$ und $B = (a, b, d)$, mit $\tau_T = \tau_B = 10 + \phi$, wobei $\phi \in \{\phi_{ab}, \phi_{cd}\}$ den jeweiligen Fluss über den Link darstellt. Beide Pfade sind symmetrisch und haben identische Kosten. Der UE-Fluss stellt sich folglich mit $\phi_{ab} = \phi_{ac} = \phi_{bd} = \phi_{cd} = 2$ und den Kosten von jeweils 12 pro Pfad ein. Auch der SO-Fluss nutzt diese Aufteilung. Nun sei $(b, c) \in L$, wodurch der mittlere Pfad $M = (a, b, c, d)$ mit $\tau_M = \phi_{ab} + 5 + \phi_{cd}$ möglich wird. Gemäß Annahme 4.1 möchten Fahrer den kürzesten Weg nutzen und nach Annahme 4.2 kennen sie diesen. Das UE ist demnach gestört, da der mittlere Pfad M geringere Kosten

erlaubt. So wird ein Fahrer, bspw. vom unteren Pfad B, auf den mittleren Pfad M wechseln, da dadurch nur die Kosten 10 statt 12 entstehen. Weitere Fahrer werden folgen, da auf M stets geringere Kosten entstehen als über T oder B. Paradoxerweise jedoch ist die Fahrzeit im so entstandenen UE für jeden Einzelnen schlechter als im SO – obwohl die Infrastruktur, durch den neuen Link, eigentlich verbessert wurde.

Folgende Aufstellungen zeigen die Kosten der Pfade bei unterschiedlichen Flüssen. Dabei ist in der ersten Spalte der neue Link (b, c) noch existent. Ab der zweiten Spalte, nutzen die Fahrer M aus:

Pfad T :	2 : 12	2 : 13	2 : 14	1 : 14	0 : 0
Pfad M :	– : –	1 : 10	2 : 11	3 : 12	4 : 13
Pfad B :	2 : 12	1 : 12	0 : 0	0 : 0	0 : 0

Ist M verfügbar, wechselt in diesem Beispiel zunächst ein Fahrzeug von B zu M. Dadurch steigen allerdings auch die Kosten auf (c, d) von 2 auf 3, da dann $\phi_{(c,d)} = 3$. Somit wird der Wechsel zu M noch attraktiver. In der nächsten Iteration wechselt ein weiteres Fahrzeug von B zu M und die Kosten auf (c, d) steigen weiter. Schließlich wechseln auch die Fahrzeuge von T zu M. Es existiert dann für kein Fahrzeug ein Anreiz zurück zu T oder B zu wechseln, da dort die Kosten jeweils höher wären. Der Neubau der Straße stellt sich so als für alle Verkehrsteilnehmer als nachteilig heraus, da keinerlei Absprache über deren Nutzung zwischen den Teilnehmern stattfindet.

Wie oben gezeigt, kann mittels MCP ein SO erreicht werden (vgl. Abbildung 4.3).

Die Maut für die beiden Links (a, b) und (c, d) beträgt jeweils $\phi_l \dfrac{d\tau_l(\phi_l)}{d\phi} = \phi_l$, d.h. $\tau_T^{MCP} = \tau_{(a,c)}^{MCP} + \tau_{(c,d)}^{MCP} = 2\phi_{(c,d)} + 10$, analog gilt $\tau_B^{MCP} = 2\phi_{(a,b)} + 10$ und für M: $\tau_M^{MCP} = \tau_{(a,b)}^{MCP} + \tau_{(b,c)}^{MCP} + \tau_{(c,d)}^{MCP} = (\phi_{(a,b)} + \phi_{(a,b)} \cdot 1) + (5 + 0) + (\phi_{(c,d)} + \phi_{(c,d)} \cdot 1) = 2\phi_{(a,b)} + 2\phi_{(c,d)} + 5$.

Der UE-Fluss auf diesen Gesamtkosten ist gleich dem SO-Fluss, da kein Anreiz besteht, den mittleren Pfad zu nutzen:

Pfad T :	2 : 14	2 : 16
Pfad M :	– : –	1 : 15
Pfad B :	2 : 14	1 : 14

Wäre ein Fahrzeug, wie in der zweiten Iteration dargestellt, versucht auf M zu wechseln, würde dieses Fahrzeug Kosten von 15 erfahren. Der ursprüngliche Pfad B wäre zu bevorzugen. Auch wenn sich alle vier Fahrzeuge auf M befänden und erst ab dann eine MCP-Maut erhoben würde, würde sich das dargestellte UE ohne M einstellen, da individuelle Kosten eines jeden Fahrzeuges von 21 auf 14 sänken. □

Der nächste Abschnitt zeigt, wie ein UE/SO im Allgemeinen bestimmt werden kann.

4.2.3. Frank-Wolfe-Algorithmus

Der bekannteste Lösungsansatz zur Bestimmung des UE (bzw. SO mittels MCP) ist der iterative, auf Gradientenabstieg basierende, *Frank-Wolfe-Algorithmus* (FW) [50] zur Lösung

konvexer Optimierungsprobleme. Pro Iteration ersetzt der FW-Algorithmus das ursprünglich konvexe durch ein lineares, deshalb leichter zu lösendes, aber das ursprüngliche nur approximierende, Optimierungsproblem. Dabei wird für die approximierte Zielfunktion der minimale Extrempunkt bestimmt und anschließend die minimale Lösung auf der Strecke zwischen dem aktuellen Punkt und dem ermittelten Extrempunkt abgesucht. Das Verfahren weist eine Konvergenzrate von $\mathcal{O}(1/i)$ nach i Iterationen auf und ist in dieser Hinsicht optimal für allgemeine konvexe Probleme.

Gegeben sei ein konvexes Optimierungsproblem der Form

$$\min f(\mathbf{x}) \text{ unter der Nebenbedingung } \mathbf{x} \in \mathbf{S}, \tag{4.14}$$

wobei $f\colon \mathbb{R}^n \to \mathbb{R}$ konvex auf $\mathbf{S} \subset \mathbb{R}^n$, dem n-dimensionalen (konvexen) Polyeder der zulässigen Lösungen ist. Der Algorithmus besteht dann aus fünf Schritten:

1. *Finde initiale Lösung,* d.h. wähle eine beliebige Lösung $\mathbf{x_0} \in \mathbf{S}, \mathbf{x_i} = \mathbf{x_0}$.
2. *Bestimme die Abstiegsrichtung* $\mathbf{p_i} = \mathbf{x_i^*} - \mathbf{x_i}$ mit

   $$\mathbf{x_i^*} = \arg\min_{\mathbf{x}\in\mathbf{S}} \left(f(\mathbf{x_i}) + \nabla f(\mathbf{x_i})^T (\mathbf{x} - \mathbf{x_i}) \right), \tag{4.15}$$

   d.h. statt der ursprünglichen Funktion $f(\mathbf{x})$ wird eine tangentiale Approximation durch das Taylorpolynom 1. Ordnung an der Stelle $\mathbf{x_i}$, $f(\mathbf{x_i}) + \nabla f(\mathbf{x_i})^T(\mathbf{x} - \mathbf{x_i})$, minimiert. Dabei ist $\mathbf{x_i^*}$ stets ein Extrempunkt von S.

   Da $\mathbf{x_i}$ während der Iteration i konstant ist und demnach nur $\mathbf{x}$ variabel, ist Gleichung 4.15 mithin ein lineares Programm.
3. *Bestimme Schrittweite* α, durch Lösens des eindimensionalen Problems:

   $$\min_{\alpha\in[0,1]} f(\mathbf{x_i} + \alpha \mathbf{p_i}), \tag{4.16}$$

   zur Bestimmung des Minima auf der Strecke zwischen x_i und x_i^*.
4. *Aktualisiere Lösung,* $\mathbf{x_{i+1}} = \mathbf{x_i} + \alpha_i \mathbf{p_i}, i = i + 1$
5. *Prüfe Abbruchbedingung,* ggf. zu Schritt 2

Bei der Anwendung des FW-Algorithmus zur Bestimmung des UE entspricht das LP des zweiten Schritts gerade dem MC-MCFP, da das Skalarprodukt aus Gleichung 4.15 der Produktsumme aus Gleichung 4.3a entspricht und die Nebenbedingungen des UE-Programms in Gleichung 4.7 Pfadfluss-basierte Varianten der Nebenbedingungen des MC-MCFP darstellen. Zudem entfällt das Kapazitätslimit u_l und nach Proposition 4.6 kann daher Schritt 2 durch LCP-Bestimmung (bzgl. der Linkkosten $\nabla T(\Phi_i)^T$ der aktuellen Iteration i) aller OD-Paare (o, d) mit $\lambda_{od} \neq 0$ gelöst werden.

Wie angemerkt, wird pro Iteration ein Extrempunkt $\mathbf{x_i}$ bestimmt. Die intuitive Interpretation eines Extrempunkts in diesem Kontext ist eine All-or-Nothing-Zuweisung (aufgrund der LCP-Bestimmung im LP-Subproblem). Pro Iteration wird somit zwischen der letzten Lösung und einer All-or-Nothing-Zuweisung in Schritt 3 ein Minimum gesucht. Dazu kann in Schritt 3 eine Heuristik für das eindimensionale Minimierungsproblem verwendet werden, wie z.B. Line Search nach Brent [18].

4.2.4. Dynamische Routenwahl

Ein offensichtlicher Nachteil der im vorherigen Abschnitt dargelegten Techniken ist die temporale Konstanz: Fluss okkupiert permanent die Kanten. Ein Fahrzeug α ist daher ein Hindernis für alle übrigen, denselben Link l nutzenden Fahrzeuge, unabhängig davon, ob in der Realität diese vor oder nach α Link l passieren. Für Rush Hour-Situationen, in denen stets Fahrzeuge nachfließen, mag diese Annahme noch akzeptiert werden (und wird es in der Verkehrsplanung auch). Im Allgemeinen aber ändert sich der Fluss eines Links über die Zeit. Realistischer ist auch, dass $\alpha - 1$ nur dann eine Verzögerung durch α erfährt, wenn beide sich noch zeitgleich auf demselben Link befinden. Vorausfahrende Fahrzeuge hingegen werden nicht aufgehalten, im Gegensatz zum obigen, statischen Fall. Der zentrale Unterschied ist folglich die Link-Verweildauer eines Fahrzeugs. Dadurch entsteht eine temporale Relation zwischen den Fahrzeugen auf einem Link: Ein Fahrzeug kommt früher (oder später) als ein anderes Fahrzeug auf einen gemeinsam genutzten Link.

Die Frage ist, ob unter diesen Umständen dennoch ein – wie auch immer geartetes – „Optimum“ überhaupt existiert. Für das *Dynamische System Optimum* (DSO) lässt sich diese Frage leicht beantworten, denn es muss (mindestens) ein Fluss existieren, der zu minimalen globalen Fahrzeiten führt. Auch existiert ein *Dynamisches User Equlibirum* (DUE), zumindest, wenn folgende FIFO-Eigenschaft gilt [3]:

Annahme 4.8 (FIFO-Eigenschaft)**.** Erreicht Fahrzeug α einen Link vor $\alpha - 1$, so muss α diesen auch vor $\alpha - 1$ wieder verlassen.

Insbesondere für gestaute Zustände ist das aber eine vertretbare Annahme.

Jedoch ist die Berechnung erheblich rechenaufwendiger als im statischen Fall. Viele dynamische Modelle nehmen im Wesentlichen Zeitdiskretisierungen vor, wodurch die dynamische letztlich zu einer statischen Routenwahl degeneriert. Somit ist der FW-Algorithmus verwendbar, wobei wieder das MC-MCFP als Teilproblem entsteht. Auch das MCP-Konzept kann an dynamische Verhältnisse angepasst werden [164]. Jedoch zeigten Koch und Skutella in [78] 2009, dass der PoA im dynamischen Fall im Allgemeinen mit $\Omega(|V|)$ unbegrenzt wachsen kann (im Gegensatz zum statischen Fall, was Roughgarden in dem bereits referenzierten Paper mit dem deutlichen Titel „The price of anarchy is independent of the network topology“ [119] zeigte). Simulative Untersuchungen dieses Zusammenhangs unter realitätsnahen Umständen scheinen bisher noch nicht zu existieren. Es bleibt zu hoffen, dass Selfish Routing in der Realität nicht allzu weit vom Optimum entfernt ist.

Durch die Modellerweiterung um temporale Eigenschaften wird somit nicht mehr der statische Fluss ϕ_l auf Link $l \in L$ betrachtet, sondern der Fluss $\phi_l(t)$ zum Zeitpunkt t. Das betrachtete Zeitintervall wird hier in N diskrete Zeitschritte $t_n, n \in \{0, \ldots, N\}$ unterteilt. Zusätzlich wird die Anzahl des ankommenden Flusses auf einem Link l (inflow) mit $i_l(t_n)$ (sowie mit i_l^r der Inflow ausgehend von Knoten r) und der abfließende Fluss (outflow) mit $o_l(t_n)$ bezeichnet. $t^{rs}_{k(n)}$ bezeichne die Abfahrtszeit eines Fahrzeugs von Knoten r der zum Zeitpunkt t_n an Knoten s ankommt.

Damit lässt sich nach [114, Kaptiel 4] die folgende konvexe Zielfunktion angeben:

$$T(\mathbf{\Phi}) = \sum_{n=0}^{N} \sum_{l=(v,w)\in L} \left\{ \int_0^{i_l(t_n)} \tau_l(\phi_l(t_n), i, o_l(t_n))\, di \right. \tag{4.17}$$

$$\left. + \sum_{o\in O} i_e^r(t_n) \left(\overline{\pi}^{ov}(t_{k(n)}) - \overline{\pi}^{ow}(t_{k(n)}) \right) \right\}, \tag{4.18}$$

wobei $\overline{\pi}^{od}$ die optimale Fahrzeit zwischen o und d zum Zeitpunkt $t_{k(n)}$, dem Startzeitpunkt eines zum Zeitpunkt t_n ankommenden Fahrzeugs von o nach v, darstellt. Für die (zahlreichen und wiederum linearen) Nebenbedingungen sei auf die letztgenannte Quelle verwiesen.

Entscheidend für die weitere Diskussion ist nämlich allein die Feststellung, dass die erste Summe über alle diskreten Zeitschritte läuft. Aufgrund dieser Diskretisierung muss der Graph G angepasst werden. Für jeden Zeitschritt $0, \ldots, N$ und jeden Link aus G muss jeweils ein zusätzlicher Knoten in einen Hilfsgraph eingefügt werden. Dadurch ist es möglich, abzubilden, wann ein Fahrzeug einen Knoten erreicht (für die genaue Konstruktion sei auf die letztgenannte Quelle verwiesen). Im Ergebnis enthält der finale Hilfsgraph ca. $3N$ so viele Kanten und N-mal so viele Knoten.

Das lineare Subproblem des FW-Algorithmus entspricht hierbei wiederum dem MC-MCFP, jedoch auf dem vergrößerten Hilfsgraphen. Dabei müssen aber auch N-mal so viele LCP bestimmt werden wie im statischen Fall (um die zeitlichen Abfolgemöglichkeiten zu beachten). Wird Dijkstras LCP-Algorihtmus mit $\mathcal{O}(|E| + |N| \log |N|)$ verwendet, steigt der Aufwand für die Berechnung eines einzelnen LCP um ca. $3N + N = 4N$. Inklusive des genannten Faktors N, steigt der Aufwand pro FW-Iteration insgesamt um $4N^2$. In [53, Abschnitt 2.3.6] soll ein ganzer Tag mit einminütigen Zeitschritten betrachtet werden, wodurch sich ein Faktor von $8 \cdot 10^6$ gegenüber des statischen Falls pro Iteration ergibt. (Bei Verwendung des A*-Algorithmus mit monotoner Heuristik würde der Aufwand insgesamt um den Faktor $N \cdot 3N = 3N^2$ steigen, da die Laufzeit für ein einzelnes LCP $\mathcal{O}(|V| \log |V|)$ beträgt. In dem konkreten Einzelfall betrüge der Zunahmefaktor immer noch $6 \cdot 10^6$.)

Dieses Beispiel stellt nur eine Approximation des DUE/DSO dar, da eine einminütige Auflösung nicht alle Wechsel von Fahrzeugen auf neue Links, ob kürzerer FFT, berücksichtigen kann. In den verwendeten Graphen des späteren Evaluationskapitels 7 existieren Links mit FFT im Sekundenbereich, wodurch der Rechenaufwand noch mindestens um den Faktor 10 anstiege.

Der nachstehende Abschnitt diskutiert die dargelegten Techniken im Lichte des Verkehrsroutings.

4.3. Diskussion

Dieses Kapitel zeigte auf, wie der Begriff *Optimalität* im Straßenverkehr (hinsichtlich der Fahrzeit) verstanden werden kann (und auch üblicherweise in den Verkehrswissenschaften interpretiert wird). Im User Equilibirum (UE) ist keinem Fahrzeug eine unilaterale Fahrzeitverbesserung möglich, im System Optimum (SO) ist die durchschnittliche Fahrzeit minimal. Die beiden Flüsse unterscheiden sich und zur Erreichung des SO müssen Fahrzeuge i.A. Einbußen hinsichtlich ihrer individuellen Fahrzeit in Kauf nehmen. Da schwerlich von vollkommen altruistischen Fahrern auszugehen ist, kann mittels zusätzlicher Maut-Bepreisung Zwang aufgebaut

werden oder eine Belohnung für gewünschtes Verhalten ausgelobt werden – sofern seitens aller Beteiligten (bzw. der Politik) das SO trotz potentieller individueller Verschlechterung angestrebt wird.

Jedoch erfordert die Bestimmung des UE bzw. SO zum Einen Annahmen, die in der Realität nicht erfüllbar sind. Die Annahme, dass a priori die gesamte Verkehrsnachfrage bekannt ist, widerspricht der Offenheit des dynamischen Systems „Straßenverkehr“. Auch wird davon ausgegangen, dass die Transitzeit exakt bekannt ist, d.h. unvorhergesehene Ereignisse (wie z.B. Unfälle, Trödeln, Witterungsverhältnisse) werden vollkommen ausgeblendet. Aber selbst angenommen, dass die Verkehrsnachfrage und weitere Einflüsse durch historische Daten und Expertenwissen hinreichend genau geschätzt werden können, ist die schlechte Skalierbarkeit der zentralen Optimierung ein Flaschenhals: Sind die Ergebnisse (zentral) berechnet, hat sich die Situation in der Zwischenzeit so sehr verändert, dass sie wertlos sind. Wie die empirische Studie [16] zeigt, bedarf es für Netze realistischer Größe (Dellaware Valley) über 500 Iterationen des FW-Algorithmus bis zur Konvergenz der Lösung eines statischen TAP. Zum Zeitpunkt der Erstellung der Studie im Jahre 1999 bedurfte es weit über 10 Stunden für diese Berechnungen. Trotz fortschreitender Verbesserung der zur Verfügung stehenden Rechenkraft wäre, ob der deutlich erhöhten Anforderungen im dynamischen Fall, ein DUE bzw. DSO für realistische Anliegen nicht hinreichend schnell zu bestimmen. Zudem müsste diese Berechnung, aufgrund der Ungenauigkeit der eingegangenen Schätzungen, kontinuierlich wiederholt werden.

Verteilte Routingprotokolle verzichten daher auf teure, zentrale Optimierung und verwenden im Wesentlichen (mit Unterschieden je nach Protokoll) den LCP, um Fahrzeuge (oder Datenpakete) online weiterzuleiten. Das BeeJamA-Protokoll macht da keine Ausnahme. Allerdings sind zwei Erweiterungen des Protokolls direkt durch diese Darstellungen inspiriert. Erstens wird ein dynamisches Reservierungsmodell in Abschnitt 6.11 zu dem in den Abschnitten 6.1–6.8 eingeführten BeeJamA-Basisprotokoll hinzugefügt und zweitens in Abschnitt 6.11.3 das MCP-Konzept ergänzt. In dieser Arbeit beschränkt sich das BeeJamA-Protokoll aber auf eine Iteration der Pfadzuweisung. Anders ausgedrückt, Fahrzeuge werden über den dynamischen bzw. dynamisch-konsekutiven LCP (vgl. Gleichung 1.5 und 1.6) weitergeleitet.

In zukünftigen Arbeiten kann das BeeJamA-Protokoll mit Hilfe des Reservierungsmechanismus aber in der Art weiterentwickelt werden, dass „Verhandlungsrunden“ zur Vergabe der freien Reservierungsplätze analog zu den Iterationen ähnlich des FW-Algorithmus durchgeführt werden. Zusätzlich könnte ein DUE/DSO im Sinne verteilter Systeme zumindest in lokalen Bereichen zentral bestimmt werden. Statt dem FW-Algorithmus könnten hierzu neuere Verfahren wie Algorithm B [33] und TAPAS [8] mit besseren Konvergenzeigenschaften Verwendung finden.

Kapitel 5

Verteilte Systeme

Dieser Abschnitt führt wichtige Begriffe und Konzepte verteilter Systeme ein. Es wird ein verteilter Leader Election-Algorithmus und mit dem asynchronen Bellman-Ford-Algorithmus ein verteiltes LCP-Verfahren vorgestellt. Beide Algorithmen stellen wesentliche Bausteine des BeeJamA-Protokolls (und vieler anderer) dar. Anschließend werden die Grundzüge von Routingprotokollen dargestellt und das Themengebiet Multiagentensysteme und Schwarmintelligenz diskutiert. Darauf aufbauend kann sodann im folgenden Kapitel das BeeJamA-Protokoll dargestellt werden.

5.1. Grundlegende Begriffe

Der Begriff „verteiltes System“ ist vielschichtig und interpretationsabhängig, es existiert daher keine einheitliche Definition. Enslow diskutierte und bemängelte diesen Umstand [43] schon 1978 und führte sogleich aus, dass ein System nur dann als „verteilt“ bezeichnet werden könne, wenn mindestens die folgenden drei Kriterien erfüllt seien:

Multiple Berechnungskomponenten: Es existieren mehrere *Loci of Computation*, bspw. mehrere heterogene Hardwarekomponenten (statt einem einzigen zentralen Hardwareknoten), Cloudapplikationen oder Serverprozesse. Im Folgenden wird vereinheitlichend der Begriff *Komponente* verwendet. Jede Komponente kann zur Laufzeit prinzipiell verschiedene Aufgaben übernehmen, welche a priori nicht bekannt sind.

Daten: An keiner Komponente sind initial alle notwendigen Daten vorhanden, da kein gemeinsamer, globaler Speicher zur Verfügung steht. Stattdessen muss mittels asynchroner Nachrichten über ggf. verlustbehaftete Kanäle kommuniziert werden. Kollaboration findet mittels Nachrichtenaustausch statt.

Berechnung: Die Softwareprozesse an den einzelnen Komponenten tragen gemeinsam zur Lösung des Problems bei.

Enslow lässt das notwendige Ausmaß der Ausprägung dieser Mindestkriterien intentional offen. Reicht es bspw. wenn viele Komponenten nur einen sehr geringen Teil zur Lösung beitragen, damit eventuell ersetzbar oder gar unnötig wären, und eine Hauptkomponente den Großteil? Oder müssen alle ungefähr identische Beiträge leisten? Dürfen alle Hardwarekomponenten

die ihnen bekannten Informationen zu einem einzelnen Hauptknoten senden, welcher sodann das Ergebnis global berechnet? Enslow selbst gibt umgangssprachlich ohne nähere (formalere) Erläuterung an, dass jeweils ein „hoher Grad“ an Verteilung der drei genannten Aspekte vorhanden sein muss. Gerade an diesem Punkt, was als genügend hoch angesehen kann, machen potentielle formale Definitionen sich angreifbar.

Tanenbaum und van Steen geben in ihrem weitverbreiteten Lehrbuch [136, Abschnitt 1.1] zu dem Thema hingegen die Definition, dass ein verteiltes System „eine Sammlung von unabhängigen Computern ist, die den Benutzern wie ein einzelnes, kohärentes System erscheint“. Dies mag aus Sicht eines Benutzers eine zutreffende Beschreibung sein, unterschlägt aber wichtige Aspekte der vorherigen Mindestkriterien.

Insgesamt scheint eine exakte und zugleich konsensfähige Definition eines verteilten Systems, kaum vorstellbar, da mitunter anwendungs- und kontextabhängige Kriterien relevant sind. Zwar könnte eine geschlossene Arbeitsdefinition nur für diese Arbeit gegeben werden, aber auch diese wäre zwangsläufig suboptimal. Statt dessen wird im Folgenden eine phänomenologische Charakterisierung verteilter Systeme, anhand der für diese Arbeit relevanten, Eigenschaften vorgenommen. Ein Anspruch auf Vollständigkeit wird aus vorstehenden Gründen nicht erhoben.

Die Systemarchitektur, die üblicherweise in diesem Kontext als Vorbild dient, besteht aus, zur Kooperation gezwungenen, vernetzten und geographisch getrennten Computern. *Verteilte Systeme* setzen sich demnach aus heterogenen, von einander unabhängigen Komponenten zusammen, welche als, zur asynchronen Kommunikation mittels Nachrichten untereinander fähigen, Zustandsautomaten, genannt *Prozesse*, modelliert werden. Ein gemeinsamer Speicher existiert nicht. Die Topologie des verteilten Systems wird als Graph $G = (V, E)$ modelliert. Jedem Prozess ist genau einem Knoten zugewiesen und umgekehrt, weswegen im Folgenden die Knotenmenge V auch als Menge aller Prozesse verwendet wird. Die Kanten entsprechen Kommunikationskanälen, wobei nur benachbarte Prozesse direkt kommunizieren können. Jedem Prozess $p \in V$ ist zum Zeitpunkt t ein Zustand $\theta_p(t)$ zugewiesen. Ziel ist, von einem systemweiten Initialzustand $\Theta(t_0) = (\theta_{p_0}(t_0), \ldots, \theta_{p_{|V|}}(t_0))$ aus, durch Zustandstransitionen Konvergenz zu einem ausgezeichneten Zielzustand Θ^* zu erreichen. Zustandstransitionen benötigen zum Teil Eingaben, die lokal nicht verfügbar sind. Um den Zielzustand zu erreichen, muss folglich eine Kooperation mit zumindest einer Teilmenge der übrigen Prozessen via Nachrichten stattfinden.

Zwei Eigenschaften verteilter Systeme seien deutlicher hervorgehoben:

Lokalität: Jeder Prozess verfügt, zumindest initial, nur über lokale Informationen, die nicht ausreichen, selbstständig Zustandstransitionen bis zum gewünschten Zielzustand auszuführen. Der Prozess ist stattdessen abhängig von Informationen der übrigen Prozesse. Ausgetauscht werden Informationen zwischen Prozessen mittels Nachrichten. Es kann unterschieden werden, ob zwischenzeitlich globale Informationen akkumuliert werden, oder ob ein Prozess bis zur Erreichung des Zielzustandes stets nur über lokale Informationen verfügt. Ein Beispiel hierfür ist die dichotome Klassifikation von Routingalgorithmen für Computernetze in Link State- und Distanzvektor-Protokolle (wie sie in Abschnitt 5.5 vorgestellt werden). Das erste Protokoll sammelt an jedem Knoten globale Information und berechnet zentral den LCP, das zweite kennt stets nur einen Nachfolger über den, im Falle von Computernetzen, Pakete weitergeleitet werden sollen und benötigt daher keine globalen Daten. Dennoch gelten beide als verteilte Routingprotokolle. Hierbei wird auch besonders die Schwierigkeit einer sinnhaften Definition deutlich. Angenommen, es existieren n Prozesse, und ein ausgezeichneter Prozess n_0 akkumuliert die gesamten

Information von weiteren $n-2$ Prozessen. Der Definition nach besitzt n_0 keine globalen Informationen, da ihm nur Informationen aus einer beschränkten Umgebung zur Verfügung stehen. Dieser Aspekt wird weiter unten als „Zentralisierung“ noch einmal aufgegriffen.

Verteilte Kontrolle: Die einzelnen Komponenten unterstehen im Allgemeinen keiner zentralen Kontrolle, jeder Prozess entscheidet hingegen selbstständig über lokale Zustandsübergänge. Die Erreichung des Zielzustandes steht somit unter verteilter Kontrolle. In diesem Sinne sind Prozesse und die Kommunikation nebenläufig und voneinander unabhängig.

Asynchronität: Verteilte Kontrolle impliziert letztlich auch, dass die Nachrichtenkommunikation asynchron verläuft: Ein Empfänger kann nicht vorhersagen, ob oder wann eine Nachricht eintrifft. Nicht mal der Clock Drift eines Prozesses in Bezug auf die reale Zeit ist bekannt. Das Gegenteil dieses *asynchronen Modells* ist das *synchrone Modell*: Zustandstransitionen treten Runden-basiert zu global fixierten Zeitpunkten auf, Nachrichten traversieren binnen einer Runde einen Kanal und beliebige, lokale Berechnungen eines Prozesses werden innerhalb einer Runde abgeschlossen. Im Allgemeinen sind die Prozesse aber aufgrund der verteilten Kontrolle asynchron, und müssten mit zusätzlichen Protokollen kostenaufwendig zur Synchronität gezwungen werden.

Die notwendigerweise miteinander durch Nachrichten kooperierenden, von den Prozessen ausgeführten, Algorithmen werden dann als *verteilte Algorithmen* [91] bezeichnet. Mit Ausnahme von einigen Vorüberlegungen werden nachfolgend nur asynchrone verteilte Algorithmen betrachtet. Synchrone verteilte Algorithmen lassen sich zwar häufig leichter entwerfen, umsetzen und beweisen, da nur innerhalb eines starren Rasters agiert wird. Jede Ausführung eines synchronen Systems führt zudem zur gleichen Reihenfolge von Zustandsübergängen. Dafür kann der Aufwand zur Etablierung von Synchronität beträchtlich sein (und letztlich kann vollumfängliche Synchronität nur theoretisch garantiert werden).

Für verteilte Systeme werden u.a. die folgenden Komplexitätsmaße betrachtet:

Nachrichtenkomplexität: Anzahl der gesendeten Nachrichten. Da in der Realität häufig die Kommunikation den Flaschenhals darstellt, handelt es sich i.d.R. um das entscheidende Maß. Mitunter wird auch die *Nachrichtenlänge* zusätzlich betrachtet.

Zeitkomplexität pro Prozess: Klassische Zeitkomplexität der lokalen Algorithmen, die von den Prozessen ausgeführt werden. Nicht selten ist in verteilten Systemen dieser Aspekt von keiner besonderen Bedeutung, da eingehende Nachrichten nur wenig Rechenbedarf auslösen.

Speicherkomplexität pro Prozess: Klassische Speicherkomplexität eines Prozesses. Ähnlich wie bei der Zeitkomplexität ist der benötigte Speicher häufig vernachlässigbar gering.

Zeit bis zur Konvergenz: Benötigte Zeit bis aus dem initialen Zustand der systemweite Zielzustand erreicht wurde. Neben der Nachrichtenkomplexität häufig der relevante Aspekt.

Das BeeJamA-Konzept sei als Beispiel der genannten Begriffe betrachtet. Die vernetzten Navigatoren repräsentieren Prozesse eines verteilten Systems. Linkkosten fallen dezentral an, wodurch nur lokale Informationen an den Navigatoren vorhanden sind. Die Agenten verbreiten

Teilmengen dieser Informationen. Deren Ziel ist es, jedem Navigator bekannt zu machen, über welchen Nachfolger des Straßennetzes, Fahrzeuge mit einem bestimmten Ziel sinnigerweise weiterzuleiten sind. Dieser Zielzustand verändert sich kontinuierlich über die Zeit, weswegen eine ständige Anpassung notwendig ist.

In Abgrenzung dazu stehen parallele Systeme, bei denen prinzipiell vollständige Information vorhanden ist, die Lösung eines Berechnungsproblems auch durch ein einzelne Komponente berechnet werden könnte, zur Laufzeitreduzierung die Aufgabe aber auf mehrere Komponenten verteilt wird.

Wie bereits bemerkt, können verteilte Systeme potentiell „zentralisiert“ werden. Dabei entsteht eine asymmetrische Rollenverteilung, im Extremfall in einem $|V|$:1-Verhältnis: Alle Knoten senden ihre Informationen an einen zentralen Prozess; dieser berechnet die Lösung und verteilt diese wieder an die einzelnen Knoten. Dadurch ergäben sich allerdings mehrere potentielle Nachteile:

Erhöhte Nachrichtenkomplexität: Durch die Migration aller benötigten Daten zu einem zentralen Knoten, bildet sich potentiell ein Flaschenhals im Kommunikationskanal.

Single Point of Failure: Fällt ein zentraler Knoten aus, kann die Berechnung nur fortgeführt werden, sofern andere Knoten diese Arbeit übernehmen können. Es entstünden erneut die Probleme der Migration der benötigten Daten zu einem (noch funktionierenden) zentralen Knoten.

Zu geringe Rechenkraft: Die Rechenkraft des zentralen Prozesses beschränkt die Größe des Gesamtsystems.

Alternativ können die globalen Informationen, statt an einen einzigen zentralen Knoten, auch an allen übrigen Knoten zusätzlich gesammelt werden und dort jeweils identische Berechnungen auslösen. Zwar entfällt dann der Single Point of Failure, die Nachrichtenkomplexität steigt allerdings noch weiter und die Rechenkraft mehrerer Prozesse kann überfordert sein.

Zentralisierungen sind somit nicht immer sinnvoll. Allerdings existieren verteilte Algorithmen (bzw. Zielzustände), bei denen es unausweichlich ist, zumindest gewisse Teilinformation als globale Information in einem Prozess zu akkumulieren. So können viele verteilte Algorithmen nicht ohne die Rückmeldung aller Prozesse über die systemweite Erreichung eines Ziel, der *Konvergenz*, entscheiden (synchrone Varianten i.d.R. schon). Solche Rückmeldungen sind meistens allerdings mit deutlich geringerem Datenaufkommen verbunden, als sämtliche Informationen zu zentralisieren.

Wenn auch nicht immer in Gänze zu vermeiden, sind Zentralisierungen insbesondere dort unvorteilhaft, wo inhärente Deadline-Nebenbedingung existieren. *Echtzeit-fähige verteilte Systeme* sollen somit die Eigenschaft erfüllen, dass sie nicht mehr Informationen an Knoten akkumulieren bzw. nicht längere Berechnungen auf diesen Daten durchführen, als eine mit dem Zielzustand vereinbarte Deadline gebietet.

Wieder diene das Problem des Fahrzeugroutings als Beispiel für die Nachteile einer Zentralisierung eines verteilten Echtzeit-Problems. Würden von einer zentralen Instanz alle Routen berechnet und fiele diese aus, könnte keine weitere Route berechnet werden, alle Fahrer wären betroffen. Auch ist es mit erheblichen Aufwand verbunden, alle Auslastungsdaten zentral zu sammeln, die zeitliche Nebenbedingung – stets aktuelle Information als Grundlage von Routingentscheidungen zu verwenden – wäre gefährdet. Gleiches gilt für die Rechenkraft

eines zentralen Knotens zur Berechnung aller Routinganfragen aller Fahrzeuge, die Antworten kämen für große Verkehrsnetze im Zweifelsfall zu spät.

Unterbleibt eine Zentralisierung, wird im Folgenden von einem *dezentralen* statt einem *zentralen* verteilten Algorithmus gesprochen.

Der anschließende Abschnitt beschreibt kursorisch die benötigten Kommunikationskonzepte in verteilten Systemen.

5.2. Kommunikation in verteilten Systemen

Nachrichten zwischen Prozessen werden über *Kommunikationskanäle* übertragen. In diesem Kapitel wird von bidirektionalen Kanälen ausgegangen, wodurch das gesamte Netz stets stark zusammenhängend ist. Zu Beginn drei Annahmen um potentielle Fehlerquellen zu auszuschließen:

Annahme 5.1 (Verlust- und fehlerfreie Kanäle mit FIFO-Eigenschaft)**.** Nachrichten auf Kommunikationskanälen gehen nicht verloren und kommen beim Empfänger fehlerfrei in der Reihenfolge an, in der sie versendet wurden.

Im Allgemeinen gilt diese Annahme zwar nicht, so kann theoretisch jede Nachricht, inkl. jeder Neuübertragung, auf einem Kommunikationsmedium verloren gehen. Allerdings ist in der Praxis davon nicht regelmäßig auszugehen und üblicherweise eingesetzte Kommunikationsstacks sichern die Fehlerfreiheit und Einhaltung der FIFO-Eigenschaft. Der TCP/IP-Stack nutzt hierfür Sequenznummern, Checksummen, Acknowledgments, Timeouts und Neuübertragungen. Wird stattdessen beispielsweise UDP verwendet – und das böte sich ob des geringeren Overheads für viele der im Folgenden besprochen verteilten Verfahren an – können besonders benötigte Teilaspekte auf den User-Layer verlagert werden. Wie lange eine Übertragung dauert und wann die Nachricht ankommt, ist hingegen vollkommen offen in asynchronen, verteilten Systemen.

Um allerdings Komplexitätsabschätzungen durchführen zu können, ist es hilfreich eine beliebige obere Grenze b für die Übertragung einer Nachricht auf einem Kommunikationskanal festzulegen. Die Algorithmen selbst müssen unabhängig von dieser Obergrenze funktionieren, theoretische Analysen vereinfachen sich dadurch aber erheblich, weswegen dies eine gängige Annahme darstellt.

Fällt die Übertragung auf den Kanälen als Fehlerquelle somit aus, könnte prinzipiell auch ein Prozess Nachrichten verfälschen oder zurückhalten. Dies kann aufgrund eines Fehlers unabsichtlich passieren oder absichtlich, um das verteilte System zu stören. Dies sei ebenfalls ausgeschlossen.

Annahme 5.2 (Ausschluß Byzantinischer Fehler [85])**.** Kein Prozess verfälscht Nachrichten.

Anders ausgedrückt, jeder Prozess versendet Nachrichten in der Form wie sie gemäß Programmlogik vorgesehen ist. Kein Angreifer kompromittiert einen Prozess.

Annahme 5.3. Kein Prozess fällt aus.

Diese Annahme stellt sicherlich die weitreichendste Annahme dar, wird in verteilten Systemen doch üblicherweise einkalkuliert, dass zumindest ein Teil der Prozesse ausfällt und das Gesamtsystem dennoch (soweit möglich) weiterhin funktioniert. Es existieren klassische Verfahren, um den Ausfall von Prozessen zu erkennen, woraufhin die restlichen Prozesse angemessen darauf reagieren können. Das verkompliziert aber nur unnötig die Darstellung der

Verfahren und falls notwendig, kann solch eine Ausfallserkennung leicht ergänzt werden. Ein weiterer Grund ist, dass im späteren BeeJamA-Anwendungsfall der (kurzfristige) Ausfall der Prozesse, wie noch erläutert wird, kein sonderliches Problem darstellt. Die Annahme scheint daher gerechtfertigt.

Zusammengefasst bedeutet dies, dass die Anzahl der Prozesse konstant ist, kein Angreifer die Prozesse manipuliert und Nachrichten fehlerfrei in der ursprünglichen Reihenfolge beim Empfänger ankommen.

In verteilten Systemen können vier Varianten von Kommunikation auftreten:

Direkt: Gilt für zwei Prozesse $r \to s$, so kann eine Nachricht m von r direkt an den Nachfolger $s \in S_r$ gesendet werden. Andernfalls muss m über einen Pfad p an s gesendet werden, wobei p von einem Routingalgorithmus für Computernetze bestimmt wird.

Flooding: Häufig müssen Prozesse aber eine Nachricht in der kompletten d-Umgebung verbreiten. Beim Flooding sendet r die Nachricht m an alle Nachfolgerprozesse $s \in S_r$ und erniedrigt die maximale Reichweite $m.d$, die mit jeder Nachrichtenkopie verknüft ist um 1. Die Empfänger senden m wiederum an alle Nachfolger, aber nur sofern $m.d > 0$ und der Nachfolger nicht der Prozess ist, von dem m empfangen wurde. Ob m auch weitergesendet wird, wenn m bereits zuvor von diesem Prozess aus weitergesendet wurde und ob m auch zum vorherigen Prozess zurückgeschickt wird, hängt von der genauen Umsetzung ab. In jedem Falle geht über jede Kante mindestens eine Nachricht. Mitunter wird die Nachricht dabei selbst auf dem Pfad von Prozesses verändert. Routingprotokolle akkumulieren dadurch Kosteninformationen über die traversierten Links.

Broadcast: Auch hierbei soll jeder Knoten in der d-Umgebung Nachricht m erhalten. Dabei muss m nicht zwangsläufig über jede Kante gesendet werden. Stattdessen wird i.d.R. zunächst verteilt ein minimaler Spannbaum auf dem Netz konstruiert und m darüber versendet. Der Fokus hierbei liegt auf der Zustellung einer invarianten Nachricht, z.B. über den Zustand des Senders. Jeder Knoten im Netz (oder einer gewissen Umgebung) soll die Nachricht empfangen, auf welchem Wege ist dabei irrelevant. Soll m nur eine Teilmenge von Prozessen erreichen, wird von einem Multicast gesprochen.

Convergecast: Das zum Broadcast gegenteilige Verfahren bei dem alle Knoten eine Nachricht an einen einzigen ausgezeichneten Empfängerknoten senden, wird *Convergecast* genannt. In manchen verteilten Sensornetzen müssen die einzelnen Sensoren als Beispiel ihre Messdaten an einen einzigen (zentralen) Auswertungsprozessor senden. Realisiert werden kann ein Convergecast mittels Flooding oder (ggf. kostengünstiger) indem ein minimaler Spannbaum konstruiert wird und die Nachrichten der einzelnen Prozesse darüber an den ausgezeichneten Knoten gesendet werden.

Bei der direkten Variante handelt es sich somit um eine 1:1, bei Flooding und Broadcast um eine 1:n und bei Convergecast um eine n:1 Kommunikation.

Nachrichten können aus unterschiedlichen Gründen von Prozessen versendet werden:

On-Demand: Nachrichten werden bei bestimmten lokalen Zustandsänderungen getriggert, oder von anderen Prozessen explizit angefordert. Es werden dabei nicht mehr Nachrichten versendet als notwendig.

Kontinuierlich: Hierbei werden in einem, je nach Anwendungsgebiet fixen oder variablen, Intervall Nachrichten gesendet. Häufig kann so Programmlogik eingespart werden:

1. Wenn häufig On-Demand-Nachrichten getriggert werden, bspw. um dynamische Umgebungen zu explorieren, kann ebenso kontinuierliches, automatisches Versenden verwendet werden und so komplexe, zustandsabhängige Trigger-Logik eingespart werden.
2. Angenommen, das Intervall ist für den Anwendungskontext klein genug, so dass das Ausbleiben einer Nachricht durch Warten auf die folgende Nachricht kompensiert werden kann. Dann muss z.B. für verlustbehaftete Übertragungskanäle (die nach Annahme 5.1 zunächst ausgeschlossen sind), welche nur gelegentliche Verluste aufweisen, keine zusätzliche Wiederherstellungslogik bereit gehalten werden.

Es entsteht ein asynchrones Runden-Konzept, wobei Nachrichten einer Runde mit einem entsprechenden Tag versehen werden. Empfänger können eingehende Nachrichten somit nach ihrem Alter ordnen.

Kurzum, es ist ein simples, robustes Verfahren, eine in verteilten Systemen gewünschte Eigenschaft. Nachteilig ist die erhöhte Nachrichtenkomplexität, die ggf. auftreten kann, wenn mehr und häufiger Nachrichten erzeugt werden als mit intelligenter On-Demand-Triggerung notwendig.

Auch die Weiterleitung von Nachrichten kann auf unterschiedliche Weise funktionieren:

Stop-And-Wait: Idee ist, synchrone Runden mit einfachen Mitteln in asynchronen Systemen nachzubilden, so dass pro Runde genau eine Nachricht weitergereicht wird. Jeder Prozess speichert seine aktuelle (subjektive) Runde, beginnend bei 0. Die erste, initiale Nachricht (der Runde 0) darf von jedem Prozess ohne weitere erfüllte Voraussetzungen gesendet werden. Die Zustandstransition in die folgende Runde $i > 0$ (inkl. eigenem Nachrichtenversand) ist aber erst zulässig, wenn von einer (kontextabhängigen) Teilmenge der Knoten alle Nachrichten der Runde $i - 1$ angekommen sind. Anders ausgedrückt, wird namensgebend gestoppt und auf andere Prozesse gewartet, bis alle Nachrichten eingetroffen sind. Der Vorteil ist, dass die Nachricht der Runde i auf vollständigen Informationen der vorherigen Runde basieren kann. Nachteilig ist die Wartezeit, ein Knoten muss auf alle Nachrichten der Runde $i - 1$ der direkten Vorgänger warten, diese wiederum müssen zuvor die $i - 2$ Nachrichten ihrer Vorgänger empfangen haben usw. Das Versenden der i-ten Nachricht ist also von allen Vorgängerknoten mit Distanz $\min\{i, \gamma\}$ abhängig, mit γ als Durchmesser des Netzes. Dieses einfache Stop-And-Wait-Verfahren ist ein Spezialfall des allgemeinen Konzepts der *Synchronization* mittels dessen unter gewissen Voraussetzungen synchrone Algorithmen auch in asynchronen Systemen genutzt werden können (siehe hierzu [91, Kapitel 16]).

Best Effort: Im Gegensatz zu einem Stop-And-Wait-Verfahren, wird auf eine eingehende Nachricht unverzüglich eine ausgehende Nachricht gesendet, auch wenn noch nicht alle Vorgängerknotennachrichten vorliegen. Es entfällt die Wartezeit, zuungunsten der verfügbaren Information.

Zur Beschreibung verteilter Systeme und asynchroner Kommunikation existieren verschiedene Formalismen und Notationen. Beispiele hierfür sind (Timed) I/O Automata [75], Pi-Calculus [95], Petri Netze [99] und Temporal Logic Of Actions [84]. In dieser Arbeit werden

Timed I/O Automata (TIOA) als Notation verwendet, da dort leicht die imperative Programmlogik des Prozesses beschrieben werden kann und ein eleganter Weg für die Interprozesskommunikation umgesetzt ist. Dies ist ein erheblicher Gewinn im Gegensatz zu üblicher Darstellung in Pseudocode, wie es aus nicht-verteilten Systemen bekannt ist, da die Lesbarkeit und Verständlichkeit bei asynchronen Zusammenhängen stark leiden kann.

Prinzipiell sind TIOA-Automaten intuitiv verständlich, allerdings benötigt der Aspekt der Kommunikation zusätzliche Erläuterung. Kommunikation wird über sogenannte Input/Output-Aktionspaare realisiert. Zur Übertragung der Nachricht m von Prozess i zu j dient die Ausführung einer Output-Aktion $a(m, i, j)$ in Prozess i. Dadurch wird der (entfernte) Aufruf einer gleichnamigen Input-Aktion $a(m, i, j)$ in Prozess j angestoßen. Ausführlicher stellt Anhang A die notwendigen Konzepte der Timed I/O Automaten vor.

In den beiden folgenden Abschnitten seien zwei klassische verteilte Probleme betrachtet, die Wahl eines Leader-Knotens, sowie die Bestimmung eines LCPs. Sie werden sich als grundlegend für die in den folgenden Abschnitten dargestellten Routingalgorithmen erweisen.

5.3. Leader Election

In verteilten Systemen muss mitunter ein einzelner Knoten ermittelt werden, der stellvertretend für übrige Knoten eine bestimmte Aufgabe übernimmt. In Routingalgorithmen werden häufig Zielbereiche des Netzes durch einen einzelnen Knoten repräsentiert. BeeHive und BeeJamA stellen, wie später noch aufgezeigt, hierfür Beispiele dar. Die Selektion solch eines Knoten wird verallgemeinert als *Leader Election* bezeichnet. Im Folgenden wird ein klassisches Verfahren zur Bestimmung eines Leaders in asynchronen, verteilten Systemen mit beliebiger Topologie erläutert (siehe z.B. [91, Kapitel 15.2]). Für diese Arbeit wurde eine Abwandlung dieses globalen Leader Election-Verfahrens erarbeitet, welches Leader nur in einer lokalen Umgebung zulässt. Zunächst wird die globale Variante vorgestellt, anschließend die lokale.

Sei $G = (V, E)$ der betrachtete Graph. Jedem Prozess $p_i \in V$ ist eine eindeutige ID (unique identifier, UID) i zugewiesen. Sei i^* die größte vorkommende UID. Der Prozess p_{i^*} wird dann als *Leader* bezeichnet. Das Leader Election Problem (LEP) besteht darin, den Leader zu identifizieren und im Netz bekannt zu machen, wobei jeder Prozess initial nur seine eigene ID kennt[1]. Sei $L(p)$ die UID von der $p \in V$ aktuell ausgeht, dass es i^* ist. Ein LEP-Algorithmus ist konvergiert, mithin das LEP korrekt gelöst, sobald gilt

$$L(p) = i^* = \max\{i | p_i \in V\} \qquad \forall p \in V \tag{5.1}$$

In synchronen verteilten Systemen (und bei bekannten Durchmesser γ des Netzes) sendet jeder Prozess in jeder Runde die höchste bekannte UID zu allen Nachbarn, initial seine eigene. Nach γ Runden terminiert der Algorithmus: Genau ein Prozess hat dann seine eigene UID als Maximum behalten und erklärt sich zum Leader. In asynchronen Systemen lässt sich dieser Algorithmus nicht ohne Weiteres umsetzen, da das Konzept der Runde nicht existiert. Zwei Ansätze stehen zur Verfügung:

Synchronisation: Im ersten Fall wird ein einmaliges On-Demand-Flooding mit Stop-and-Wait verknüpft, wodurch Runden entstehen und der vorige, synchrone Ansatz nachgeahmt werden kann. In Runde 0 sendet jeder Prozess seine eigene UID zu seinen Nachfolgern. Hat

[1] Im Allgemeinen ist das Optimum einer total geordneten Menge zu finden.

ein Prozess alle Nachrichten der Runde 0 erhalten, wird ggf. der Leader-Status geändert und das Maximum als Runde-1-Nachricht an die Nachfolger gesendet. Spätestens nach γ Runden ist der Leader-Status eines Knotens konvergiert. Denn: Pro Runde wird eine Nachricht weitergegeben und der Leader kann maximal die Distanz γ aufweisen. Die Nachricht der Runde 0 wird On Demand gesendet, die Nachrichten der späteren Runden sind nur noch weitergereichte.

Vollständig asynchron: Hierbei wird einmaliges On Demand-Flooding und Best Effort miteinander verknüpft. Jeder Prozess sendet initial einmal seine UID an alle Nachbarn. Jeder Prozess reicht eine eingehende Nachricht mit höheren UIDs umgehend weiter. Schlussendlich erhält somit jeder Prozess eine Nachricht vom Leader, da diese stets weitergeleitet wird. Allerdings kann ein Prozess dann nicht ohne weiteres selbst die Konvergenz feststellen, denn eventuell ist eine Nachricht mit potentiell höherer UID nur noch nicht angekommen. In der Praxis kann eine gewisse Zeit gewartet werden, bis davon auszugehen ist, dass mit genügend großer Wahrscheinlichkeit der richtige Leader (eindeutig) gefunden wurde. (Der BeeJamA-Algorithmus wird von diesem heuristischen Warte-Ansatz Gebrauch machen.)

Aufgrund der potentiell entstehenden Wartezeit wird das Synchronisationverfahren verworfen. Die asynchrone Variante sei als *AsyncGlobalLE* bezeichnet. Die Korrektheit folgt aus den beiden folgenden Sätzen:

Satz 5.4. *AsyncGlobalLE* konvergiert zu dem systemweiten Zustand in Gleichung 5.1.

Beweis. Nach Konstruktion werden Nachrichten mit größerer UID weitergeleitet. Da i^* maximal ist, wird die Nachricht von jedem Prozess weitergeleitet. Für jeden Prozess p gilt somit mindestens einmal $L(p) = i^*$. Da nach Konstruktion $L(p)$ nur geändert wird, wenn ein $i^{**} > i^*$ p erreichen würde. Dies ist nicht möglich, da i^* maximal ist. Daraus folgt die Behauptung. □

Satz 5.5. *AsyncGlobalLE* konvergiert spätestens zum Zeitpunkt $\gamma \cdot b$, mit γ als Durchmesser und b als maximaler Transmissionszeit auf einem Kanal.

Beweis. Kein Knotenpaar kann eine größere Distanz als durch γ gegeben aufweisen, woraus unmittelbar die Behauptung folgt. □

Diese LEP-Variante wird typischerweise in der Literatur betrachtet [91, Kapitel 15.2] . Sollen aber mehrere lokale Leader gefunden werden, um das Netz in *Verantwortungsbereiche* partitionieren zu können, muss das Verfahren angepasst werden. Solch eine lokale Variante wird nachstehend erarbeitet.

Als d-LEP sei die LEP-Verallgemeinerung bezeichnet, in welcher der Leader höchstens in einem Umkreis von d Knoten Distanz liegen darf. $N_i^{d,G}$ bezeichne die d-Nachbarschaft von Knoten i in Graph G. Abbildung 5.1 illustriert die Idee. Der Graph zerfällt dabei in mehrere markierte Leader-Komponenten. Eine Leader-Komponente zeichnet sich im konvergierten Zustand dadurch aus, dass nach Entfernung aller Leader-Komponenten mit höherem Leader aus dem Graph, alle Prozesse in der Umgebung den auf diesem Subgraphen übriggebliebenen Leader kennen.

Für ein gegebenes d ist Konvergenz eingetreten, wenn alle Leader-Komponenten etabliert sind, d.h. wenn gilt:

$$L(p) = i^*_{G_n} := \max\{i | p_i \in V_n\} \qquad \forall p \in N^{d,G_n}_{i^*_{G_n}}, 0 \leq n \leq |V| \tag{5.2a}$$

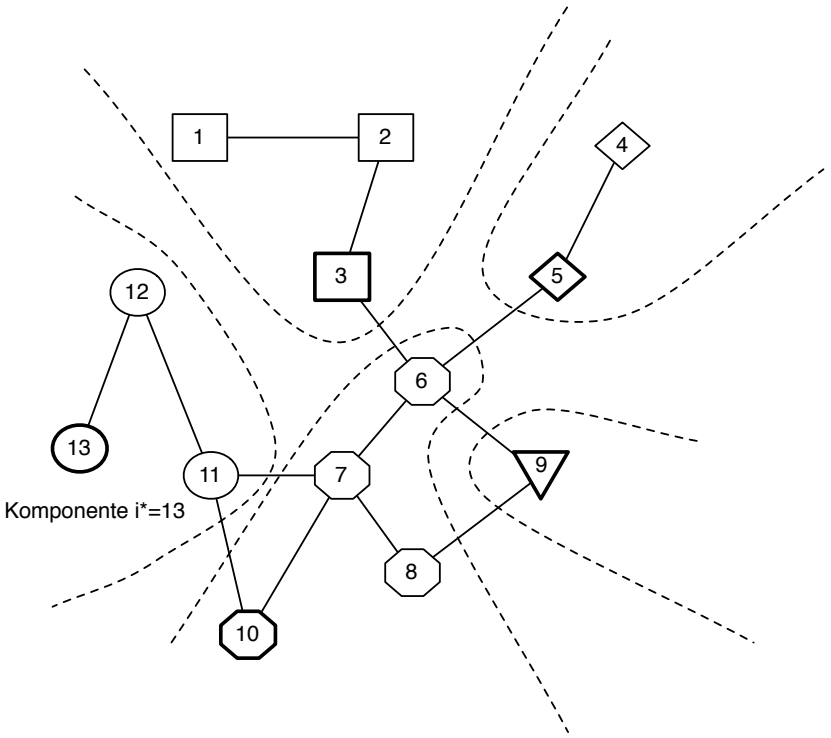


Abbildung 5.1.: Konvergenz einer 2-LEP-Instanz

mit

$$V_0 = V, E_0 = E, G_0 = G, i^*_{G_0} = i^* \tag{5.2b}$$

$$V_n = V_{n-1} \setminus N^{d,G_{n-1}}_{i^*_{G_{n-1}}} \qquad \forall n \geq 1 \tag{5.2c}$$

$$E_n = E_{n-1} \setminus \{(i,j) | i \notin V_n \vee j \notin V_n\} \qquad \forall n \geq 1 \tag{5.2d}$$

$$G_n = (V_n, E_n) \tag{5.2e}$$

Um dieses Problem zu lösen, kann AsyncGlobalLE so abgeändert werden, dass die Nachrichten ein Hop Limit von d erhalten. Dabei entsteht jedoch ein zu berücksichtigendes Problem: Wenn ein Leader i seinen Leader-Status verliert, müssen die umliegenden Prozesse darüber informiert werden, da sie eventuell p_i noch als Leader führen. Angenommen für einen Zeitpunkt t_1 und ein $p_i \in V$ gilt $L(p_i) = i$ (p_i ist demnach Leader) und für einen $p_j \in N^d_{p_i}$ gilt $L(p_j) = i$ (p_j glaubt, dass p_i der Leader ist). Weiter angenommen, zu einem Zeitpunkt $t_2 > t_1$ gilt nunmehr $L(p_i) = k$ (i ist somit kein Leader mehr) und $k \notin N^d_{p_j}$, weswegen stets gelten wird $L(p_j) \neq k$. Stattdessen besteht die Möglichkeit, dass der Zustand $L(p_j) = i$ fälschlicherweise ad infinitum erhalten bleibt.

Dieser Umstand wird berücksichtigt, indem der ehemalige Leader die Nachricht, ob dessen er seinen Leader-Status verlor, unkonditional mit vollem Hop Limit weiterleitet, auch wenn das Hop Limit es eigentlich nicht nicht mehr gestatten würde. Dadurch erfahren alle Prozesse $p_j \in N^d_{p_i}$ von dem Leader-Wechsel. Falls sie p_i als Leader führten, beginnen sie den Prozess von vorne, d.h. sie erklären sich selbst zum Leader. Falls es aber weitere potentielle Leader-Prozess in ihrer Umgebung gibt, müssen sie nicht zwangsläufig davon erfahren. Daher wird

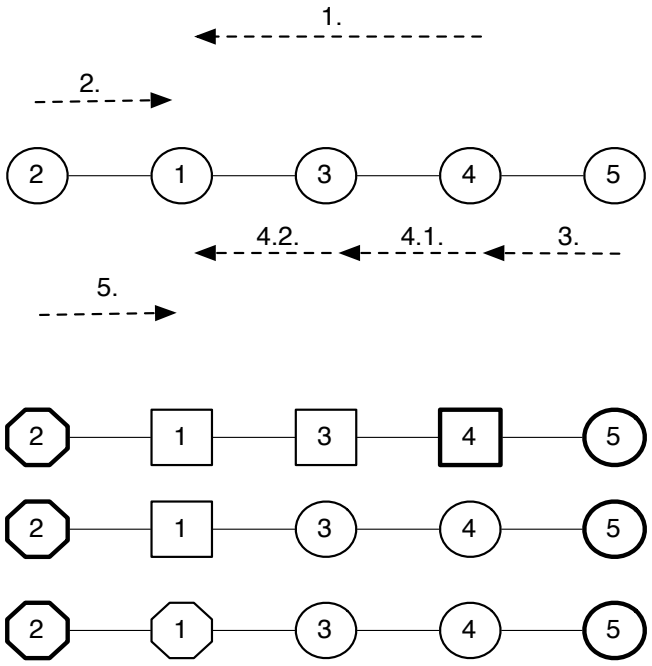


Abbildung 5.2.: Veraltete Leader-Informationen

von On-Demand-Flooding auf kontinuierliches Flooding umgestellt, d.h. alle Leader-Prozesse senden nach einem bestimmten Intervall erneut.

Abbildung 5.2 verdeutlicht die Situation, wobei $d = 2$. Im oberen Teil wird schrittweise eine mögliche Nachrichtenausbreitung dargestellt. Im unteren Bereich sind Leader (fett) und Leader-Komponenten (Knotenform) im Verlauf bis zur Konvergenz dargestellt.

Ersichtlich ist, dass als erstes die Nachricht von Prozess 4 den Prozess 1 erreicht. Dementsprechend nehmen die Prozesse 3 und 1 den Prozess 4 als Leader an. Als zweites erreiche die Nachricht von Prozess 2 den Prozess 1. Diese Nachricht wird aber verworfen, da kleiner als 4. Als drittes erreicht nun die langsamere Nachricht von Prozess 5 den Prozess 4. Dadurch verliert dieser seinen Leader-Status und leitet die Nachricht mit vollem Hop Limit weiter, weswegen die Nachricht in Schritt 4.2. auch Prozess 1 erreicht. Dieser lernt, dass 4 seinen Leader-Status verloren hat, erklärt sich selbst zum Leader und floodet. Bliebe es dabei, bildeten jeweils Prozess 1 und 2 eine eigene Leader-Komponente. Da aber nach Verstreichen eines bestimmten Intervalls Prozess 2 ebenfalls floodet, erreicht schließlich Prozess 1 die Nachricht von 2 in Schritt 5 erneut.

Der Algorithmus ist in Automat 4 dargestellt. Die *floodQueue* wird vom Automat *Flooder* geerbt (siehe Anhang A). Eine Nachricht muss vom Typ *[CurrentLeader, ExcludedNodes, HopLimit]* sein, wobei der zweite Parameter eine Möglichkeit bietet, Knoten auszuschließen, zu denen die Nachricht gefloodet wird. Dadurch kann ausgeschlossen werden, das Nachrichten wieder zurück zu dem vorherigen Prozess gesendet werden. Die Input-Aktion kapselt die zuvor genannte Logik. Kommt die Nachricht von einem Leader und ist diese UID größer als die höchste aktuell bekannte UID (Zeile 13), wird die Leader-Variable angepasst und die Nachricht weitergefloodet (Zeile 14–15). Kommt die Nachricht hingegen von einem Prozess j, der selbst nicht Leader ist, j aber noch an i als Leader geführt wird, muss dies geändert werden und die Nachricht wird mit maximalen Hop Limit weitergesendet (Zeile 16–18).

Automaton 4 Async Leader Election

```
1:  Automaton AsyncLE(i, Δs, d) extends Flooder(i, N_i)
2:
3:      States:
4:          L(i) ← i
5:          floodQueue ← [leader, ∅, d]
6:          nextSend ← clock + Δs
7:
8:      Transitions:
9:          Input receive(m, j, i)
10:             Effects
11:                 m.leader ← m[0]
12:                 m.limit ← m[1]
13:                 if j = m.leader ∧ m.leader > L(i) then
14:                     L(i) ← m.leader
15:                     floodQueue ⊢ [m.leader, {j}, m.limit]
16:                 else if (L(i) = j ∧ m.leader ≠ j then
17:                     L(i) ← i
18:                     floodQueue ⊢ [i, {j}, d]
19:         Internal flood()
20:             Preconditions
21:                 clock = nextSend ∧L(i) = i
22:             Effects
23:                 floodQueue ⊢ [i, ∅, d]
24:                 nextSend ← clock + Δs
25: End
26:
27: Automaton AsyncLESystem(Δs, b, d) where Δs > 0, b > 0, d > 0
28:     Components:
29:         ∀i ∈ V: AsyncLE(i, Δs, d)
30:         ∀(i, j) ∈ E: TimedChannel(i, j, b)
31:         ∀(i, j) ∈ E: ζ_ij ← 1
32: End
```

Der an gleicher Stelle abgebildete Automat AsyncLESystem bildet den Rahmen des AsyncLE-Algorithmus. Für jeden Knoten wird eine Algorithmus-Instanz erstellt, sowie die zugehörigen Kanäle. Abschließend werden die Nachrichtenkosten ζ_{ij} auf 1 gesetzt, wodurch der Flooder bei jedem Weiterleiten einer Nachricht, das Hop Limit der Nachricht um Eins dekrementiert.

Satz 5.6. AsyncLE konvergiert zum systemweiten Zustand in Gleichung 5.2a.

Beweis. Es wird konstruktiv gezeigt, dass die Leader-Komponenten der Reihe nach entwickelt werden können. Für $n = 0$ wird auf G_0 die d-Umgebung von $i^*_{G_0}$ betrachtet. Für diesen Subgraphen ist Satz 5.4 anwendbar, da die d-Umgebung aufgrund des Hop Limits d den gesamten erreichbaren Graphen darstellt. Da letztlich kein anderer Prozess von außerhalb dieser d-Umgebung Leader für einen Prozess innerhalb der d-Umgebung werden kann, ist

der übrige Graph irrelevant. Somit ist schließlich an jedem Prozess $p_j \in N^d_{i^*_{G_0}}$ der Leader i^* bekannt, mithin ist das Konvergenzkriterium für $n = 0$ erfüllt. Für $n = 1$ entsteht G_1 durch Entfernen von $N^d_{i^*_{G_0}}$ aus G_0. In G_1 existiert wieder eine maximale UID $i^*_{G_1}$, für die die d-Umgebung betrachtet wird. Es ist das gleiche Argument erneut anwendbar, womit das Kriterium für $n = 1$ erfüllt ist. Dieses Konzept lässt sich fortsetzen. Bei spätestens $n = |V|$ sind alle Leader-Komponenten entstanden, woraus die Behauptung folgt. □

Für die Zeit bis zur Konvergenz gilt $\Omega(\gamma)$:

Satz 5.7. AsyncLE konvergiert spätestens zum Zeitpunkt $\gamma \cdot (3db + \Delta s)$, mit d als maximaler Distanz des Leaders, b als maximaler Transmissionszeit auf einem Kanal und Δs als Flooding-Intervall.

Beweis. Bezeichne $K(l)$ die Komponente eines Leaders l. Angenommen bis auf $K(i^*)$ haben sich bereits alle Leader-Komponenten etabliert. Wenn $K(i^*)$ sich als letztes etabliert, können Knoten ihre Komponentenzugehörigkeit wechseln oder sogar ihren Status von Leader zu Non-Leader. Das Worst Case-Szenario geht demnach davon aus, dass $K(i^*)$ sich zunächst etablieren muss und erst dann alle weiteren folgen. Die Leader-Komponente für i^* ist spätestens zum Zeitpunkt db etabliert und wird sich fortan nicht mehr ändern. Angenommen, einen Prozess $p_j \in K(i^*), p_j \neq p_{i^*}$, war zuvor Leader für einen Prozess $p \notin K(i^*)$. Da p_j nun kein Leader mehr ist (da durch p_{i^*} ersetzt), leitet dieser Prozess die Nachricht von p_{i^*} mit vollem Hop Limit weiter und informiert so alle Prozesse $p \in N^d_{p_j}$ von denen er potentiell Leader gewesen sein könnte. Diese Nachricht ist höchstens nach db zusätzlicher Zeit bei all diesen Prozessen angekommen. Ggf. ernennt sich ein Prozess $p_k \in N^d_{p_j}$ deswegen selbst zum Leader und floodet erneut (vgl. Abbildung 5.2). Sollte ein anderer Leader $p_l \in N^d_{p_k}$ mit $l > k$ in der Umgebung sein, wird er spätestens nach einer Wartezeit von Δs ebenfalls flooden und p_k wird die Nachricht spätestens nach db weiteren Zeiteinheiten erhalten und seinen Status von Leader auf Non-Leader ändern. Insgesamt sind demnach $K(i^*)$ und alle inzidenten Komponenten zum Zeitpunkt $db + (db + \Delta s + db)$ etabliert.

Es kann jedoch sein, dass durch die Etablierung von $K(i^*)$ und der angrenzenden Komponenten weitere ehemalige Leader (aus den zu $K(i^*)$ inzidenten Komponenten) ihren Status verloren haben. Dann dauert es wieder maximal $db + \Delta s + db$ Zeit, bis die nächsten Komponenten etabliert sind. Diese Komponenten grenzen jeweils an die zuletzt etablierten. Insgesamt kann sich dieses Prozedere maximal γ-mal wiederholen, da spätestens dann alle Prozesse in einer etablierten Komponente sein müssen. Somit gilt $db+\gamma\cdot(db+\Delta s+db) = db+\gamma\cdot(2db+\Delta s) \leq \gamma\cdot(3db+\Delta s)$ und damit die Behauptung. □

Damit ist *AsyncLE* korrekt und lokale Leader lassen sich effizient verteilt bestimmen.

5.4. Least Cost Paths

Ein weiterer elementarer Baustein ist die verteilte Bestimmung von LCPs. Die hier betrachtete Aufgabe besteht darin, an einem Rootknoten r die LCPs zu umgebenden Zielknoten zu sammeln. Diese Single-Source-Variante wird im Routingkontext häufig betrachtet, ermöglicht es doch, wie im folgenden deutlich wird, ohne zusätzlichen Nachrichtenaufwand die LCPs für alle Prozesse zum Rootknoten zu ermitteln.

Das verteilte LCP-Problem gliedert sich in zwei Phasen: *Exploration* und *Einsammeln*. In der ersten Phase werden ausgehend von r, Nachrichten im Netz verbreitet, dabei die Kosten

des von einer Nachricht zurückgelegten Pfades aggregiert und an den empfangenden Knoten hinterlegt. Als Aggregierungsfunktion wird i.d.R. die Summierung der Linkkosten verwendet. In der zweiten Phase werden diese Informationen eingesammelt und zu r zurückgebracht.

Die Exploration kann mit einer verteilten, asynchronen Variante [91, Kapitel 15.4] des klassischen Bellmann-Ford-Single-Source-LCP-Algorithmus [25] umgesetzt werden. Die nicht-verteilte Variante des Algorithmus wird als Ersatz für den Dijkstra-LCP-Algorithmus [35] verwendet, wenn negative Kantengewichte auftreten oder wenn negative Kreise in einem Graphen erkannt werden sollen. Für verteilte Anwendungen ist der Algorithmus geeignet, da, im Gegensatz zu dem Dijkstra-Algorithmus, alle ausgehenden Kanten relaxiert werden, nicht nur die günstigste. Dabei werden Pfade nebenläufig anhand von lokalen Informationen erkundet. Der Dijkstra-Algorithmus hingegen entscheidet mittels globaler Informationen, welche Kante als nächstes relaxiert wird. Zwar kann auch ein Dijkstra-Algorithmus verteilt und asynchron umgesetzt werden, jedoch wird dazu zusätzliche Kommunikation zur Synchronisation benötigt, um eine Entscheidung über die als nächstes zu relaxierenden Links treffen zu können. Der Bellman-Ford-Ansatz basiert auf einer Breitensuche, der Dijsktra-Ansatz hingegen auf einer Bestensuche.

Der *verteilte, asynchrone Bellmann-Ford*-Algorithmus floodet vom Rootknoten r ausgehend Nachrichten, in welcher die Kosten von r zum aktuellen Knoten der Nachricht aggregiert werden. In der Literatur ist wieder die Variante unbegrenzter Reichweite beschrieben. Für das BeeJamA-Protokoll wird jedoch, wie im Falle der Leader Election, eine lokale Variante benötigt, bei der nur die Ziele aus der d-Umgebung von r exploriert werden.

Ein Empfängerprozess $n \in N_r^d$ hält zwei Variablen vor. Zum Einen die Variable $C(n)$, in der die geringsten Kosten von $r \rightsquigarrow n$ gespeichert werden, die durch eine Nachricht an n bekannt gemacht wurden. Zum Anderen die Variable $P(n)$, in der der Vorgängerknoten $p \in P_n$ gespeichert wird, von dem die Nachricht mit den geringsten Kosten gesendet wurde.

Sei $\check{\pi}_{rn} = (r, \ldots, p_{rn}, n)$ ein LCP von r zu n. Dann ist Konvergenz eingetreten, wenn gilt:

$$P(n) = p_{rn} \qquad \forall n \in N_r^d \tag{5.3}$$

$$C(n) = \delta(\check{\pi}_{rn}) \qquad \forall n \in N_r^d \tag{5.4}$$

Der AsyncBF-Algorithmus ist in Automat 5 dargestellt. Ein ausgezeichneter Rootknoten r floodet initial eine Nachricht mit Kosten 0 an alle seine Nachfolger. Nicht-Root-Prozesse speichern die empfangenen Kosten plus Linkkosten w_{ji} vom Sender $j \in P_i$ zum Empfänger $i \in V$ in der lokalen Variable *cost*, sofern durch die Nachricht ein günstigerer Pfad entdeckt wurde und leiten die Nachricht weiter. Jeder der eingehenden Nachrichten wird, in Hinblick auf die approximierten Kosten des LCP, auch als *Schätzung* interpretiert. Nachfolgende Nachrichten können jedoch durchaus verbesserte Kosten bekannt machen. Daher sind bis zur Konvergenz nur Schätzungen bekannt, die im Folgenden auch als *Estimated LCP-Kosten* (ELCP-Kosten) bezeichnet werden.

Ist dieser Algorithmus konvergiert, kennt jeder Knoten i, wie nachstehend noch gezeigt wird, den Vorgängerknoten auf dem LCP von r zu i. Jedoch kann wiederum von den Prozessen (somit insbesondere auch von r) mit lokalen Informationen nicht festgestellt werden, ob Konvergenz eingetreten ist. Auch kennt der Rootknoten r dann noch nicht die LCP zu den übrigen Knoten oder die damit verknüpften Kosten. Mittels Convergecast ließe sich beides durch folgende Ergänzung beheben: Sobald ein Prozess i eine neue, verbessernde Kostenschätzung erhält und als Reaktion diese Distanz weiterverbreitet, wird zusätzlich ein Acknowledgement an den

Automaton 5 Local Dynamic LCP

```
1: Automaton AsyncBFRoot(r, d) extends Flooder(r, N_r)
2:
3:     States:
4:         floodQueue ⊢ [0, ∅, d]
5: End
6:
7: Automaton AsyncBFNonRoot(i) extends Flooder(i, N_i)
8:
9:     States:
10:        cost ← ∞
11:        parent ← nil
12:
13:    Transitions:
14:        Input receive(m, j, i)
15:            Effects
16:                m.cost←m[0]
17:                m.limit←m[1]
18:                if m.dist + ω_ji < cost then
19:                    cost ← m.cost + ω_ji
20:                    parent ← j
21:                    floodQueue ⊢ [cost, {j}, m.limit]
22: End
23:
24: Automaton AsyncBFSystem(r, b, d) where b > 0, d > 0
25:    Components:
26:        AsyncBFRoot(r, d)
27:        ∀i ∈ V, i ≠ r: AsyncBFNonRoot(i)
28:        ∀(i, j) ∈ E: TimedChannel(i, j, b)
29:        ∀(i, j) ∈ E: λ_ij ← 1
30: End
```

Vorgängerknoten $P(i)$ gesendet – jedoch erst wenn von allen Nachfolgern ein Acknowledgement angekommen ist. Blätter senden somit als erstes ein Acknowledgement. Hat der Rootknoten von allen Nachbarn ein Acknowledgement empfangen, ist Konvergenz eingetreten, ggf. können die übrigen Prozesse mittels Broadcast darüber informiert werden. Für diese Arbeit ist diese Convergecast-Ergänzung aber nicht notwendig.

Im Folgenden sei die Korrektheit des Verfahrens mittels vier Lemmata gezeigt:

Lemma 1. Der Wert der Variable *cost* wird durch keine eintreffende Nachricht erhöht, d.h. über die Zeit kann der Wert höchstens monoton fallen.

Beweis. Die Variable *cost* wird mit „∞“ initialisiert. Anschließend wird die Variable nur durch den Ausdruck $cost \leftarrow m.cost + \omega_{ij}$ verändert. Dabei ist das Kantengewicht ω_{ij} per Konvention endlich und positiv. Die empfangene Distanz $m.cost$ muss ebenfalls endlich sein, da nur Distanzen versendet werden, die kleiner (und damit endlich) sind als der aktuelle Wert

der Variable *cost*. Da die Aktualisierung von *cost* konditional erfolgt, und zwar nur, wenn ein Absinken des Wertes erfolgt, kann der Wert von *cost* nicht steigen. □

Lemma 2. AsyncBF terminiert (in endlicher Zeit).

Beweis. Da nur positive Linkgewichte zugelassen sind, können Nachrichten keine Kreise durchlaufen. Denn sonst würden sich die Kosten erhöhen und die Nachricht bei Schließung eines Kreises gemäß Konstruktion und Lemma 1 (Kosten an einem Knoten steigen zwischenzeitlich nicht) nicht weitergeleitet. Die Anzahl der (einfachen) Pfade ist aber endlich ist, weswegen an einem Knoten auch höchstens endlich viele Nachrichten ankommen und folglich höchstens endliche viele Relaxierungen auftreten können. Bei jeder Relaxierung wird eine endliche Anzahl Nachrichten generiert, die in endlicher Zeit beim Zielprozess ankommen. Somit vergeht höchstens endlich viel Zeit bis AsyncBF keine Nachrichten mehr generiert, was der Behauptung entspricht. □

Lemma 3. Für jeden erreichbaren Zustand gilt: Für alle Nachbarn $i, j \in V$ gilt entweder $i.cost + \omega_{ij} \geq j.cost$ oder $i.cost$ ist an i in Richtung j versendet oder für den Versand bereits vorgesehen ($i.cost \in i.floodQueue$).

Beweis. Mittels Fallunterscheidung wird geprüft, unter welchen Umständen die Ungleichung des Lemmas nicht erfüllt ist:

1. Angenommen $i.cost = \infty$, dann kann $j.cost$ entweder ebenfalls noch uninitialisiert sein oder bereits Kenntnis über einen günstigeren Pfad erhalten haben, jeweils gilt die Ungleichung $i.cost + \omega_{ij} \geq j.cost$.

2. Jetzt sei angenommen $i.cost$ habe einen endlichen Wert:

   a) Gilt $i.cost \geq j.cost$ ist die Ungleichung ebenfalls erfüllt.

   b) Gilt $i.cost < j.cost$ sind zwei Fälle zu unterscheiden:

      i. Entweder kann über die Linkkosten die Ungleichung wieder hergestellt werden, $\omega_{ij} \geq j.cost - i.cost \Rightarrow i.cost + \omega_{ij} \geq j.cost$.

      ii. Oder dies ist nicht der Fall, $\omega_{ij} < j.cost - i.cost$. Dieser Zustand kann nur durch Absinken des Wertes von $i.cost$ eintreten, denn zunächst sind $i.cost$ und $j.cost$ uninitialisiert. Sinkt danach $j.cost$ bleibt die Ungleichung erhalten, folglich kann $i.cost < j.cost$ nur durch Absinken von $i.cost$ eintreten. Bei jedem Absinken von $i.cost$ sendet i eine Nachricht an j und es muss dann unterschieden seit wann $i.cost < j.cost$ gilt:

         A. Entweder schon seit einem früheren Zustand, aber dann befindet sich die Nachricht im Kanal (und kommt in endlicher Zeit an), wodurch das Lemma erfüllt ist.

         B. Das Absinken hat sich erst im aktuellen Zustand ergeben, dann befindet sich die Nachricht mindestens in der Versendequeue, wodurch das Lemma ebenfalls erfüllt ist.

Mehr Fälle können nicht auftreten, es folgt die Behauptung. □

Lemma 4. Sei $\check{\pi}_{v_0,v_n} = (v_0, \ldots, v_n)$ ein LCP zwischen v_0 und v_n. Werden die Links (v_i, v_{i+1}) für aufsteigendes $i, 1 \leq i \leq n-1$, relaxiert, gilt

$$C(v_n) = \delta(\check{\pi}_{v_0,v_n}) \tag{5.5}$$

Beweis. Durch vollständige Induktion über $i \geq 1$:

Induktionsannahme: Nach der Relaxierung von (v_{i-1}, v_i) gilt $C(v_i) = \delta(\check{\pi}_{v_0,v_i})$.

Induktionsanfang: Es gilt $C(v_1) = \omega_{v_0,v_1} = \delta(\check{\pi}_{v_0,v_1})$, da AsyncBF bereits durch Initialisierung eine Nachricht mit Kosten 0 an alle Nachbarn, damit auch an v_1, sendet, diese bei Empfang (der nach Lemma 3 nach endlicher Zeit eintritt) um die Kosten des Links (v_0, v_1) erhöht werden und da auch Subpfade von $\check{\pi}_{v_0,v_n}$ LCPs sind.

Induktionsbehauptung: $C(v_{i+1}) = \delta(\check{\pi}_{v_0,v_{i+1}})$

Induktionsschritt:

$$\begin{aligned}
\delta(\check{\pi}_{v_0,v_{i+1}}) &= \sum_{k=1}^{i+1} \omega_{v_{k-1},v_k k} && (5.6)\\
&= \sum_{k=1}^{i} \omega_{v_{k-1},v_k} + \omega_{v_i,v_{i+1}} && (5.7)\\
&= \delta(\check{\pi}_{v_0,v_i}) + \omega_{v_i,v_{i+1}} && (5.8)\\
&= C(v_i) + \omega_{v_i,v_{i+1}} && (5.9)\\
&= C(v_{i+1}) && (5.10)
\end{aligned}$$

Im ersten Schritt wird die Definition ausgenutzt, im letzten, dass die Relaxierung definitiv durchgeführt wird (es könnte ein Knoten bereits zuvor so relaxiert worden sein, dass keine weitere Relaxierung mehr möglich ist; dies wird allerdings eingangs im Lemma ausgeschlossen). □

Satz 5.8. Nach Terminierung von AsyncBF ist Konvergenz zu dem Zustand in Gleichung 5.4 eingetreten.

Beweis. Da zu jedem Knoten $v \in V$ mindestens ein LCP $\check{\pi}_{r,v}$ existiert, reicht es zu zeigen, dass alle Links auf genau einem LCP der Reihe nach relaxiert werden, da dann Lemma 4 gilt und somit die Behauptung folgt. Um das zu zeigen, sei o.B.d.A. angenommen, dass nur ein LCP existiere. Existieren mehrere LCPs, hat die Nachricht nur über einen LCP die höchste Ausbreitungsgeschwindigkeit. Es reicht o.B.d.A. nur diesen LCP zu betrachten, da über andere LCPs keine Reduzierung der Kosten mehr verbreitet werden können.

Die aufsteigende Relaxierung folgt per vollständigen Induktion über die Distanz $q \geq 1$ eines Knotens v_i von r auf dem LCP aus Lemma 3. Für $q = 1$ folgt der Empfang der Nachricht in endlicher Zeit aus der Initialisierung von AsyncBF (Senden einer Nachricht an alle Nachbarn, somit auch an v_1). Nun sei angenommen, dass jeder Knoten mit Distanz $q-1$ ebenfalls in endlicher Zeit eine Nachricht erhält. Dann sei v_q der Knoten mit Distanz q in

dem LCP und sei v_{q-1} dessen Vorgänger. Da v_{q-1} eine Distanz von $q-1$ zu r hat, gilt nach Induktionsvoraussetzung, dass v_{q-1} bereits eine Nachricht erhalten hat. Nach Lemma 3 muss v_{q-1} eine Nachricht an v_q senden, welche in endlicher Zeit auch ankommt und zur Relaxierung und zum Setzen von $P(v_q) = v_{q-1}$ verwendet wird, da zum Zeitpunkt des Empfangs an v_{q-1} und bis zur Relaxierung von (v_{q-1}, v_q) gelten muss $C(v_{q-1}) + \omega_{v_{q-1},v_q} < C(v_q)$. Gölte dies nicht, hätte eine Nachricht über einen anderen LCP vorher v_t erreichen müssen, was eingangs ausgeschlossen wurde.

Da somit für jeden LCP im Graph Lemma 4 angewandt werden kann und da nach Lemma 1 an keinem Knoten die Kosten sich später noch einmal erhöhen können, folgt die Behauptung. □

Damit ist AsyncBF korrekt. Allerdings seien nachfolgend noch untere und obere Grenzen der Nachrichtenkomplexität gezeigt. Die untere Grenze geht auf [12, S. 450] zurück, hier wiedergegeben nach [91, S. 508].

Satz 5.9. Sei $n \geq 4$ die Anzahl der Knoten in der d-Umgebung des Rootknoten r. Sei b die maximale Zeit für eine Nachrichtenübertragung zwischen zwei beliebigen, benachbarten Knoten. Dann existiert stets ein gewichteter Graph mit n Knoten, für den der AsyncBF-Algorithmus mindestens $\Omega(2^n)$ Nachrichten sendet und $\Omega(2^n b)$ Zeit bis zur Konvergenz benötigt.

Demnach existieren Eingabeinstanzen für die mindestens exponentiell viele Nachrichten generiert werden. Das BeeJamA-Protokoll teilt Eigenschaften des im Folgenden noch diskutierten Distanzvektorprotokoll, welches wiederum auf dem AsyncBF basiert. Demnach wird das Distanzvektorprotokoll auch diese ungünstige Nachrichtenkomplexität erben. Für das BeeJamA-Protokoll muss dies verhindert werden. Der Beweis zu diesem Satz wird daher aus zweierlei Gründen betrachtet. Zum Einen, da die Beweismechanik weiter unten für das Distanzvektorprotokoll noch einmal aufgegriffen wird. Zum Anderen, da dabei deutlich wird, dass die große Anzahl von Nachrichten dadurch entsteht, dass Knoten mehr als eine Nachricht des Rootknotens weiterleiten müssen. Dieser Umstand wird später ausgenutzt, die Nachrichtenkomplexität zu senken.

Beweis. Sei $k = \lceil \frac{n-2}{3} \rceil$. Sei G der Graph mit n Knoten in Abbildung 5.4. Falls n ungerade ist, wird der Knoten v_{k+1} hinzugefügt. Bis auf die rechtsseitig abfallenden Kanten sind alle Gewichte Null. Sei v_0 der Rootknoten. Es wird gezeigt, dass der Knoten v_k in einer Ausführung absteigend alle Schätzwerte (ELCP-Kosten) aus $\{2^k-1, 2^k-2, \ldots, 3, 2, 1, 0\}$ annimmt. Dazu sei angenommen, dass die oberen Pfade (also ein Pfad ohne Kante $(v_a, v_b), a, b \in \{1, \ldots, v_k\}, b = a+1$) eine deutlich schnellere Übertragung ermöglicht als die unteren (horizontalen) Pfade. Dann ist die erste Schätzung, die v_k erreicht $2^k - 1$ (über den oberen Pfad), denn $\sum_{l=1}^{k} 2^{k-l} = 2^k - 1$. Anschließend erreicht die Nachricht, welche von v_0 bis v_{k-1} nur den oberen Pfad nahm und dann die Kante (v_{k-1}, v_k) den Knoten v_k, wodurch ebendort die Schätzung $2^k - 2$ bekannt wird (Senkung durch 1, da die letzte obere Kante mit Gewicht 2^0 ausgelassen wurde). Danach erreicht die Nachricht von v_{k-2} nach v_{k-1} den Knoten v_{k-1} (d.h. die Nachricht kam bis v_{k-2} über den oberen Pfad). Da die obere Kante mit Gewicht 2^1 zu v_{k-1} ausgelassen wurde, verringert sich die aktuell beste Schätzung an v_{k-1} um 2 auf $2^k - 4$. Der Knoten v_{k-1} sendet die Nachricht folglich weiter und zwar über beide Wege, oben und unten herum. Da der obere Pfad schneller ist, erreicht diese Nachricht den Knoten v_k zuerst, wodurch die Schätzung dort

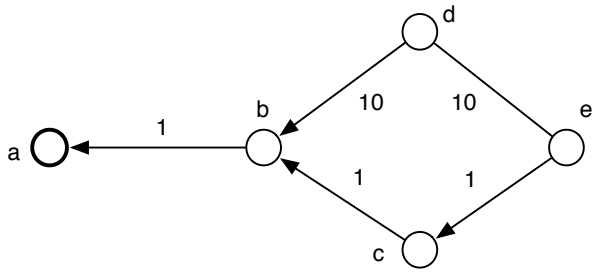


Abbildung 5.3.: Vorgänger-Beziehung des Bellman-Ford-Ansatzes

Abbildung 5.4.: Graph zum Beweis des Satzes 5.9

zu $2^k - 3$ geändert wird. Schließlich erreicht die Nachricht über den unteren Pfad ebenfalls v_k und die Schätzung dort wird auf $2^k - 4$ korrigiert. Dieses Prinzip lässt sich fortsetzen, bis eine Nachricht, die nur über den unteren, horizontalen Pfad gesendet wurde, an v_k ankommt und die Schätzung final auf 0 absenkt. Dabei wurden aber alle eingangs genannten Schätzwerte in absteigender Reihenfolge angenommen, wozu insgesamt 2^k Nachrichten benötigt wurden (pro Schätzungsänderung eine Nachricht), also $\Omega(2^k) = \Omega(2^{\frac{n}{2}})$. Jetzt wird der Übertragungskanal über (v_k, v_{k+1}) als sehr langsam im Vergleich zu den übrigen Kanälen angenommen. Dadurch stauen sich alle 2^k Nachrichten in diesem Kanal (genauer: in dem Ausgangspuffer von v_k zu v_{k+1}). Benötigt jede dieser Nachrichten die maximale Übertragungszeit (es kann jeweils nur eine Nachricht gleichzeitig übertragen werden), wird $\Omega(2^{\frac{n}{2}} b)$ Zeit bis zur Konvergenz benötigt. □

Obere Schranken fallen sogar faktoriell aus.

Satz 5.10. Sei n die Anzahl der Knoten, e die Anzahl der Kanten in der d-Umgebung des Rootknotens und b die maximale Transmissionzeit auf einem Kanal. Dann gilt für die Nachrichtenkomplexität $\mathcal{O}((n-1)! \cdot e)$ und für die Zeit bis zur Konvergenz $\mathcal{O}(n! \cdot b)$.

Beweis. Es gibt maximal $(n-1)!$ unterschiedliche (d.h. kreisfreie) Pfade von einem beliebigen Knoten i_0 zu einem Knoten j (entsprechend der Anzahl der Permutationen ohne Wiederholungen der $n-1$ Knoten in einem vollständigen Graph, wenn i_0 als Startknoten festliegt). Im Worst Case kommen die Nachrichten über die einzelnen Pfade absteigend an j an, d.h. die schlechteste Schätzung zuerst, die beste zuletzt. Der Knoten j muss daher jede dieser Nachrichten weitersenden, wodurch über jeden Link (genauer: über den zugehörigen Kanal) maximal $(n-1)!$ Nachrichten gesendet werden. Somit ist eine obere Schranken für die Nachrichtenkomplexität durch $\mathcal{O}((n-1)! \cdot e)$ gegeben. Für die Zeit bis zur Konvergenz gilt folglich $\mathcal{O}(n! \cdot b)$, denn an den n Knoten können jeweils maximal $(n-1)!$ Nachrichten ankommen, die jeweils maximal t Zeiteinheiten zur Übertragung benötigen (es kann jeweils nur eine Nachricht in einem Kanal sein). □

Average Case Analysen sind hingegen nicht so leicht durchführbar. In [146] wird gezeigt, dass die Nachrichtenkomplexität unter der Annahme von unabhängig und identisch verteilten Linkkosten polynomiell in der Anzahl der Prozesse $|V|$ beschränkt ist. Die Annahme ist aber unrealistisch, da sie bspw. impliziert, dass die durchschnittliche Verzögerung auf allen Links gleich ist. Üblicherweise werden aber einige Links stärker ausgelastet sein als andere und der Beweis zur unteren Grenze greift genau diesen Punkt auf.

Für die folgende Feststellung wird der Begriff des In-Trees benötigt. Ein *In-Tree* ist ein gewurzelter Baum mit Wurzel r, bei dem alle Kanten in Richtung von r zeigen, d.h. es existiert von jedem Knoten genau ein gerichteter Pfad zu r.

Proposition 5.11. Der Graph $(N_r^d, \{(v, P(v)|v \in N_r^d\})$ bildet nach der Terminierung von AsyncBF einen In-Tree mit Wurzel r, da nach Satz 5.4 gilt, dass jeder Knoten v einen Vorgänger $P_v \in \check{\pi}_{rv}$ speichert, wodurch genau ein LCP zu r beschrieben ist.

Wie oben bereits erwähnt, kann dieser In-Tree für ein Convergecast verwendet werden. Im Routing werden über diesen Baum die zu routenden Objekte geleitet. Diese Idee wird später wieder aufgegriffen und modifiziert, u.a. dahingehend um die exponentielle Worst Case-Komplexität zu reduzieren.

5.5. Routing

Im allgemeinsten Sinne bezeichnet Routing die Wahl von Pfaden zur Lenkung von Verkehrsaufkommen in einem Netz mit alternativen Wegen. Das bekannteste Anwendungsgebiet ist die Übermittelung von Nachrichtenverkehr in einem paketvermittelten Kommunikationsnetz, wie z.B. das Routing von Datenpaketen in IP-Netzen. Ein anderes Anwendungsgebiet ist das Thema dieser Arbeit, das Lenken von motorisierten Straßenverkehrsströmen.

Der nächste Abschnitt erläutert kursorisch wichtige Konzepte klassischer Routingprotokolle. Der übernächste Abschnitt stellt das auf dem AsyncBF-basierenden Distanzvektorprotokoll näher vor. Die dortigen Grundprinzipien finden sich in vielen naturinspirierten Routingprotokollen, so auch in BeeHive und BeeJamA.

5.5.1. Prinzipien klassischer Routingprotokolle

Dem Routing zugrunde liegt das Problem, für eine zu routende Entität (Fahrzeuge, Datenpakete, ...), hier als *Token* bezeichnet, in einem Graphen vom Startknoten einen Pfad zum Zielknoten zu bestimmen und das Token über diesen Pfad zu leiten. Die Bestimmung eines Pfades wird hierbei von einem oder mehreren *Routingkomponenten* durchgeführt, ebenso wie die ggf. notwendige Weiterleitung eines Tokens. Im Falle von IP-Netzen entsprechen diese Routingkomponenten den IP-Layern der einzelnen Netzknoten auf dem Pfad (also der eigentlichen Infrastruktur), welche das Datenpaket auch weiterleiten. Im Falle des Straßenverkehrs besteht die Infrastruktur zunächst nur aus (passiven) Straßen, weshalb offensichtlich zusätzliche Routingkomponenten eingeführt werden müssen. Dies wird in nachfolgenden Kapiteln näher dargestellt. Auch müssen Fahrzeuge im strengen Sinne nicht „weitergeleitet“ werden, sie können sich (von Schadensfällen abgesehen) automobil bewegen.

Routing findet in der Regel in einem verteilten System statt, Kommunikationsnetze sind hierfür gute Beispiele. Durch die Einführung von verteilten Routingkomponenten können auch Straßennetze in solche verteilten Systeme „umgewandelt“ werden.

Häufig wird zwischen Routing und *Forwarding* unterschieden. Routing bezeichnet dabei dann den Gesamtprozess aus globaler Sicht, Forwarding die lokale Routingentscheidung an einem Knoten und das Weiterleiten des Tokens. In diesem Kontext wird auch zwischen *Routingprotokoll* und *-algorithmus* unterschieden. Als Protokoll wird das Zusammenwirken der lokalen Algorithmen beschrieben. Anders ausgedrückt, führt jeder Prozess an einem Knoten des verteilten Systems einen Routingalgorithmus aus, welcher ankommende Token weiterleitet (Forwarding) und Nachrichten mit Statusinformationen empfängt, verwaltet und versendet. Die Gesamtheit dieser einzelnen Entscheidungen ergibt das Protokoll. Routingprotokolle in einem verteilten System benötigen Informationen über die Topologie und ggf. über die gegenwärtige Auslastung des Netzes. Die Verbreitung dieser Informationen an die einzelnen Komponenten wird im Folgenden als *Dissemination* bezeichnet.

Routingprotokolle können u.a. bzgl. folgender Aspekte klassifiziert werden:

Dezentral: Werden an einer Routingkomponente Informationen zentral gesammelt oder werden ausschließlich lokale Informationen benötigt? Bei einer zentralen Sammlung werden globale Informationen an mindestens einem Knoten gesammelt und anhand dieses Wissens optimale Pfade (bzgl. einer gegebenen Routingmetrik) bestimmt. Dezentrale Protokolle hingegen akkumulieren keine globalen Daten. Stattdessen muss das Forwarding an jedem Knoten mittels lokaler Daten ausgeführt werden.

Dynamik: Werden sich ändernde Netzzustände berücksichtigt? Das umfasst sich ändernde Netztopologie als auch sich ändernde Auslastungen, wodurch Token ggf. über andere Pfade geroutet werden sollten.

Kreise: Können Kreise entstehen oder verhindert das Protokoll, dass Token eventuell sogar beliebig häufig in eine Schleife geroutet werden? Das prinzipielle Auftreten von Kreisen ist häufig bei dynamischen Protokollen gegeben, sobald sich Topologie oder Auslastung ändern und diese Änderungen nur in Teilen des Netzes bekanntgemacht wurden. Dabei kann es zu Diskrepanzen in dem Informationsstand an den einzelnen Routingkomponenten kommen, welche zu Kreise führen. Beispiele werden im Folgenden diskutiert.

Ziele: Die Ziele von Routingprotokollen, neben der allgemeinen Anforderung Token zum Ziel zu leiten, können unterschiedlicher Natur sein. Beispielsweise kann die Maximierung des Gesamtdurchsatzes im Fokus stehen (System Optimum) oder die Minimierung der Varianz der benötigten Zeit bis zur Ankunft (der sog. Jitter).

Es haben sich zwei klassische Protokollprinzipien [82, Kapitel 4.5] herausgebildet:

1. *Link-State-Protokolle* verwenden globale Topologie- und Kosteninformationen. Dazu senden alle Knoten mittels Broadcast (i.d.R. durch Flooding umgesetzt) die Kosten aller direkt angrenzenden Links, die namensgebenden *Link States*, zu allen übrigen Knoten. Dadurch erhält jeder Knoten ein globales Weltbild und kann sodann mittels eines Single Source-LCP-Algorithmus daraus eine Routingtabelle ableiten.

2. Ein *Distanzvektorprotokoll* (DVP) verzichtet auf Zentralisierung, ist aber, wie nachfolgend noch gezeigt wird, aufgrund der rein lokalen Informationen anfälliger gegen Kreise (und weitere Störungen, die sich aus einem beschränkten Weltbild ergeben). Jeder Knoten kennt namensgebend nur die Distanz und die Richtung, den sogenannten *Vektor*, zu einem Ziel. Die Richtung entspricht einem *Next Hop* auf einem LCP zum Ziel. Realisiert

Tabelle 5.1.: Generische Routingtabelle

T_i	d_1	$\cdots$	d_n
n_1	$c_{1,1}$	$\cdots$	$c_{n,1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n_m	$c_{1,m}$	$\cdots$	$c_{n,m}$

wird das DVP, in dem Nachrichten von Nachbar zu Nachbar (üblicherweise nach dem Bellman-Ford-Prinzip) weitergereicht werden. Eine naheliegende Abwandlung des DVP ist das *Distanzpfadprotokoll* (DPP), bei welchem nicht nur der Next Hop, sondern der gesamte LCP an jedem Knoten bekannt gemacht wird. Eine Nachricht enthält somit nicht nur die akkumulierten Kosten, sondern den gesamten zurückgelegten Pfad. Die Nachrichtenlänge steigt entsprechend von $\mathcal{O}(1)$ auf $\mathcal{O}(\gamma)$ an.

Aufbauend auf diesen Prinzipien wurden unterschiedliche Routingprotokolle für IP-Netze umgesetzt, welche die Internet-Besonderheiten wie (CIDR-)Adressierung, Ports, unzuverlässige Kanäle und dergleichen berücksichtigen. Das bekannteste Link-State-Protokoll ist Open Shortest Path First (OSPF) [98]. Die bekanntesten DVPs sind das Routing Information Protocol (RIP) [61] und das Border Gateway Protocol (BGP) [115], wobei es sich bei letzterem genau genommen um ein DPP handelt.

RIP, ursprünglich für das Interior Gateway Routing innerhalb von autonomen Systemen eingesetzt, gilt ob seiner Linkkosteninsensitivität als veraltet. Zudem war der Nachrichtenoverhead in machen Situation nicht vernachlässigbar, da alle RIP-Knoten alle 30 Sekunden ihre gesamten Routingtabellen an alle direkten Nachbarn versenden. Ersetzt wurde RIP durch OSPF, welches die Linkkosten berücksichtigt und in kleineren Netzen trotz des Link State-Prinzips auch geringeren Nachrichtenoverhead aufweist. Für das Exterior Gateway Routing wird mit BGP ein DVP (bzw. DPP) eingesetzt, *da für große Netze eine Zentralisierung praktisch nicht mehr möglich ist.*

Routingtabellen Prinzipiell würde es für klassische Routingprotokolle reichen, pro Knoten und pro Ziel nur den günstigsten Next Hop zu speichern (vgl. das In Tree-Konzept des vorherigen Abschnitts), da nur über diesen Nachbarn Token weitergeleitet werden. Jedoch werden aus Redundanzgründen die Kosten über jeden Nachbarn in einer *Routingtabelle* (Kostenmatrix) mit $\mathcal{O}(|V|^2)$ Speicherkomplexität vorgehalten. Erhöhen sich nämlich über den zuvor günstigsten Next Hop die Kosten, so dass nun ein anderer Next Hop auf dem LCP liegt, kann das Forwarding sofort auf Alternativen zurückgreifen, ohne auf Nachrichten, die einen neuen, günstigsten Next Hop etablieren, warten zu müssen. Tabelle 5.1 zeigt eine generische Instanz solch einer Routingtabelle T_i für Knoten i, den Zielknoten $d_1, \ldots, d_n$ und den Nachbarn $n_1, \ldots, n_m \in N_i$. Ist d als Zielknoten in T_i vorhanden, wird auch notiert $d \in T_i$. Der Tabelleneintrag an Knoten i für Zielknoten d über Nachbarn n wird notiert als T_i^{nd}.

Das folgende Kapitel geht auf das DVP näher ein, stellen die dortigen Prinzipien doch eine wichtige Grundlage für das BeeJamA-Protokoll dar.

5.5.2. Das Distanzvektorprotokoll

Ein Distanzvektorprotokoll greift klassischerweise den asynchronen Bellman-Ford-Algorithmus aus Abschnitt 5.4 zur Lösung des verteilten Single Destination-LCP-Problems auf. Das Grundprinzip ist einfach: Jeder Knoten v floodet Nachrichten als Rootknoten gemäß des Bellman-Ford-Ansatzes. Angenommen, eine Nachricht traversiert dabei den Link (i, j). Im üblichen Bellman-Ford-Algorithmus würden dann die Linkkosten ω_{ij} zu den Gesamtkosten addiert und die Knoten dadurch mit den Kosten $v \rightsquigarrow j$ annotiert. Nach einem zusätzlichen Convergecast wäre damit die Single Source-Variante gelöst. Werden allerdings die Linkkosten ω_{ji} addiert, lernen die Knoten die Kosten $j \rightsquigarrow v$. Dadurch etabliert sich ein In-Tree mit Wurzel v und LCP-Kosteneinträgen in den Routingtabellen. Somit ist – ohne zusätzliches Convergecast – das verteilte Single Destination-Problem gelöst. Da jeder Knoten einen Rootknoten im Sinne des AsyncBF-Algorithmus darstellt, kann auch an jedem Knoten ein Token mit beliebigem Ziel weitergeleitet werden.

Wie im vorherigen Abschnitt bereits dargelegt, wird aber statt nur der günstigsten, die Kosten über jeden Nachbarn in einer Tabelle gespeichert. Jede über den Pfad $\pi = (v_0, \ldots, v_{n-1}, v_n)$ an Knoten v_n ankommende Nachricht $m = \delta(\pi)$ ist eine Schätzung der Kosten zu einem Ursprungsknoten v_0. Das Tupel $(v_{n-1}, \delta(\pi))$ wird namensgebend als *Distanzvektor* bezeichnet, da es die „Richtung“ und die ELCP-Kosten zu einem Zielknoten angibt.

Ein eintreffender Distanzvektor wird als Tabelleneintrag $T_{v_n}^{v_0,v_{n-1}}$ gespeichert.

Automat 6 stellt eine simple Umsetzung eines DVP dar. Der Rootsubprozess an jedem Knoten floodet zu Beginn eine Nachricht analog zum Bellman-Ford-Algorithmus. Die Vorgehensweise des Empfangsprozesses im Einzelnen:

1. Erzeugen einer leeren Kostenmatrix als Routingtabelle.
2. Wird eine Nachricht empfangen, wird die lokale Routingtabelle aktualisiert.
3. Sind durch den Empfang von Nachrichten neue beste Einträge entstanden, werden die assoziierten Kosten weiterpropagiert. Dies entspricht exakt dem Bellman-Ford-Prinzip, Nachrichten immer dann weiterzuleiten, sobald sie günstigere Kosten mitbringen.

Eingehende Token können dann anhand eines günstigsten Eintrags in der Routingtabelle weitergeleitet werden.

Satz 5.12. Für das Distanzvektorprotokoll gilt für die Nachrichtenkomplexität $\Omega(|V|2^{\frac{|V|}{2}})$ und für die Zeit bis zur Konvergenz $\Omega(|V|2^{\frac{|V|}{2}}b)$, mit b als maximaler Transmissionszeit auf einem Kanal.

Beweis. Der Beweis baut auf dem Beweis für Satz 5.9 auf, jedoch sei $|V|$ ungerade. Dann sei der Graph in Abbildung 5.5 betrachtet, bei dem eine Schleife den Knoten u_{k+1} mit dem Knoten u_1 verbindet, mit $k = \dfrac{|V|+1}{2}$ und $s = k - 1$. Knoten u_1 sendet nun die Nachricht und erzeugt dabei nach der Mechanik des ursprünglichen Beweises 2^k Nachrichten. Danach passieren alle diese Nachrichten die Schleife, wo sie von ihrem Ursprungsknoten u_1 nicht weitergesendet werden. Der Knoten u_2 ist im DVP im Gegensatz zum AsyncBF-Szenario ebenfalls „aktiv“ und sendet Nachrichten statt sie nur weiterzuleiten. Diese Nachricht erzeugt bis zur Schleife nur 2^{k-1} Nachrichten. Nach der Passage der Schleife erreichen diese Nachrichten jedoch auch das letzte Dreieck bis zum Ursprungsknoten, wodurch insgesamt auch wieder 2^k Nachrichten

Automaton 6 Simple Distance Vector Protocol

Automaton DVPRoot(i) **extends** Flooder(i, N_i)

  States:

    floodQueue $\vdash [0, \emptyset, \infty, i]$

End

Automaton DVPNonRoot(i) **extends** Flooder(i, N_i)

  States:

    T$\leftarrow$ [][]

    $\forall i \in V$: T[i].cost $\leftarrow \infty$

    $\forall i \in V$: T[i].parent $\leftarrow$ nil

  Transitions:

    Input receive(m, j, i)

      Effects

        m.cost$\leftarrow$m[0]

        m.src$\leftarrow$m[2]

        T[m.src][j].cost $\leftarrow$ m.cost + τ_{ij}

        newMinParent$\leftarrow \arg\min_{k \in N(i)} \{T[m.src][k].cost\}$

        newMinCost$\leftarrow T[m.src][k]$

        if newMinCost $<$ T[m.src].cost **then**

          T[m.src].cost $\leftarrow$ newMinCost

          T[m.src].parent $\leftarrow$ newMinParent

          floodQueue $\vdash [T[m.src].cost, \emptyset, \infty, m.src]$

End

Automaton DVPSystem(b) **where** $b > 0$

  Components:

    $\forall i \in V$: DVPRoot(i), DVPNonRoot(i)

    $\forall (i, j) \in E$: TimedChannel(i, j, b)

    $\forall (i, j) \in E$:$\lambda_{ij} = 1$

End

erzeugt wurden. Diesem Schema folgend erzeugt jeder der unteren Knoten 2^k Nachrichten. Für die oberen Knoten gilt das aber ebenso, denn $o_j, 1 \leq j \leq s$, sendet eine Nachricht an u_j, wo dann ebenfalls wieder die gesamte Schleife durchlaufen wird. Jedoch leitet o_j die nach der Schleifenpassage ankommende Nachricht nicht an u_{j+1} weiter, da es der Ursprungsknoten ist. Diese eine Nachricht weniger wird allerdings durch die erste Nachricht, die o_j an u_{j+1} sendet kompensiert. Insgesamt erzeugt somit auch jeder obere und damit alle $|V|$ Knoten mindestens 2^k Nachrichten. Somit gilt $\Omega(|V|2^{\frac{|V|}{2}})$ für die Nachrichtenkomplexität. Da angenommen werden kann, dass die Nachrichten auf der Schleifenkante (u_{k+1}, u_1) die maximale Übertragungszeit benötigen, auch $\Omega(|V|2^{\frac{|V|}{2}} b)$ für die Zeit bis zur Konvergenz.

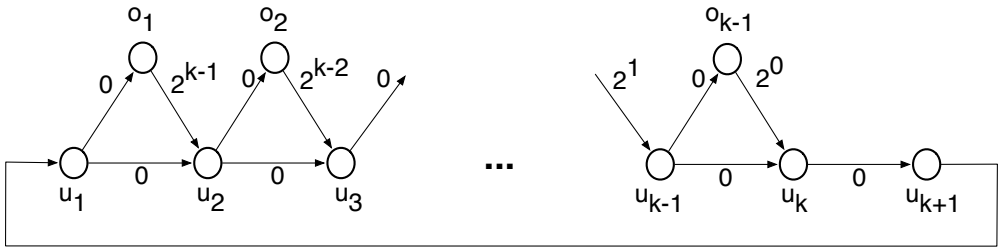


Abbildung 5.5.: Graph zum Beweis des Satzes 5.12

□

Satz 5.13. Für das Distanzvektorprotokoll gilt für die Nachrichtenkomplexität $\mathcal{O}(|V|!|E|)$ und für die Zeit bis zur Konvergenz $\mathcal{O}((|V|+1)!|E|b)$, mit b als maximaler Transmissionszeit auf einem Kanal.

Beweis. Der Beweis baut auf dem Beweis für Satz 5.10 auf. An einem Knoten können wiederum maximal $(|V|-1)!$ viele Nachrichten von einem Rootknoten ankommen, die weitergeleitet werden müssen auf maximal $|E|$ Kanten. Allerdings existieren $|V|$ Rootknoten, woraus die Behauptung folgt. □

Das bereits angesprochene RIP ist eine Umsetzung dieses Konzepts für IP-Netze. RIP ist lastinsensitiv, d.h. es unterstützt als Linkkosten nur 1 und ∞ für verfügbare respektive ausgefallene Links. So können dynamische Topolgieänderungen antizipiert werden, jedoch keine Änderungen in der Auslastung. Um die Dynamik abzubilden, sendet der Rootprozess nicht nur initial, sondern kontinuierlich erneut eine Nachricht. Da ausgefallene Links (bzw. die Router an Knoten) in Computernetzen keine Nachrichten mehr weiterleiten können, ist das obige simple Protokoll dort minimal abgeändert, so dass Knoten mittels Discovery-Prozessen inzidente Links überprüfen und die Nachrichten mitversenden (statt der ggf. ausgefallen Knoten). In der Praxis wird das dadurch realisiert, dass die Ergebnisse der Discovery-Phase in die lokalen Tabellen eingearbeitet werden und anschließend die gesamte Tabelle alle 30 Sekunden an alle direkten Nachbarn gesendet wird. Auch werden Nachrichten nicht unmittelbar weitergeleitet, sondern durch diesen Austausch der Tabellen zweimal pro Minute. Die Ausbreitungsgeschwindigkeit von Nachrichten ist in der Realität daher recht gering.

Diese dynamische DVP-Variante leidet unter zwei bekannten, teilweise zusammenhängenden, Problemen, die später im BeeJamA-Protokoll vermieden werden müssen:

Count-to-Infinity: Veränderte Kantengewichte werden unter Umständen nur sehr langsam im Netz propagiert. Zur Verdeutlichung sei der linke Graph aus Abbildung 5.6 betrachtet. Im konvergierten Zustand weiß Knoten b, dass c direkter Nachbar zu Kosten 1 ist und Knoten a weiß, dass c über b zu Kosten 2 erreichbar ist. Darüber hinaus hat b aber auch erfahren, dass c über a zu Kosten 3 (von b zu a, zurück zu b und schließlich zu c) erreichbar ist. Mit den vorhandenen, rein lokalen Informationen kann b nicht wissen, dass der Weg über a zu c auch wieder über b führt. Solange der konvergierte Zustand erhalten bleibt, führt das zu keinerlei Problemen: b wird ob der geringeren Kosten Pakete immer direkt zu c senden. Ändern sich die Kosten der Kante (b, c) allerdings schlagartig, z.B. auf 100, führt das zu folgendem Problem: Knoten b kennt neben der

direkten Verbindung zu c noch die Route über a und ändert somit seine Minimalkosten zu c auf 3 (zu a und von a zu c). Die geänderten Minimalkosten propagiert b, wodurch Knoten a erfährt, dass die Kosten über b nicht mehr 2, wie vor der Gewichtsänderung, sondern 4 beträgt (die Kosten von b zu c plus die Kosten von a zu b). Somit ändern sich auch die Minimalkosten von a zu c, was a ebenfalls propagiert. Diese Information führt an b allerdings wiederum zu einer Änderung der Minimalkosten auf 5 und einer damit verbundenen Propagierung dieser Information. Die beiden Knoten treiben somit wechselseitig ihre Kosten in die Höhe, bis die Kosten über den direkten Link zu c wieder günstiger sind. Dieses sogenannte *Hochzählen* versucht somit eine hohe Anzahl von eigentlich unnötigen Advertisments. Fällt der direkte Link (b, c) gänzlich aus, steigen die Kosten demnach auf Unendlich, tritt das namensgebende, nie endende *Hochzählen bis Unendlich* auf.

Kreise: Das zweite Problem besteht in dem Auftreten von Routigkreisen, genauer: von Tabellenkreisen. Angenommen, man betrachtet zu einem fixierten Zeitpunkt alle Routingtabellen. Ein *Tabellenkreis* liegt dann vor, wenn bezüglich eines fixierten Ziels d, die LCP-Referenzen (die minimalen Einträge aus den Tabellen) einen nicht-kreisfreien, gerichteten Graphen darstellen, statt bspw. einem In-Tree.

Solche Inkonsistenzen treten z.B. auf, wenn sich noch nicht alle Knoten an eine neue Situation adaptiert haben. Im rechten Teil der Abbildung 5.6 könnte bspw. das Gewicht der Kante (d, b) teurer werden. Angenommen ein Paket mit Ziel c wurde von a zu d gesendet und währenddessen ändert d seine Minimalkosten derart ob der Gewichtsänderung (d, b), dass die Pakete von d zu a – statt über b wie bisher – gesendet werden. Zu diesem fixierten Zeitpunkt existiert somit ein Tabellenkreis, da a über d weiterleitet und d über a. Das Paket, welches an d ankommt, wird folglich sofort zu a zurückgeschickt. Hat a die lokale Tabelle an diese Situation angepasst bis das Paket ankommt, wird das Paket von a über b weitergeleitet. Falls nicht, wird erneut versucht das Paket über d weiterzuleiten. Tabellenschleifen, die in solchen Situationen entstehen, werden auch als *short-lived routing loops* bezeichnet, da sie nur solange auftreten können bis die Nachrichten das gesamte Netz erreicht haben, also maximal $\gamma \cdot d$. Das DVP ist aufgrund des Count-to-Infinity-Problems aber auch anfällig für *long-lived routing loops*. Zwischen den hochzählenden Knoten sind die Pakete nämlich gefangen bis das Maximum (im schlimmsten Fall Unendlich) erreicht wurde. Selbst wenn Tabellenschleifen in einem Protokoll vollständig vermieden werden können, *Routingschleifen* bleiben: Angenommen ein Paket wird von a mit Ziel c über d gesendet und es ädert sich währenddessen das Gewicht von (d, b) derart, dass d über a weitergeleitet würde. Selbst wenn Konsistenz herrschte, a also zu keinem Zeitpunkt weiter über d weiterleitet nachdem d den Next-Hop zu a änderte (für Ziel c), traversiert das Paket Knoten a unvermeidlich ein zweites Mal, da die nun beste Route schlicht darüber führt.

Routingkreise mögen in Computernetzwerken, zumindest in gewissen Grenzen, tolerierbar sein, im Straßenverkehr hingegen würde dies bei Fahrern, die bemerken, dass sie im Kreis geführt würden, zu Ablehnung des Systems führen. Dementsprechend muss dieses Problem für das BeeJamA-Protokoll ausgeschlossen werden.

Um das Count-to-Infinity-Problem und dadurch induzierte Tabellenkreise abzumildern, wurden i. W. vier Erweiterungen für dynamische DVPs, genauer für RIP, entwickelt. Da diese Ansätze auch für BeeJamA relevant sind, seien sie kurz erwähnt: Bei der *Split Horizon-*

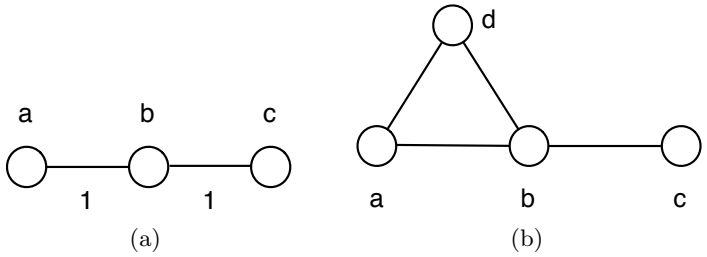


Abbildung 5.6.: Graphen zum Count-to-infinty-Problem.

Technik werden Nachrichten nicht über Links gesendet, über die sie empfangen wurden. Dadurch kann das Count-to-Infinity Problem zwischen zwei benachbarten Knoten beseitigt werden, nicht jedoch wenn längere Kreise im Graph zwischen Knoten existieren. Mittels *Route Poisoning* werden ausgefallene Links unmittelbar im Netz bekannt gemacht und zugehörige Tabelleneinträge unmittelbar gelöscht, wodurch das Count-to-Infinity in diesem Spezialfall abgemildert wird. Statt in einem festen (30sec-)Intervall zu senden, können *Triggered Updates* gesendet werden, wenn bestimmte Ereignisse eintreten. So können sich dynamische Änderungen schneller verbreiten. Auch können *Holddown Timer* verwendet werden, wodurch nach einer Tabellenänderung an einem Knoten eine gewisse Zeit keine weiteren Änderung bzgl. des gleichen Zielknotens durchgeführt werden und somit wieder das Count-to-Infinity-Problem abgemildert wird.

Zwar verhindern diese Maßnahmen Kreise in IP-Netzen in der Praxis häufig, völlig ausschließen können sie sie aber nicht. Um Tabellenkreisfreiheit zu garantieren, muss ein DVP derart erweitert werden, dass die Next Hops stets einen In-Tree bilden bzw. allgemeiner, dass die Tabelleneinträge stets einen kreisfreien zu einem Rootknoten gerichteten Graphen bilden. Das erste Protokoll, welches diese Invariante praktisch (d.h. ohne komplexe zusätzliche Steuerungsmechanismen) erfüllte, ist das *Destination-Sequenced Distance Vector*-Protokoll (DSDVP) [108]. Das 1994 von Perkins und Bhagwat vorgestellte Protokoll nutzt nur zusätzliche Sequenznummern, wodurch Nachrichten von den Ursprungsknoten mit einer eindeutigen Sequenz ausgezeichnet werden. In dem linken Beispiel der Abbildung 5.6 zeichnet der Knoten c die initiale Nachricht bspw. mit der Sequenznummer $c0$, die zweite mit $c1$ usw. aus. Statt Tabellen wird wieder nur ein einzelner Next Hop gespeichert. Dieser wird nur geändert (und somit Nachrichten weitergeleitet), wenn die eintreffende Nachricht die aktuelle oder eine neuere, d.h. diesen Knoten noch nicht erreichte, Sequenznummer besitzt. Die erste Nachricht einer Sequenznummer wird unkonditional zur Änderung des Next Hops verwendet, wodurch sich auch höhere Kosten verbreiten können (die ansonsten aufgrund des Bellman-Ford-Prinzips nicht weitergereicht würden). Später eintreffende Nachrichten der aktuellen Sequenznummer, lösen nur Veränderungen aus sofern sie günstigere Kosten mitbringen. Im Gegensatz zu dem obigen simplen DVP, wird somit nicht selbstständig und unkoordiniert von einem Prozess der Next Hop geändert. Der In-Tree bleibt stets erhalten, weswegen kein Tabellenkreis auftreten kann, da die erste eintreffende Nachricht einer Sequenznummer keinen Kreis durchlaufen hat (sonst wäre sie aufgrund von zu hohen Kosten nicht weitergeleitet worden) und nachfolgende Nachrichten den Next Hop höchstens verändern können, wenn sie über einen günstigeren und damit ebenfalls kreisfreien Pfad den Knoten erreicht haben. Auch tritt ein Count-to-Infinity nicht mehr auf, da die Ursache, das unkoordinierte Ändern des Next Hops, unterlassen wird.

Seit dem DSDVP nutzen üblicherweise alle Routingprotokolle Sequenznummern, so auch

BeeHive und BeeJamA.

5.6. Agententechnologie

Die Intention dieses Abschnitts ist die Einführung und Diskussion der wesentlichen und benötigten Begriffe aus dem Kontext der Multi-Agenten Systeme sowie der Schwarmintelligenz. Es ist bezeichnend, dass drei der vier bisher veröffentlichten verteilten VRGS explizit Bezug zu Schwarmintelligenzkonzepten nehmen (vgl. Abschnitt 2). Ursächlich ist, wie gezeigt wird, die inhärente Dynamik des Schwarmverhaltens.

Zentral in diesem Zusammenhang ist der Begriff des Software-Agenten, oder kurz: *Agent* (Handelnder, von lat. agere, u.a. „handeln, verhandeln“). Wenngleich keine allgemein akzeptierte Definition dieses sehr weitläufigen Begriffs existiert, zielen viele Erläuterungen und Definitionen darauf ab, die *Autonomie des Agenten* zu betonen. Demnach beschreibt ein Agent eine Softwarekomponete, die selbst- oder fremdgesteckte Ziel erreicht, indem sie möglichst selbstständig, in Hinblick auf das Ziel sinnvolle, Umwelt-Interaktionen durchführt. Ein Agent ist dazu mit einer Wahrnehmung ausgestattet und kann seine Umwelt in Grenzen modifizieren. Es existieren vielfältige Klassifikationsmerkmale von Agenten. So kann z.B. zwischen reaktiven und proaktiven Agenten unterschieden werden. Im ersten Fall reagiert der Agent nur auf einen Stimulus, im zweiten werden selbstständig Aktionen geplant und durchgeführt. Ein Agent kann auch lernend sein (auch oft als *kognitiv* bezeichnet), wenn aus zurückliegenden Erfahrungen neues Wissen abgeleitet wird. Daneben existieren viele weitere Attribuierungen, eine Diskussion der Begrifflichkeiten findet sich z.B. in [167]. Eine gängige Beschreibung des Zustandes eines kognitiven Agenten stellt das sogenannte *BDI-Schema* dar, Kurzform für: Beliefs, Desires, Intentions. Die Beliefs stellen das aktuelle Wissen des Agenten dar. Die Desires modellieren die Ziele des Agenten und die Intentions, die dazu ausgewählten Aktionen. Der Agent aktualisiert dazu mittels seiner Wahrnehmung kontinuierlich seine Beliefs, schlußfolgert sodann intern eine Aktion (Intention) zur Erreichung des oder der Ziele (Desires).

5.6.1. Multi-Agenten Systeme und Schwarmintelligenz

In einem Multi-Agenten System (MAS) interagieren mehrere, häufig kognitive, Agenten mit ihrer Umwelt und in der Regel auch untereinander (soziale Agenten), um ein gemeinsames Ziel oder individuelle, ggf. konfligierende, Ziele zu erreichen. Die Agenten des MAS können dabei statisch an einem Knoten (oder seltener einem Link) eines Graphen angesiedelt sein oder mobil sein, d.h. ihren „Aufenthaltsort“ ändern, indem sie sich von einem Knoten zu einem anderen bewegen (sogenannte *Migration*). Im Wesentlichen ist dies ein rein philosophischer Unterschied, denn im ersten Falle kommunizieren Agenten mit festen Aufenthaltsort via Nachrichten, im zweiten Falle werden die notwendigen Informationen durch die migrierenden Agenten (als Teil ihrer Beliefs) selbst verbreitet. Daneben existieren hybride Systeme, die feste und migrierende Agenten kombinieren.

MAS sind prinzipiell als verteilte Systeme aufzufassen, da die einzelnen Agenten nur eine lokale Weltsicht haben, nämlich die durch die Wahrnehmung entstandenen Beliefs und die Veränderungenmöglichkeiten ihrer lokalen Umwelt.

Schwarmintelligenz (SI) [15], auch kollektive Intelligenz bezeichnet, kann im technischen Kontext als ein Spezialfall eines MAS betrachtet werden, in dem soziale, kognitive und häufig mobile Agenten interagieren. Inhaltlich wird dabei auf natürliche Vorbilder rekurriert: das Schwarmverhalten von Vögeln, Ameisen, Bienen, Fischen, Termiten und vielen anderen sozialen

Tieren. Auch hierbei existiert keine einheitliche Definition, eher eine phänomenologische Charakterisierung, nach der *Emergenz durch beobachtbares, heuristisches Individualverhalten* entscheidendes Wesensmerkmal der Schwarmintelligenz ist:

- *Emergenz* (lat. „auftauchen“) bezeichnet das spontane, nicht selten nur schwerlich vorhersagbare, Eintreten von sogenannter Übersummativität. *Übersummativität* (teilweise in diesem Kontext auch als Synergie oder Holismus bezeichnet) wiederum geht auf Aristoteles Metaphysik zurück, in der er feststellt, dass bisweilen „das Ganze mehr ist als die Summe der Einzelteile“[2].
- *Individualverhalten* betont wiederum die Autonomie des einzelnen Agenten.
- Dieses Verhalten ist *heuristisch*, da der einzelne Agent nur anhand von lokalen Wissen agieren kann, eine „optimale“ Globalsicht ist ihm fremd.
- Ferner muss das Verhalten durch die übrigen Schwarmmitglieder *beobachtbar* sein, d.h. sie müssen durch direkte oder indirekte Kommunikation von den autonom getroffenen Entscheidungen erfahren können.
- Dabei kann das Gesamtziel nicht von einzelnen Agenten erreicht werden, sondern ergibt sich aus der Übersummativität. Eine Folgerung daraus ist, dass der einzelne Agent nur einfache Aktionen ausführen kann und begrenzte kognitive Fähigkeiten aufweist. Diese Beschränktheit führt schlussendlich dazu, dass nur das Kollektiv die zu bewältigende Aufgabe lösen kann.

Als Beispiel diene das natürliche Futtersuchverhalten von Ameisen und Honigbienen: Kein einzelnes Tier könnte genug Futter in ausreichendem Maße beschaffen. Vermutlich ist das einzelne Individuum nicht einmal kognitiv in der Lage den gesamten Prozess vorauszuplanen und entsprechend durchzuführen. So umspannt das Gedächtnis einer Ameise gerade einmal zehn Sekunden. Aber auch nicht nur die schiere Masse der Tiere ist das Geheimnis, wenngleich auch nicht zu vernachlässigen. Der übersummative Effekt ergibt sich hingegen aus der Kollaboration. Im Falle der Ameisen bspw. durch die indirekte Phermonkommunikation (als ein Beispiel von *Stigmergie*, der Modifikation der Umwelt). Nachfolgende Ameisen erhalten durch die dynamische Pheromonspur deutliche Hinweise über die günstigsten Pfade zu der Futterquelle. Zwei Ameisen, die jeweils nur einen suboptimalen Pfad zur Futterquelle exploriert haben, können es einer dritten Ameise aber unter Umständen dennoch ermöglichen, den optimalen Pfad zu nutzen. Im Falle von Bienen wird vermöge des sogenannten Schwänzeltanzes – eine nähere Erläuterung wird im folgenden Kapitel noch dargelegt – eine dynamische Zuweisung von Arbeitskraft (Sammelleistung) zu Futterquellen in der Art dezentral zugewiesen, dass ein hoher Ertrag gesichert wird, aber gleichzeitig keine Überlastung einzelner Quellen stattfindet. Im Vergleich zu einer unkoordinierten Vorgehensweise ist der Ertrag höher, da Kundschafterinnen den Sammlerinnen via Tanzsprache Fundstellen mitteilen. Diese Arbeitsteilung erhöht die Effizienz, da nur ein Bruchteil der Arbeiterinnen weit entfernte Futterquellen explorieren müssen und sofort auf zuverlässige Informationen zurückgreifen können. Gleichzeitig signalisieren zurückkehrende Arbeiterinnen, ebenfalls per Tanz, Veränderungen in der Futterverfügbarkeit, wodurch sich ein selbstverstärkendes System ergibt, bei dem die Bienen von untereinander verbreiteten Informationen profitieren.

[2] Aristoteles selbst gibt als Beispiel Silben an, deren Zusammensetzung als Wort substantiell mehr Bedeutung haben als eine reine Aneinanderreihung von Buchstabenketten.

Die ersten technischen Umsetzungen von SI-Verfahren gehen auf Arbeiten von Dorigo aus dem Jahre 1991 zurück und bezogen sich auf die Futtersuche der Ameisen [38].

SI-Verfahren werden häufig auch als Bottom-Up-Ansatz charakterisiert, die zur Selbstorganisation fähig sind. *Bottom-Up* betont die bei verteilten Systemen bereits übliche Eigenschaft eines Systems durch Interaktion verteilter Komponenten den angestrebten Zielzustand zu erreichen. *Selbstorganisation* stellt den Spezialfall dar, dass aus einem ungeordneten Zustand als das angestrebte Ziel ein Gleichgewichtszustand (i.S. eines konvergierten Zustands) angestrebt wird und nach dessen Störung ein Ausgleich gesucht wird. Häufig werden die beiden Begriffe aber ohne diese Diskriminierung und stattdessen synonym verwendet.

Bei den genannten natürlichen Beispiele handelt es sich aufgrund der Abwesenheit einer zentralen Entscheidungsstelle und der lokalen Interaktion der Schwarmmitglieder um Systeme mit Bottom-Up-Eigenschaft. Auch findet Selbstorganisation statt: Angenommen, zu Beginn haben die Schwarmmitglieder noch keine effektive Futtersuchstrategie für die gegenwärtige Situation gefunden. Mittels der, dem Schwarm zur Verfügung stehenden, Explorationseigenschaften werden sodann effektive Strategien dezentral entwickelt. Nimmt man weiter an, dass die Umgebung statisch wäre, würde sich schließlich ein Gleichgewichtszustand der selbstverstärkenden Maßnahmen einstellen. In der Realität ändert sich die Umgebung jedoch dynamisch, so dass stets aufs Neue versucht wird, das System entsprechend auszutarieren, um effektive, an die Veränderung angepasste, Strategien zu finden.

Obschon Multi-Agenten Systeme und SI-Verfahren prinzipiell verteilt konzipiert sind, werden sie nicht immer im Sinne klassischer, geographisch-verteilter Systeme eingesetzt. Stattdessen werden sehr häufig Schwarmintelligenzverfahren als Vorbild für Metaheuristiken zur zentralen Lösung von Optimierungsproblemen eingesetzt. In solchen Fällen sind alle, das Problem betreffende, Daten an einer zentralen Stelle verfügbar. In der Realität agieren Schwärme aber in Umgebungen, in denen die Informationen nicht kostengünstig zentralisiert werden können. Eine Ameise kennt nicht die Pheromonkonzentrationen der gesamten Umgebung und wird sie gleichsam auch nicht in vertretbarer Zeit sammeln können (von der Limitierung des Gedächtnis sogar einmal abgesehen). Analog dazu kennt keine Biene alle Futterquellenqualitäten im Einzugsgebiet des Bienenstocks. Zwar sehen die Implementierungen der Schwarmintelligenz-Metaheuristiken dieses gloable Wissen nicht immer für einen einzelnen Agenten vor, aber prinzipiell wäre es vorhanden. Es ist nicht unmittelbar ersichtlich, weshalb Schwarmintelligenzverfahren anderer Optimierungsverfahren und Metaheuristiken, die diese globale Informationen ausnutzen, konzeptionell überlegen sein sollen. Ihre eigentliche Stärke können Schwarmintelligenzverfahren dafür aber in verteilten, dynamischen Umgebungen ausspielen [117].

Bei vielen SI-Verfahren – allein schon ob der schieren Anzahl – ist kritisch zu hinterfragen, in wie weit substantiell neue Ideen eingebracht werden. Nicht selten, so zumindest der subjektive Eindruck des Autors, handelt es sich um minimale Veränderungen gegenüber klassischen Verfahren, denen nachträglich eine biologische Metapher zur Seite gestellt wurde. Eine Diskriminierung zwischen klassischen Verfahren und SI-Verfahren ist daher mitunter schwer. Aufgrund wenig scharfer Definitionen ist die Bandbreite selbsternannter SI-Verfahren weitläufig. Natürlich darf das BeeJamA-Protokoll nicht von diesem kritischen Diskurs ausgenommen werden. Hierzu wurde bereits vorbereitend die klassischen Grundelemente, die sich abgewandelt in dem Protokoll wiederfinden lassen, nämlich insbesondere der asynchrone Bellman-Ford-Algorithmus und das Distanzvektorprotokoll, ausführlich dargetan.

5.6.2. BeeHive

BeeHive [45] ist ein Routingprotokoll für Computernetze, inspiriert durch das Verhalten von Honigbienen bei der Futtersuche. Dabei wird besonders das geopgraphisch-dichotome Explorationskonzept der Bienen zum Vorbild genommen: Ein Großteil der Kundschafterinnen exploriert die nahe Umgebung, ein geringerer Teil auch weit entfernt liegende Gebiete. Entdeckt eine Kundschafterin eine Futterquelle bzw. kehrt eine Sammlerin zum Bienenstock zurück, wird mit Hilfe eines Bienentanzes der Fundort sowie Qualität kundgetan. Je nach Intensität des Tanzes werden unterschiedlich viele Sammlerinnen auf diese Futterquelle aufmerksam und versuchen an dem kommunizierten Fundort Futter zu sammeln. Das Protokoll bildet dieses Vorbild auf ein hierarchisches Routing mit zwei Hierarchiestufen ab, in dem jeder Netzknoten (Router) über Routingtabellen verfügt, die von den Kundschafterin-Agenten aktualisiert werden. Es wird zwischen zwei Typen von Agenten mit gleicher Funktionalität aber unterschiedlicher Reichtweite unterschieden: Short und Long Distance Bees. Erstere werden von allen Knoten gestartet, letztere nur von einigen ausgewählten Knoten. Über Zielknoten in der näheren Umgebung sind so genauere Informationen vorhanden, als über weit entfernte. Durch diese Stufung kann bei weit entfernten Zielen in die grobe Richtung geroutet werden statt exakt zum Ziel. Der Vorteil liegt in der geringeren Information, die im Netz verteilt werden müssen: Nur eine Teilmenge aller Zielknoten muss inkl. Angaben zur Erreichbarkeit im gesamten Netz bekannt sein.

Technisch handelt es sich bei BeeHive um ein DVP, bei dem alle Knoten den AsyncBF-Algorithmus kontinuierlich ausführen. Die Nachrichten des DV-Protokolls werden durch die Agenten ersetzt. Jeder Knoten emittiert Short Distance Bees, d.h. Agenten mit geringer Reichweite. Zusätzlich werden mittels eines, dem LocalLeaderElection-Algorithmus ähnlichen, Verfahrens sogenannte repäsentative Knoten ausgewählt, welche neben den Short auch Long Distance Bees versenden.

Idee dabei ist, dass wenn das Paket noch weit vom Ziel entfernt ist, zu dem stellvertretenden Leader, dem nächsten repräsentativen Knoten, weitergeleitet wird. Das Paket wird somit in die grobe Richtung des Ziels geschickt. Ist das Ziel näher, wird hingegen ein Forwarding zum direkten Ziel durchgeführt.

Vorteil solch eines hierarchisches Vorgehen ist, dass die notwendige Anzahl der Nachrichten reduziert werden kann gegenüber einer ungestuften Variante.

BeeHive wurde in dem Netzwerksimulator ns/2 gegen andere Routingprotokolle, wie das Ameisen-basierte AntNet [34] und das praktisch eingesetzte Link State-basierte Interiror Gateway Protocol OSPF [98], evaluiert. Es stellte sich heraus, dass BeeHive in den getesteten Szenarien mit höheren Durchsatz, geringeren Prozessorzyklen-Overhead und einer geringeren Anzahl der Agenten abschnitt.

Allerdings weist BeeHive einige Nachteile auf, welche für die Verwendung für das Routing im Straßenverkehrskontext hinderlich sind. Zum Einen setzt BeeHive die Infrastruktur eines Computernetzes voraus, zum Anderen werden Tabellenschleifen nicht ausgeschlossen. Ferner ist es auslastungsinsensitiv und außerdem werden in gewissen Situationen veraltete, oder überhöhte Kosten als Entscheidungsgrundlage verwendet. Auf diese Probleme werden bei der Beschreibung des BeeJamA-Protokolls genauer eingegangen und entsprechende Lösungen dargelegt.

5.7. Komponenten eines VRGS

Vorstehend wurden Routingprotokolle insbesondere am Anwendungsgebiet von (Computer-)-Kommunikationsnetzen erläutert. Im VRGS-Kontext bestehen drei wesentliche Unterschiede:

Trennung zwischen Token- und Kommunikationsnetz: Fahrzeuge befinden sich im Straßennetz, hier verallgemeinernd als Tokennetz bezeichnet, welches prinzipiell über keine Kommunikationsmöglichkeiten verfügt. Daher muss ein zusätzliches Netz ausschließlich zur Kommunikation geschaffen werden, um so ITS-Anwendungen erst zu ermöglichen. Beim Routing in Computernetzen hingegen, sind Token- und Kommunikationsnetz identisch. Die eigentliche Nutzlast als auch die Kontrollnachrichten werden als Pakete in ein und demselben Netz verbreitet.

Unterschiedliche Ausbreitungsgeschwindigkeit: Fahrzeuge im Straßennetz bewegen sich um Größenordnungen langsamer als Pakete in Kommunikationsnetzen. Dies gibt einen zeitlichen Spielraum in Hinsicht auf die Deadline zur Erfüllung einer Routinganfrage.

Menschliche Reaktionen: Statt Datenpakete werden letztlich Menschen gelenkt, wodurch jede Routenempfehlung unmittelbar durch den Nutzer erlebt und subjektiv bewertet wird. Eine Konsequenz daraus ist, dass Kreise vermieden werden sollten.

Der erste Punkt erzwingt eine andere Systemarchitektur eines VRGS im Gegensatz zu einem Protokoll für Computernetze. Der zweite Punkte erlaubt einige Freiheiten beim Design des Protokolls, die der dritte wiederum einschränkt. Diese Punkte werden bei der Vorstellung des BeeJamA-Protokolls wieder aufgegriffen.

Beim Routing in Computernetzen werden Datenpakete mittels Routern geleitet. Im VRGS Kontext lassen sich die folgenden Entitäten identifizieren, die die obige Trennung zwischen den Netzen verdeutlicht:

1. Fahrzeug
2. Straßennetz
3. Kommunikationsnetz, vermöge dessen die nachfolgend genannten Agenten kommunizieren
4. Observation Agent (OA), zur Überwachung von Teilbereichen des Straßennetzes hinsichtlich relevanter Verkehrsdaten (vgl. Abschnitt 3.3.1) und dazu entweder stationär einem Link oder mobil einem Fahrzeug zugeordnet sind
5. Processing Agent (PA), empfängt Auslastungsdaten von den OAs, kommuniziert ggf. mit weiteren Processing Agenten
6. Storing Agent (SA), speichert routingrelevante Knoten- und Link-spezifische Daten
7. Forwarding Agent (FA), beantwortet vom Fahrzeug gestellt Routinganfragen mit einem Teilpfad zum Ziel und greift dazu auf die Daten des Storing Agent zurück

Anhand dieser Komponenten lassen sich drei typische Architekturen konstruieren:

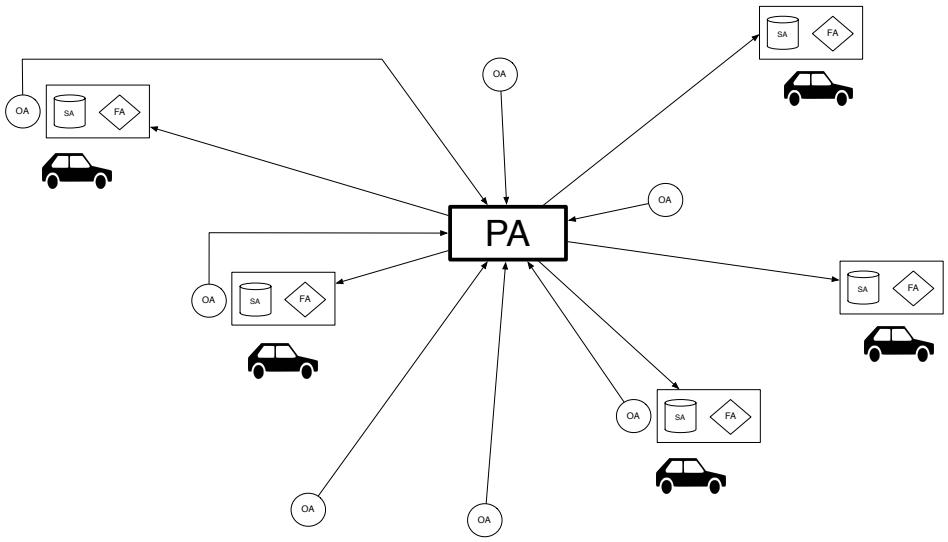


Abbildung 5.7.: Zentralisierte Akkumulation

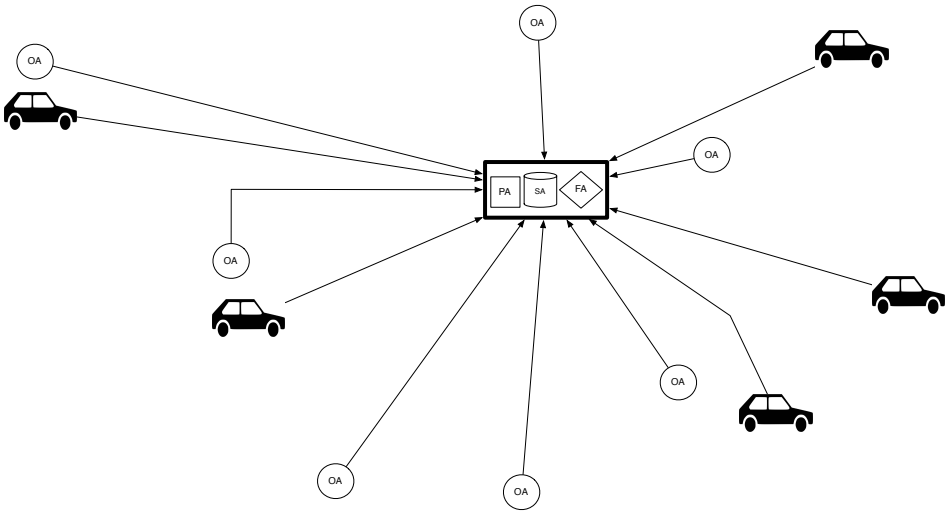


Abbildung 5.8.: Zentralisierte Berechnung

Zentralisierte Akkumulation: Die OAs senden ihre Messergebnisse zu einem zentralen PA, die die zusammengesetzten Daten zu den SAs in den Fahrzeugen sendet (vgl. Abbildung 5.7). Das Processing beschränkt sich hier auf das simple Aktualisierungen von Kantengewichten, welche von den SAs lokal gespeichert werden. Der FA beantwortet Anfragen von einem Verkehrsteilnehmer. Aktuelle kommerzielle VRGS sind nach diesem Schema konzipiert. Es entsteht eine unter den Fahrzeugen unkoordinierte Routenführung, wobei der Flaschenhals die Nachrichtenkomplexität darstellt.

Zentralisierte Berechnung: Neben dem PA ist auch der SA und FA zentral angelegt (vgl. Abbildung 5.8). Alle Fahrzeuge richten ihre Anfragen an diese Zentrale. Der Frank-Wolfe-Algorithmus ließe sich mittels dieser Architektur umsetzen. Zwar entstünde eine koordinierte Routenführung, jedoch dräut neben dem Flaschenhals der Nachrichtenkom-

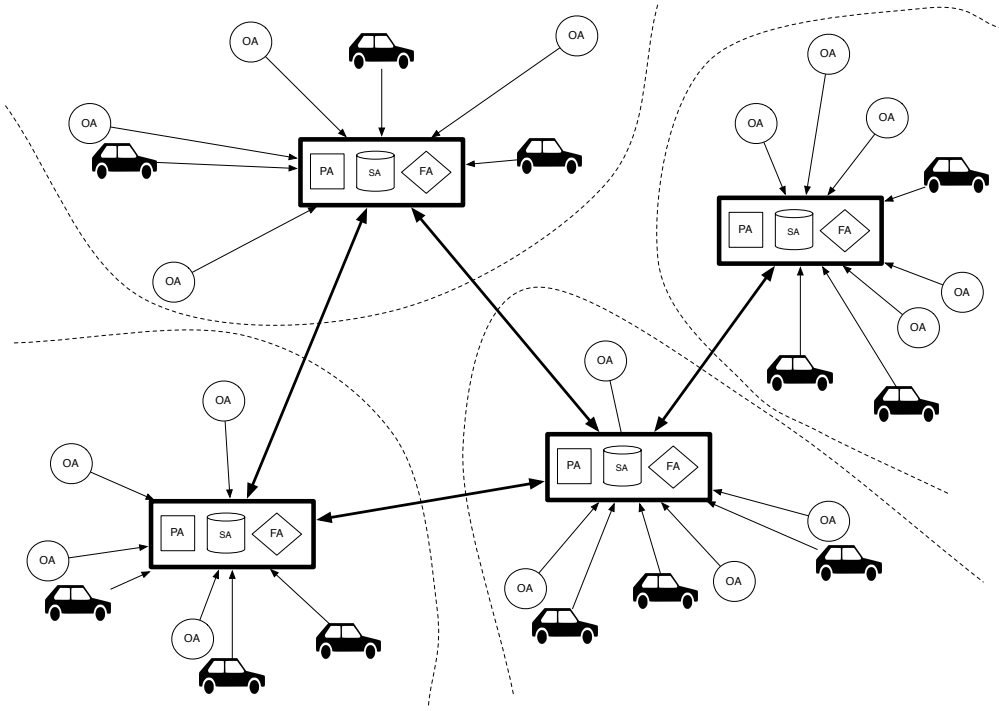


Abbildung 5.9.: Dezentrale Weiterleitung

plexität auch der der Zeitkomplexität: Die notwendige Rechenkraft für die Ausführung des Algorithmus ist für große Netze enorm, weswegen die zeitlichen Deadlines nicht eingehalten werden könnten. Aber auch LCP-Berechnungen ließen sich analog durchführen (wobei dies der ersten Iteration des Frank-Wolfe-Algorithmus entspricht).

Dezentral: Hierbei wird das abgedeckte Straßennetz in lokale Bereiche unterteilt (vgl. Abb. 5.9). Die OA senden ihre Daten an den lokalen PA, ebenso wie die Fahrzeuge ihre Anfragen an den lokalen PA stellen. Die PA sind untereinander verbunden und tauschen Daten aus, um Fahrzeuge über Bereichsgrenzen hinweg weiterleiten zu können. Die dargestellte Architektur ist eine Möglichkeit eines dezentrales VRGS. Die SA und FA könnten auch am Fahrzeug angesiedelt sein, oder mehrere PA teilen sich mehrere SA und FA. Die dargestellte Variante spiegelt das Konzept des BeeJamA-Protokolls wieder.

Die Komponenten beschreiben die notwendigen Abläufe: Beobachtungen der Infrastruktur werden verbreitet, aufgearbeitet und bis zur Forwarding-Entscheidung zwischengespeichert. Je nach exakter Umsetzung des Routingprotokolls sind die Komponenten in unterschiedlicher Art den einzelnen Knoten im Straßennetz zugeordnet.

Prinzipiell handelt es sich bei obigen Komponenten um ein Multi-Agentensystem aus autonomen, stationären Agenten. Auch ließe sich jeder Agent mittels des BDI-Paradigmas beschreiben, jedoch ist die Aufgabenzuteilung – zumindest in dieser abstrakten Darstellung – ziemlich eindeutig. Erst bei genauer Betrachtung einer exakten Architektur ergibt sich die Notwendigkeit einer präziseren Darstellung.

Das im folgenden Kapitel beschrieben BeeJamA-Protokoll verwendet das dezentrale Konzept. Die PA kommunizieren mittels mobilen, sozialen und kognitiven Agenten untereinander, um die Verkehrsdaten der OA adäquat auszutauschen.

Kapitel 6

Das BeeJamA Multi-Agenten System

Inhalt dieses Kapitels ist die Darstellung des BeeJamA Multi-Agenten System (MAS), aufbauend auf den zuvor beschriebenen Grundlagen. Zunächst wird im nächsten Abschnitt die dem MAS zugrundeliegende *Systemarchitektur* vorgestellt und in den anschließenden Abschnitten die MAS-Funktionsweise sukzessive eingeführt. Dabei werden ausgehend von einem *BeeJamA-Basisprotokoll* verschiedene zusätzliche Varianten des Protokolls eingeführt und diskutiert. Ist im Folgenden die Rede von *dem* BeeJamA-Protokoll, ist das Basisprotokoll gemeint. Ausnahmen werden deutlich gemacht. Das nächste Kapitel evaluiert die Protokollvarianten dann empirisch.

In [122] stellen die Autoren eine VRGS-Klassifikation vor, die grundsätzliche, algorithmische Designmöglichkeiten aufzeigt:

- **Statisch vs. dynamisch:** Wie bereits in Kapitel 1 erläutert, werden bei *statischen* VRGS keine veränderten Linkkosten zur Fahrtzeit berücksichtigt. Entweder werden konstante, oder nur einmalig zu Beginn der Fahrt aktualisierte, Linkkosten zur Pfadberechnung genutzt. Viele handelsübliche Navigationssysteme ohne Online-Aktualisierungen sind hierfür beispielhaft. Fahrzeuge bekommen vor Fahrtantritt eine vollständig im Voraus berechnete Route, welcher bis zum Ziel gefolgt wird. *Dynamische* VRGS hingegen berücksichtigen veränderte Linkkosten auch noch während der Fahrt, wodurch es zu Veränderungen der geplanten Route während der Fahrt kommen kann.

- **Deterministisch vs. stochastisch:** Hierbei wird unterschieden, ob die berücksichtigten Linkkosten als deterministisch oder stochastisch betrachtet werden. *Deterministische* Kosten können zwar, nach dieser Klassifikation, dynamisch sein, ergeben sich aber ohne zufällige Einflüsse aus vorherigen Zuständen. Beispiele sind Linkkosten gemäß BPR-Funktion oder einer LPF. Bei der *stochastischen* Modellierung werden Linkkosten als Zufallsvariable aufgefasst.

- **Reaktiv vs. prädiktiv:** *Reaktivität* besagt, dass das Routing anhand von aktuellen Echtzeit-Linkkosten geschieht, im Gegensatz zu vorhergesagten, *prädiktiven* Kosten.

- **Zentralisiert vs. dezentralisiert:** Zentralisierte Systeme sammeln alle Linkkosten an (mindestens) einem zentralen Knoten, dezentrale Verfahren unterlassen dies.

BeeJamA zielt darauf ab, möglichst häufig aktuelle Echtzeit-Linkkosten im Netz verfügbar zu machen. An jeder Kreuzung erhält jedes Fahrzeug eine neue, auf aktuellen Verkehrsinformationen basierende, individuelle Forwarding-Entscheidung. Es handelt sich somit um ein *dynamisches* VRGS. Im Allgemeinen werden stochastische Linkkosten, genauer ein Mittelwert der letzten Linkpassagen, verwendet. Allerdings handelt es sich bei den Linkkosten nur um Zufallsvariablen, sofern entweder der Verkehrssimulator oder das Protokoll Zufallseinflüsse beinhaltet. Da der verwendete MATSim-Simulator allerdings deterministisch funktioniert und die meisten der im folgenden diskutierten BeeJamA-Varianten ebenfalls rein deterministisch arbeiten, handelt es sich streng genommen bei den Linkkosten nicht um Zufallsvariablen.

Das zunächst eingeführte BeeJamA-Basisprotokoll arbeitet vollständig *reaktiv*. In einem weiteren Schritt wird mittels eines Reservierungskonzeptes eine *prädiktive* Komponente hinzugefügt. Um eine möglichst hohe Aktualisierungsfrequenz für möglichst große Netze zu erreichen, arbeitet BeeJamA vollständig *dezentral*.

Der folgende Abschnitt erläutert die notwendige Systemarchitektur.

6.1. Eine Vehicle-to-Infrastructure Architektur als Grundlage

Kommunikation ist in verkehrstelematischen Ansätzen von hoher Bedeutung. So müssen beispielsweise infrastrukturelle Komponenten überwacht und gesteuert werden. Bekanntes Beispiel hierfür sind Lichtsignalanlagen. Jedoch werden verstärkt auch ehemals rein passive Elemente wie Kreuzungen oder ganze Straßenabschnitte in Überwachungs- und Regelkonzepte eingebunden. So existieren Querschnittszähler in Form von Induktionsschleifen und Videoauslastungsüberwachung, wobei die anfallenden Daten klassischerweise zentral gesammelt und ausgewertet werden. Der nächste Schritt in der telekommunikationstechnischen Erfassung des Straßenverkehrs stellt die Einbindung der Verkehrsteilnehmer selbst dar. Im Bereich der ITS haben sich zwei grundlegende Modi herauskristallisiert, wie Kommunikation zwischen Fahrzeugen und ihrer Umgebung umgesetzt werden (siehe auch Kapitel 2):

Vehicle-to-Vehicle-Kommunikation: Bei der Vehicle-to-Vehicle (V2V) Kommunikation tauschen Fahrzeuge lokal in einem Ad-Hoc-Nahbereichsfunknetz Informationen aus. Die Einsatzgebiete dieses Modus umfassen insbesondere Unfallverhütungsmaßnahmen. So können sich bspw. entgegenkommende Fahrzeuge vor temporären Unfall- und Gefahrenstellen, wie Stauenden, warnen. Das Ad-Hoc-Netz muss dabei von mindestens einem Fahrzeug zur Verfügung gestellt werden, wodurch Regelungsbedarf entsteht, wer diese Rolle in ggf. schnell veränderlichen Verkehrssituationen übernimmt. Netzteilnehmer können als Relays agieren und somit das Abdeckungsgebiet vergrößern. Dennoch ist die Abdeckung aufgrund der Reichweite der eingesetzten Funktechnologie beschränkt. Ist die Distanz zwischen zwei Fahrzeugen größer als die Funkreichweite (und befinden sich keine Relays dazwischen), kann keine Kommunikation stattfinden. In der Realität ist es daher schwer, eine verlässliche Kommunikation zu garantieren. Der zweite, folgende Modus kann potentiell Abhilfe schaffen.

Vehicle-to-Infrastructure-Kommunikation: Bei der Vehicle-to-Infrastructure (V2I) Kommunikation existiert eine Kommunikationsinfrastruktur, über welche die Fahrzeuge in Kontakt treten. Üblicherweise handelt es sich dabei um sogenannte *Road Side Units* (RSU), welche am Straßenrand positioniert sind und mittels Nahbereichsfunk mit den Fahrzeugen kommunizieren. Dadurch, dass die RSU statisch positioniert sind und mit

dedizierter Technik ausgestattet werden können (z.B. optimal positionierten Antennen), lässt sich somit eine verlässlichere Kommunikation als im V2V-Modus aufbauen. Daten können, vermittelt über die Kommunikationsinfrastruktur, zwischen Fahrzeugen ausgetauscht werden (wie bei V2V) oder an ein TIC zur zentralen Auswertung geschickt werden. Eine neuere Entwicklung stellt eine rein virtuelle, in der Cloud verortete, Infrastruktur dar. Die Internet-Kommunikation mit den Cloudprozessen wird via Mobilfunk abgewickelt. Dieses Konzept wird als *Vehicle-to-Cloud* (V2C) bezeichnet.

BeeJamA baut auf einer V2I-Architektur auf, deren Infrastrukturkomponenten die sogenannten *Navigatoren* darstellen. Ein Navigator ist dabei für einen räumlich begrenzten Teil des Straßennetzes, sogenannte Bereiche oder *Areas*, zuständig: Fahrzeuge, die sich im Einflussgebiet dieses Navigators befinden, kommunizieren exklusiv nur mit diesem.

Ausgetauscht werden dabei zweierlei Informationen:

1. Eine *Routinganfrage* (Request) von einem Personal Navigation Assistant (PNA) als On-Board Unit eines Fahrzeuges an den lokalen Navigator, als auch die individuelle *Routingempfehlung* (Response) als Antwort auf umgekehrten Wege. Die Response enthält den empfohlenden *Next Hop*.

2. Die *tatsächlich erfahrene Transitzeit* eines Fahrzeugs nach einer Linkpassage, durch den PNA an den lokalen Navigator. So kann der Navigator empirische Linkkosten bestimmen und nachhalten. Hat längere Zeit kein Fahrzeug den Link befahren (ausbleibende Routinganfragen und übermittelte Transitzeiten), sinken die assoziierten Kosten wieder auf die FFT. In [49] werden verschiedene Verfahren vorgestellt, wie aus diesen Informationen auf Linkkosten geschlossen werden kann. Für die im folgenden Kapitel dargestellten Simulationen, wird einfach ein Mittelwert der letzten n rückgemeldeten Transitzeiten gebildet. Dieser Mittelwert steht den Protokollen als Linkkosten ω_l des Links l gemäß ihrer Aktualisierungsintervalle zur Verfügung (vgl. Abschnitt 3.4). Befindet sich kein Fahrzeug auf dem Link (Anzahl der Anfragen minus Anzahl der Rückmeldungen gleich Null), werden die Kosten hingegen auf die FFT zurückgesetzt. Liefern nicht alle Fahrzeuge diese Informationen (unvollständige Durchdringung, vgl. Abschnitt 3.4) sinkt die Genauigkeit der Schätzung. Diese Problematik wird in den folgenden Simulationen weiter untersucht. Im Allgemeinen ist aber davon auszugehen, dass in einer V2I-Umgebung über alle Links hinreichend genaue Linkkosten vorliegen. Ergänzt werden können diese Informationen nämlich durch bereits heute im großen Maßstab eingesetzte Techniken wie Induktionsschleifen, Videodetektion und die Analyse von anonymen Mobilfunkdaten, sogenannte Phone Velocity Data (PVD) [112]. Bei letzterer Technik wird die Geschwindigkeit von Fahrzeugen aus den Differenzen der Einbuchungszeiten eines Mobilfunkgerätes in benachbarte Funkzellen entlang von Straßen abgeleitet. Es stehen somit insgesamt eine Vielzahl von Techniken zur Linkkostenapproximierung, im Folgenden auch als *Bepreisung* bezeichnet, zur Verfügung. *Daher wird im Rest der Arbeit davon ausgegangen, dass für jeden Link solche Kosten vorliegen.* Eine Bepreisung basierend auf empirisch bestimmten Durchschnittskosten wird auch als *Average Cost Pricing* (ACP) bezeichnet. Da das BeeJamA-Protokoll darauf ausgerichtet ist, sehr zeitnah eine Änderungen der ACP-Kosten zu propagieren, sei hierfür der Begriff *Current Cost Pricing* (CCP) geprägt.

Abbildung 6.1 illustriert die angesprochenen Konzepte. Das zugrundliegende Straßennetz ist logisch in Bereiche eingeteilt, in diesem Falle die Areas $A_1, \ldots, A_4$. Jede Area ist wiederum

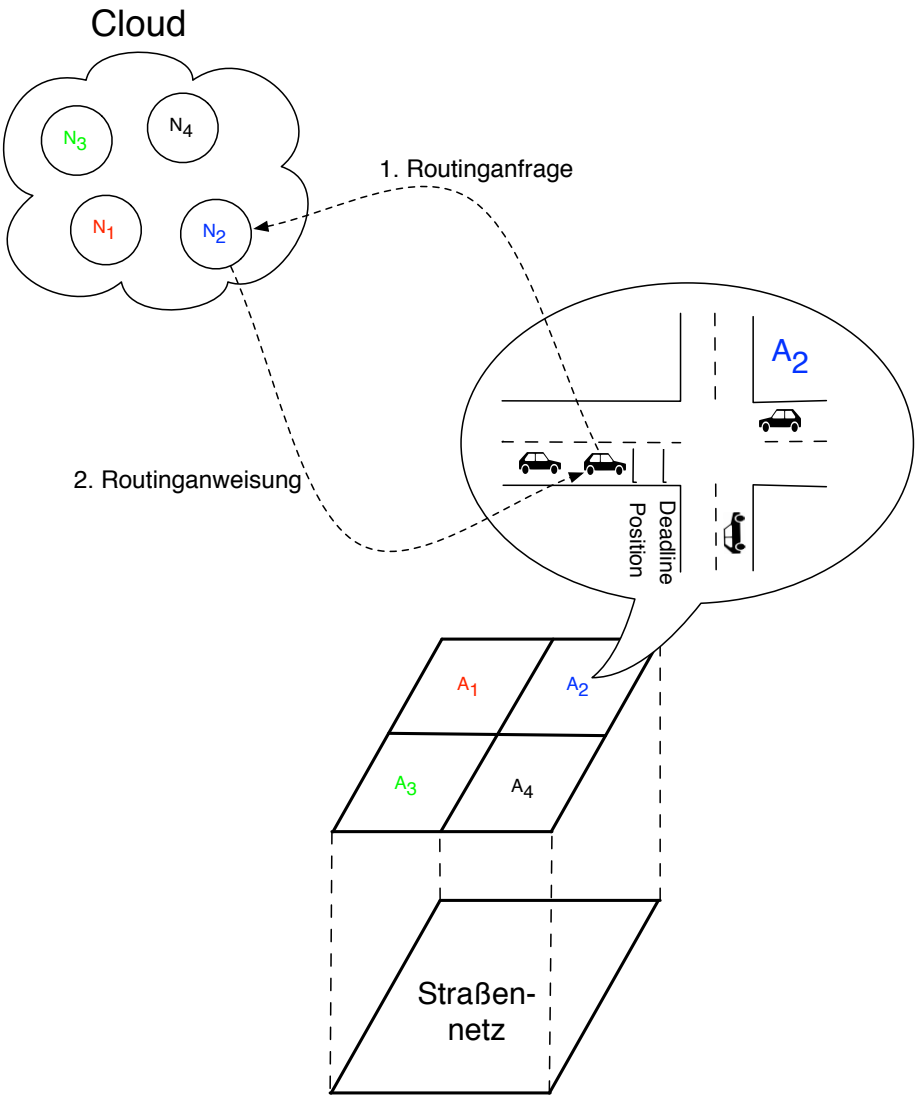


Abbildung 6.1.: Dezentrale Vehicle-to-Infrastructure Architektur

logisch mit genau einem Navigator (hier: $N_1, \ldots, N_4$) assoziiert. Fahrzeuge in einer Area kommunizieren ausschließlich mit dem zugehörigen Navigator. Die Abbildung zeigt beispielhaft die Kommunikation eines Fahrzeugs aus Area 2. Sobald ein Fahrzeug einen Link erreicht, wird die Routinganfrage an den zugehörigen Navigator abgesetzt. Die Routingantwort muss das Fahrzeug rechtzeitig erreichen, so dass dieses sich noch rechtzeitig einordnen kann. Wie groß genau die minimale Distanz bis zur Kreuzung ist, bis zu welcher spätestens die Antwort vorliegen muss, hängt zum Einen statisch von den Eigenschaften der Straße selbst ab und zum Anderen dynamisch von der Geschwindigkeit des Fahrzeugs. So muss auf einer Autobahn die Antwort spätestens mehrere Hundert Meter vor der Ausfahrt vorliegen, im innerstädtischen Bereich erst später. Schnellere Fahrzeuge benötigen die Antwort (temporal) früher als langsamere Fahrzeuge, die noch längere Zeit bis zur Ausfahrt oder Kreuzung benötigen. Zeitgleich ist es aber auch ratsam, die Antwort erst möglichst spät zu geben, so dass noch die neuesten verfügbaren Verkehrsinformationen eingebunden werden können. Zusammengenommen handelt es sich um

ein Problem mit dynamischer Deadline, so dass eventuell sogar Scheduling-Fragestellungen auftreten können. Jedoch ist die Bestimmung eines BeeJamA-Next-Hops, wie noch gezeigt wird, sehr effizient und die reine (Funk-)Übertragungszeit für solche kurze Nachrichten vernachlässigbar, denn es muss ausschließlich die ID des Next Hops übermittelt werden. Daraus folgt, dass Anfragen vom Navigator schlicht der Reihenfolge ihres Eingangs nach abgearbeitet werden können und komplexe Scheduling-Strategien hier außer Acht gelassen werden. Nimmt man eine zeitliche Obergrenze für die Bearbeitungszeit einer Antwort (inkl. Übertragung und Darstellung auf dem PNA) an, dann ist letztlich nur die Distanz bis zur Kreuzung von Bedeutung. Genau diese Problematik ist aber durch aktuelle Navigationssysteme bereits hinreichend gelöst, werden doch passende Informationen, rechtzeitig und je nach Straßentyp angepasst, audiovisuell ausgegeben.

Zusammenfassend sind die Aufgaben eines Navigators für seine Area:

1. Auf eingehende Requests individuelle Responses bestimmen und an den anfragenden PNA zurücksenden.

2. Linkkosten verwalten, mindestens aus den gemeldeten Transitzeiten der Fahrzeuge, ggf. aus weiteren zur Verfügung stehenden Quellen.

3. Ein- und ausgehende Agenten verwalten. Für jeden Knoten in der Area hält der Navigator Tabellen vor und aktualisiert diese entsprechend der Agentenkommunikation (Update). Bzgl. des Sprachgebrauchs in den nachfolgenden Abschnitten wird der Agent das Update durchführen, um die Autarkie des Agenten zu betonen, gleichwohl rein technisch der Navigator diese Änderung vornimmt.

Der Navigator übernimmt demnach nach der Klassifikation aus Abschnitt 5.7 die Aufgaben des Processing-, Storing- und Forwarding-Agenten (PA, SA und FA) gemäß des dezentralen Modells.

Zum Zeitpunkt des Verfassens dieser Arbeit ist weder V2V- noch V2I-Kommunikation für Endkunden flächendeckend verfügbar. Es stellt sich daher die Frage, ob die skizzierte Architektur in absehbarer Zeit überhaupt umsetzbar ist. Diese Frage ist positiv zu beantworten, werden beide Kommunikationsmodi bereits seit längerem erforscht und in Feldtests in der Praxis erprobt. Eine Vehicle-to-Cloud-Kommunikation via Mobilfunk ist ohne spezielle Infrastruktur möglich und wird bereits jetzt bei aktuellen Navigationssystemen eingesetzt, um die Informationen (z.B. über Auslastung und Baustellen) von einem (zentralen) Server aus dem Internet laden. Technisch steht einer signifikanten Verbreitung in der nächsten Dekade nichts entgegen. Auch politisch rückt der Straßenverkehr im Zuge der Diskussionen um Elektromobilität, zugehöriger Infrastruktur und weiterer Themen im Umfeld der Effizienzsteigerung vermehrt in den Vordergrund. Einer Realisierung stehen somit keine technischen, vermutlich auch keine politischen Hindernisse im Weg.

Auch stellt sich die Frage, wie die einzelnen Areas dimensioniert werden sollen. Die *Dimensionierung* ist abhängig von der Rechenkraft, dem zur Verfügung stehenden Speicher, den Übertragungskapazitäten der Navigatoren, und im Falle von RSUs, deren Funkreichweite. Zu berücksichtigen sind die Anzahl der Anfragen durch Fahrzeuge pro Sekunde, sowie das Empfangen, Verarbeiten und Versenden der Agenten. Da ein Navigator zur Berechnung des Next Hops, wie nachfolgend noch gezeigt wird, i.W. nur einige Hashtable-Lookups durchführen muss, entsteht dort i.d.R. kein Engpass. Deutlich aufwendiger ist das Verwalten der Agenten. Je größer ein Bereich wird, desto mehr Knoten enthalten diese und desto rechenintensiver wird

das MAS für diese Area. Prinzipiell kann eine Area so groß werden, wie die zur Verfügung stehenden Ressourcen die Möglichkeit hierzu bieten.

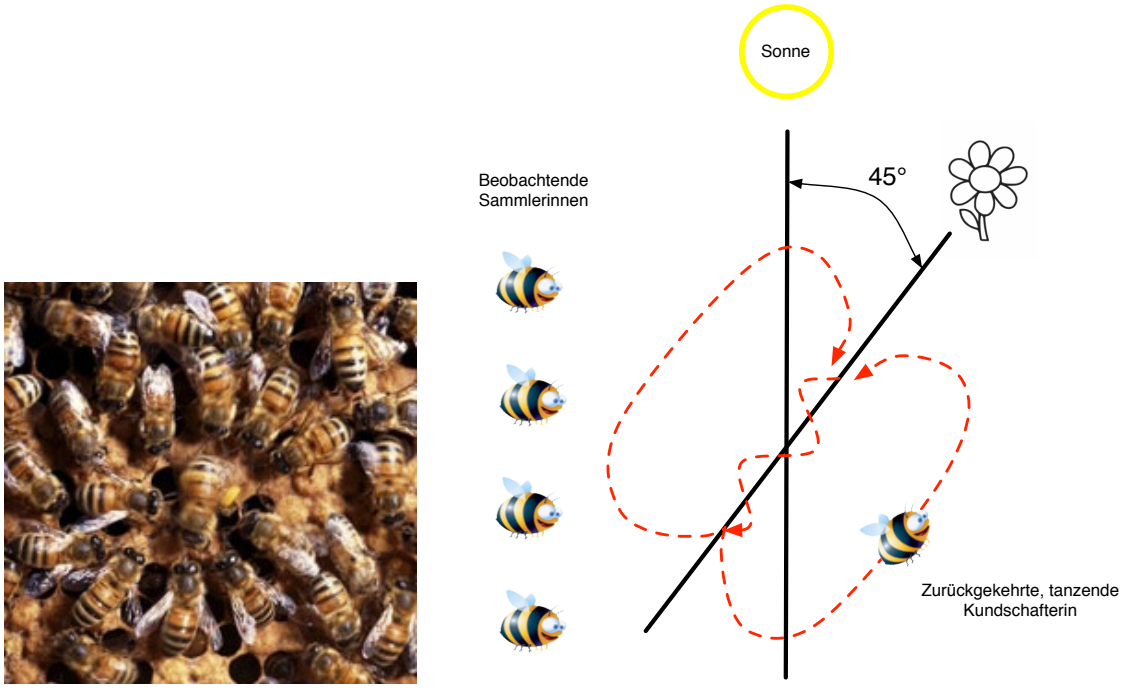
Die latente Gefahr von *Bewegungsprofilen* bei einer Architektur, die regelmäßig positionsbezogene Anfragen übermittelt, ist natürlich stets gegeben. Jedoch ist diese Gefahr nicht höher zu bewerten als bei üblichem Mobilfunk auch; auch dort buchen sich Mobiltelefone stets in GSM-Netze von lokalen Access Points ein. Im BeeJamA-Basisprotokoll muss jedoch lediglich das Ziel und die aktuelle Position an den Navigator übertragen werden, keinerlei benutzerspezifische Information ist notwendig. Werden jedoch Zulassungsbeschränkungen nötig, bspw. zu Abrechnungszwecken, wie prinzipiell bei einer späteren Erweiterung des Basisprotokolls möglich, fallen letztlich und unvermeidlich Informationen beim Systembetreiber an, aus denen Bewegungsprofile erstellbar wären. Im Regelfall haben nur Strafverfolgungsbehörden Zugriff, jedoch sind Spionageattacken und Einbrüche in IT-Infrastrukturen zum Erbeuten von sicherheitsrelevanten Informationen bekanntlich eine realistische Gefahr. Vollständige Sicherheit ist nicht zu gewährleisten, das ist der inhärente Nachteil solcher Ansätze. Jedoch weist diese Architektur nicht mehr Probleme diesbezüglich auf als heute bereits übliche vernetzte Systeme, wie beispielsweise der Mobilfunk.

Der nächste Abschnitt beleuchtet das Schwarmverhalten der Honigbienen genauer, um daraus eine naturinspirierte Adaption ableiten zu können.

6.2. Ethologie der westlichen Honigbiene

Weltweit existieren geschätzt 20.000 verschiedene Bienenarten, mit teils erheblichen Unterschieden in den Verhaltensweisen. So existieren solitäre Arten, deren Artgenossen nur zu Fortpflanzungszwecken zusammenkommen, als auch gregäre Arten, d.h. dauerhaft gesellig lebende Arten. Die umgangssprachliche Bezeichnung „Biene“ meint in aller Regel die *westliche* oder *europäische Honigbiene* (Apis mellifera), die in 25 Unterarten, den Bienenrassen, auftritt. Die Apis mellifera, im Folgenden einfach als Honigbiene oder Biene bezeichnet, zählt zu den wenigen staatenbildenden (und somit gregären), sozialen Bienenarten. Eine umfassende Beschreibung eines einzelnen Individuums, als auch die des Zusammenschlusses zu einem sogenannten Superorganismus als Staat, kann an dieser Stelle nicht geleistet werden. Das Verhalten, die sogenannte *Ethologie*, wird soweit beschrieben, wie es für die Zwecke dieser Arbeit notwendig erscheint. Pionier in der Erforschung des Bienenverhaltes war der deutsche Biologe Karl von Frisch, der 1973 hierfür den Nobel-Preis erhielt. Ein neueres Standardwerk zu diesem Thema liefert Seeley [124], die Angaben dieses Abschnittes entstammen dieser Quelle.

Ein Bienenvolk der Art Apis Melifera besteht typischerweise aus 10.000 bis 60.000 weiblichen Bienen, darunter eine gebährfähige Königin, die Mutter des gesamten Volkes. Die restlichen Bienen sind Arbeiterinnen, jede mit einer Masse von ca. 90mg und einer Lebenserwartung von 50-60 Tagen. Eine Arbeiterin verfügt über zwei ausgeprägte Orientierungssinne. Mittels visueller Wahrnehmung (Sehsinn) kann sie polarisiertes Licht, auch im ultravioletten Bereich, wahrnehmen. Durch Kombination mit dem Sonnenstand kann eine Biene die Himmelsrichtung bestimmen. Zusätzlich ist der Geruchssinn stark ausgeprägt. Es wird davon ausgegangen, dass im Nahbereich diesem besondere Bedeutung zu kommt. Im Frühsommer existieren zusätzlich 500-2000 männliche Drohnen, deren einzige Funktion die Begattung der Jungköniginnen ist und für diese Arbeit keine Relevanz haben. Nach einer kurzen Duldungsphase werden sie von den Arbeiterinnen vertrieben oder gar getötet (die sogenannte Drohnenschlacht). Im Folgenden wird daher nur die grammatikalisch weibliche Form benutzt.



(a) (b)

Abbildung 6.2.: Tanzende Biene

Keine Biene (der Art Apis mellifera) kann solitär überleben. Sie ist stattdessen gänzlich auf die soziale Interaktion mit ihrem Bienenvolk angewiesen. Wie in Abschnitt 5.6.1 bereits diskutiert, hat sich für emergentes Verhalten, resultierend aus einfachen Regeln befolgenden Individuen, die Begrifflichkeit der Schwarmintelligenz eingebürgert. Zu den wesentlichen Aspekten im Leben eines Bienenvolkes, die durch solche sozialen Interaktionen geprägt sind, gehören: Futtersuche, Nestbau und Nestsuche, Brutpflege, Temperaturregulierung und Feindabwehr.

Der Fokus dieser Arbeit liegt auf der Übertragung von Konzepten aus der Futtersuche, weswegen dieser Aspekt ausführlicher betrachtet wird.

Das Leben einer Arbeiterin ist grob in mehrere Abschnitte aufgeteilt. In den ersten Tagen fällt ihnen die Nahrungsproduktion im Stock für die Larven zu, danach der Wabenbau, anschließend die Bewachung des Flugloches (der Ein- und Ausgang des Bienenstocks) und schließlich die Futtersuche. Reguliert wird die Transition erstens durch das Alter der Arbeiterinnen und zweitens durch die Konzentration eines Pheromons, welches von der Königin emittiert wird und in ihrer Nähe in höchster Konzentration vorliegt. Aufgabe des Pheromons ist die Unterdrückung von aggressivem Verhalten. Anders ausgedrückt, solange die Biene noch mehrheitlich Arbeiten im Stock verrichtet, verhält sie sich ruhiger als in der Lebensphase als Futtersammlerin außerhalb des Stocks. Dies ist wiederum ist ein klassisches Beispiel von Schwarmintelligenz, vergleichbar mit dem Konzept der Stigmergie-Kommunikation [74], bei welcher nicht direkt, sondern durch Veränderung der Umwelt dezentral kommuniziert wird.

Das für diese Arbeit entscheidende Schwarmverhalten ist jedoch die Futtersuche. Grundlage dieses Verhaltens ist der Energievorrat: die des einzelnen Individuums und besonders auch des Volks. Fällt nämlich der Futtervorrat des Stocks unterhalb eines Schwellwertes, fliegen vermehrt Arbeiterinnen aus, um neue Futterquellen aufzutun. Arbeiterinnen, die auf der

Suche nach Futterquellen sind, werden als *Kundschafterinnen* bezeichnet. Für die Suche ist der individuelle Energievorrat einer Kundschafterin ein limitierenden Faktor, da der Flug kräftezehrend ist und mit vollem Honigmagen eine Flugdistanz von höchstens 8km überbrückt werden können. Da sich allerdings durch Hin- und Rückflug über maximale Distanzen im Zweifelsfall nur ein geringer Nettoenergiegewinn erzielen lässt, ist der Aktionsradius in den meisten Fällen geringer, nämlich im Bereich eines Kilometers. Jedoch gibt es auch immer eine kleinere Anzahl von Bienen, die in Hoffnung auf besonders ertragreiche Futterquellen diese Maximaldistanzen zurücklegen. Ist eine Futterquelle entdeckt, unabhängig von der Entfernung zum Stock, memoriert die Biene den groben Fundort, kodiert durch die Angabe des Flugwinkels (welcher durch das oben beschrieben Sehvermögen im ultravioletten Bereich ermittelt werden kann) ausgehend vom Bienenstock. Ferner werden je nach Quelle Pollen an den Hinterbeinen und/oder Nektar im Honigmagen als Futterprobe deponiert und anschließend der Rückflug angetreten. Auf dem Hin- als auch Rückweg umfliegt die Kundschafterin selbstständig eventuell auftretende Hindernisse. *Der dadurch notwendige, zusätzliche Umweg wird beim Abschätzen der Gesamtwegstrecke zwischen Stock und Futterquelle jedoch intentional ausgelassen.* Eine erhebliche kognitive Leistung, wird so doch nur die Luftlinie berücksichtigt, die im Folgenden als *Distanz* bezeichnet wird. Die Flugdauer wird hingegen nicht „begradigt“.

Im Stock angekommen, begibt sich die Kundschafterin zu einem als *Tanzboden* (dance floor) bezeichneten Bereich des Stocks nahe des Eingangs und führt dort zum Zwecke der Kommunikation einen Bienentanz, genauer gesagt einen sogenannten *Schwänzeltanz* (waggle dance) auf. Durch Tanzmuster, Frequenz und Intensität des Tanzes kommuniziert die Kundschafterin Informationen bzgl. des Fundorts (Distanz, Winkel) und der Qualität der Quelle. Es wird demnach die Distanz annonciert, nicht der von der Kundschafterin tatsächlich zurückgelegte Pfad. In die subjektive Qualitätsbewertung einer Futterquelle durch eine Kundschafterin gehen Faktoren ein wie Zuckergehalt, Reinheit, Menge, Zugänglichkeit, Viskosität und Flugdauer mit ein. Insgesamt ergibt sich eine *Abschätzung der Energiebilanz* einer Quelle. Die Qualitätsabschätzungen nachfolgender Kundschafterinnen bzgl. einer Quelle weisen aber ein gewisse Varianz auf, besonders, da der Rückflug von den Bienen selbstständig über ggf. unterschiedliche Strecken um Hindernisse herum ausgeführt wird. Dementsprechend ändert sich die Bilanz einer Quelle: dauert der Flug über die gewählte Strecke länger, ist der Nettoenergiegewinn geringer und umgekehrt. Verschiedene Kundschafterinnen melden demnach ungefähr gleich große Distanzen und Winkel zu ein und derselben Quelle, aber, je nach Rückflugstrecke, unterschiedliche Qualitäten. Auch wenn die Futterquelle sich in der Zwischenzeit nicht geändert hat.

Währenddessen beobachten unbeschäftigte Arbeiterinnen, sogenannte *Sammlerinnen*, den Tanzboden und wählen gewichtet nach der kommunizierten Qualität eine Futterquelle. Dieser Prozess wird als *Rekrutierung* bezeichnet: gute Futterquellen locken mehr Sammlerinnen an als schlechte. Eine Sammlerin extrahiert die kodierten Informationen des Tanzes, um die Futterquelle grob aufspüren zu können: die Richtung (in Form eines Flugwinkels), sowie die Distanz. Um diese groben Angaben später im Zielgebiet präzisieren zu können, läuft die Sammlerin der tanzenden Kundschafterin hinterher und streift dabei etwas von der Futterprobe ab, um den Geruch aufnehmen zu können. Die Sammlerin fliegt sodann in die grobe Richtung der Futterquelle, d.h. gemäß des Flugwinkels über die angegebene Distanz. Im Zielgebiet angekommen, lokalisiert die Sammlerin die Futterquelle anhand ihres präzisen Geruchssinn exakt.

Zurückkehrende Sammlerinnen führen, genau wie die Kundschafterinnen initial zuvor, ebenfalls einen Schwänzeltanz auf, um somit die ggf. veränderten Bedingungen bekannt zu

machen. Die temporalen Veränderungen mögen aus natürlichen Schwankungen resultieren, oder auch aus dem Sammelverhalten der Sammlerin selbst, die bis 75% ihres Körpergewichts an Futter mit zurückzubringen vermag. Werden zeitliche Veränderungen seitens der Sammlerin bemerkt, wird wie folgt reagiert: steigt die Qualität einer Quelle mit der Zeit, erhöht sich die Intensität des Tanzes; sinkt sie hingegen, wird dies durch einen langsameren Tanz zum Ausdruck gebracht. Nachfolgende, Tanzboden beobachtende, potentielle Sammlerinnen ignorieren ab einer bestimmten Intensitätsuntergrenze die Tänze, so dass keine zusätzliche Rekrutierung stattfindet. Dies ist ein durchaus entscheidender Punkt, denn nicht die tanzenden Bienen können autoritär potentiellen Sammlerinnen *befehlen* eine bestimmte Futterquelle anzufliegen. Stattdessen wird nur ein Angebot gemacht. Die unbeschäftigten Sammlerinnen beobachten und vergleichen stattdessen die Angebote – im Regelfall tanzen mehrere Bienen gleichzeitig oder zumindest in kurzen Abständen – und wählen sodann eine Quelle, entsprechend den aktuellen Bedürfnissen des Volkes. Stellt eine Arbeiterin im Stock fest, dass Mangel an einem speziellem Futtertyp (Wasser, Pollen, Nektar) herrscht, so erhöht sich ihre Sensitivität für Angebote dieses Typs. Die Selektion einer Quelle findet dabei wohl stochastisch und gewichtet nach der kommunizierten Qualität statt.

Neben der Futtersuche existieren weitere Formen des Schwarmverhaltens, von denen besonders der sogenannte *Schwarmtrieb* relevant ist. Es ist ein natürliches Bestreben des Staates, im Frühsommer eine Teilung des Volkes herbeizuführen. Zu diesem Zeitpunkt sind bereits neue Königinnen herangezogen worden. Einige Tage vor dem Schlüpfen dieser, verlässt die alte Königin mit einem Teil des Schwarms (über 10.000 Bienen) als sogenannte Schwarmtraube den alten Bienenstock, mit dem Ziel einen neuen Nistplatz zu finden. Dazu pausiert die Schwarmtraube nach Verlassen des Stock regelmäßig und sendet Kundschafterinnen, sogenannte *Spurbienen*, aus, um in der nahegelegenen Umgebung einen geeigneten, neuen Nistplatz zu finden. Hat eine Kundschafterin einen Nistplatz gefunden, kehrt sie zur Schwarmtraube zurück und annonciert diesen sogleich, wiederum mit dem Schwänzeltanz. Auch hierbei wird die Qualität und Distanz des potentiellen Nistplatzes durch Intensität und Anzahl der Wiederholungen ausgedrückt. Durch die Tänze werden andere Kundschafterinnen ebenfalls auf bessere Nistplätze aufmerksam und erkunden diese verstärkt. So konvergieren über die Zeit die Kundschafterinnen auf einen gemeinsamen Nistplatz. Die Schwarmtraube folgt ihnen dorthin, indem die Kundschafterinnen vorausfliegen, sich am Rande der Traube zurückfallen lassen und von hinten nach vorne durch Traube fliegen und die richtige Richtung weiter vorgeben. Schließlich erreicht die gesamte Traube, inkl. Königin, eventuell über mehrere Zwischenrasten, den neuen Nistplatz. (Dieses Spurbienen-Verhalten kann als eine Konvergenz zu einem Optimum interpretiert werden und ist nicht unähnlich zu dem Grundgedanken der Particle Swarm Optimization [110].)

Auch wenn die zuvor beschriebenen Verhaltensweisen in der Realität deutlich komplexer sind, als hier dargelegt werden kann, wird deutlich, dass das Verhalten nicht zentral gelenkt wird. Es existiert im Bienenstock kein Biene, die über alle, den Schwarm betreffende, Vorgänge überhaupt informiert wäre. Das Gehirn einer Biene wäre auch gar nicht in der Lage, einen Schwarm aus vielen zehntausend Individuen zu planen und zu steuern. Hingegen ergibt sich ein instinktgeleitetes, emergentes Schwarmverhalten des Superorganismus allein aus den Stimuli und Reaktionen der einzelnen Schwarmmitglieder. Eine Bewertung ihrer Umgebung findet allein anhand der jeweils lokal wahrgenommen Temperatur, Geruch (Pheromone und Futtergeruch), Geschmack (Süße und Reinheit des Nektars) und visuellen Eindrücken (Tanz, Aussehen und Himmelsrichtung) statt. Auf einen Sinnesstimulus erfolgt eine instinktgeleitete Reaktion, ein bewusstes Planen oder gar Hinterfragen geschieht nicht – ist aber für den Erfolg des Volks

auch nicht notwendig.

6.3. Konzeptübertragung der Futtersuche

Kontinuierlich passen sich Bienen ihrer dynamischen Umwelt an, indem eine Zuweisung von verfügbarer Arbeitskraft zu geographisch entfernten Futterquellen in verteilter Weise geschieht. Dieses Verhalten ist Grundlage der VRGS-Konzeptübertragung dieses Abschnitts.

Die Entitäten und Konzepte des natürlichen Systems werden wie folgt interpretiert:

Bienenstock: Jeder potentielle Startknoten eines Fahrzeugs im (Straßennetz) wird als ein Stock interpretiert. Somit ist mit jedem Knoten ein Bienenstock verknüpft.

Futterquelle: Jeder potentielle Zielknoten entspricht einer Futterquelle. Mit jedem Knoten (im Routinggraph) wird daher auch eine Futterquelle assoziiert.

Kundschafterin: Im natürlichen Vorbild schwärmen Kundschafterinnen vom Stock aus, erkunden Futterquellen und kehren mit den Ergebnissen sodann zurück. In der Übertragung wird nur der zweite Teil umgesetzt, die Rückkehr zum Stock. Ausgehend von ihren (Ziel-)Knoten (interpretiert als Futterquellen), beginnen sogenannte *Scout*-Agenten ihren Rückflug. Das Aussenden von Kundschafterinnen in der Natur bewirkt, dass in allen Stöcken in der Umgebung die Quelle letztlich bekannt sein wird. Zu jedem Stock in Explorationsreichweite, muss daher in der technischen Umsetzung (mindesteins ein) Scout gesendet werden. Am leichtesten lässt sich dies mittels Flooding erreichen (vgl. Abschnitt 5.1). Dieser Prozess wird in der Umsetzung kontinuierlich widerholt, so dass jeder Knoten somit regelmäßig Scouts flooden wird. Dabei wird zwischen *Short* und *Long Distance* Scouts unterschieden. Analog zum Vorbild, in dem ein geringer Teil der Kundschafterinnen weiter fliegt als der Großteil, haben Long Distance Scouts ein größeres *Hop Limit* (Reichweite in der Anzahl der Knoten des zurückgelegten Pfades). Unabhängig von der Länge des Rückflugs (Short oder Long), wird die Flugzeit abgebildet auf die Fahrzeit eines Fahrzeugs auf dem Pfad hin zur Futterquelle.

Tanz: Erreicht ein Scout seinen Stock (von dem dieser ursprünglich losgeschickt worden wäre), teilt dieser seine Erkenntnisse mittels einer Aktualisierung der Routingtabelle, was dem Aufführen des Schwänzeltanzes in der Natur entspricht. Aber nur wenn die einzutragende Fahrzeit gewissen Mindestanforderungen entspricht, analog zu der Überprüfung der Vorkosterin, darf eine Aktualisierung auch durchgeführt werden. Durch die Tabelleneintragung wird neben der Fahrzeit zusätzlich die Richtung zum Ziel bekannt gemacht. Dies ähnelt zwar dem natürlichen Vorbild, gleicht diesem aber nicht gänzlich. Melden in der Natur alle Kundschafterinnen die gleiche Richtung zur Quelle, wird in der Umsetzung als Richtung genau der Vorgängerknoten auf dem Pfad des Scouts von der Quelle zum Stock bezeichnet. Da unterschiedliche Scouts auf unterschiedlichen Pfaden zum Stock zurück finden können (aufgrund des Floodings), können mehrere Richtungen zum Ziel bekannt sein. Da aber auch an jedem Vorgängerknoten die Quelle bekannt ist, existiert dort ebenfalls (mindesten eine) Richtungsangabe zu dieser Quelle. Es entsteht so (mindestens) eine durchgehende Kette von Next Hop-Verweisen vom Stock zur Quelle auf dem Pfad des Scouts (in umgekehrter, d.h. Downstream-Richtung).

Tabelle 6.1.: Konzeptabbildung der Futtersuche

Natürliches Vorbild	Vehicle Route Guidance
Bienenstock	Knoten
Nektarquelle	(Ziel-)Knoten
Kundschafterin	Scout-Agent
Sammlerin	Forager-Agent
Explorationsflug	Flooding
Vorkosterin	Vorabprüfung der Pfadqualität
Tanzboden	Routingtabellen
Tanz	Tabellenaktualisierung / Update
Rekrutierung	Next Hop-Selektion
Tanzsprache (Richtung)	Next Hop
Tanzsprache (Qualität)	Aggregierte Transitzeiten
Tanzsprache (Distanz)	Entfernung in Hops
Nahbereich des Stocks	Foraging Zone
Weitbereich des Stocks	Foraging Region

Sammlerin: Sammlerinnen werden in der Umsetzung als *Forager* bezeichnet und entsprechen Fahrzeugen. Forager sind rein imaginär und dienen rein der Komplettierung der Abbildung des natürlichen Systems: es existieren keine Agenten die tatsächlich erzeugt oder versendet werden (zumindest im Basisprotokoll). Forager bewegen sich mit identischer Geschwindigkeit wie die Fahrzeuge und bilden diese (zumindest imaginär) auf der logischen Agentenebene ab. Der Zielknoten d der Foragers (die Futterquelle) entspricht in der Übertragung dem Ziel d des Fahrzeugs.

Rekrutierung: Wie Sammlerinnen anhand aus dem Tanz extrahierter Informationen vom Stock zur Quelle finden, werden Forager vom Startknoten über die zuvor genannten, von den Scouts in die Tabellen eingetragenen, Richtungsangaben zur Futterquelle d geleitet. An jedem in Kürze erreichten Knoten i, wird eine Rekrutierung durchgeführt. Der Forager wählt aus den Einträgen der Routingtabelle einen Next Hop s und reicht diesen an den PNA des Fahrzeugs (rechtzeitig) weiter, wo dieser als Routingempfehlung visualisiert wird. Ähnlich wie Scouts bestreiten Forager in dieser Umsetzung nur eine der beiden Flugrichtungen, hier vom Stock zur Futterquelle, also vom Start- zum Zielknoten. Zurück kehren sie in dieser Umsetzung nicht.

In der Natur wählen Sammlerinnen aus dem annoncierten Angebot entsprechend der Qualität. In der Übertragung muss zusätzlich beachtet werden, dass der Forager, im Gegensatz zur natürlichen Sammlerin, ein durch das Fahrzeug vorgegebenes, fixes Ziel, die Futterquelle, hat. Es können demnach nur Tabelleneinträge zum oder zumindest in die Richtung des Ziels beachtet werden.

Nah- und Fernbereich: Die dichotome Reichweitenunterscheidung im natürlichen Vorbild,

umgesetzt durch Short und Long Distance Scouts, ist Grundlage eines Hierarchiekonzepts zur Reduzierung der (absoluten) Anzahl an versendeten Scouts (und damit Nachrichten). Im Nahbereich, *Foraging Zone* genannt, werden ausgehend von vielen Quellen Scouts verbreitet; im Fernbereich, *Foraging Region* genannt, verbreiten weniger Quellen Scouts. Im Ergebnis liegen für weit entfernte Zielbereiche weniger präzisere Informationen vor, als für nahe. Dies ist vergleichbar zu der Situation, in der Sammlerinnen zunächst nur grobe Informationen besitzen (Richtung & Entfernung) bis mittels des Geruchssinns präzisere Informationen genutzt werden können.

Die zwischen Kundschafterin und Sammlerin auf dem Tanzboden stattfindende Kommunikation, entspricht in der Übertragung demnach dem Lesen und Schreiben von Routingtabelleneinträgen. Ein Tanz wird jedoch nur von heimkehrenden Scouts dargeboten, wenn die Vorkosterinnen aufgrund genügend hoher Qualität hierzu auffordern, was in der technischen Umsetzung einem Vergleich mit bereits vorhandem Wissen gemäß Tabelle bzgl. der Kosten zu einem Ziel gleichkommt. Nach erfolgter Aktualisierung können Forager, die avisierten Richtungs- und Qualitätsangaben aus den Tabellen bewerten und je nach Ziel des Foragers (und damit dem Ziel des Fahrzeugs) einen entsprechenden Tabelleneintrag als Grundlage einer Forwardingentscheidung wählen.

Die durch die Scouts vermittelte Attraktivität einer Futterquelle hängt im Straßenverkehr von der aggregierten Transitzeit, also der Gesamtfahrzeit, auf dem zurückgelegten Pfad des Scouts in umgekehrter Richtung (zur Futterquelle hin) ab. Statt wie im Vorbild die (Nektar-) Eigenschaften der Futterquelle zu bewerten, wird in der technischen Umsetzung hingegen die Gesamtqualität einer Quelle durch Eigenschaften des zurückgelegten Pfades determiniert.

Die vorstehenden Konzepte werden in Form eines mehrschichtigen Distanzvektorprotokolls umgesetzt, bei dem Scouts Pfade entdecken und Tabellen aktualisieren. Forager, die die Fahrzeuge auf logischer Ebene abbilden, werden anhand der Tabelleneinträge von Hop zu Hop und schließlich zum Fahrzeugziel geleitet. Tabelle 6.1 gibt eine Zusammenfassung der Konzeptabbildungen.

Diskussion der Übertragung

Die Übertragung der Futtersuche auf den Kontext des Routings weist eine fundamentale Lücke auf: Bienen kennen kein Konzept von alternativen Pfaden. Zurückkehrende Kundschafterinnen kommunizieren immer die Luftlinie als Entfernung zur Futterquelle, nicht die der zurückgelegten, zickzackförmigen Strecke. Unterschiedliche Kundschafterinnen und Sammlerinnen unterscheiden sich beim Tanz i.W. nur durch die in der Qualität inkorporierten Flugzeit zur Futterquelle, die sie selbst erlebt haben. Da Hindernisse auf dem Weg zur Quelle von den Arbeiterinnen selbstständig, ohne Informationen anderer Artgenossinnen, umflogen werden, kann die rückgemeldete Flugzeit von Biene zu Biene variieren. Die gemeldete Richtung hingegen bleibt konstant. Bienen kennen somit in ihrer Kommunikation keine alternativen Pfade um Quellen zu erreichen. So ergeben sich deutliche Unterschiede zwischen Routing und Vorbild: 1) Scouts kommunizieren die Richtung aus der sie selbst kamen, statt die Richtung der Luftlinie zur Futterquelle. 2) Forager orientieren sich an jedem Knoten basierend auf den Erfahrungen der Scouts neu und wählen dementsprechend einen Next Hop, statt nach dem Verlassen selbstständig einen Pfad ohne fremde Hilfe zu finden. Besonders besuchen Sammlerinnen dazu keine anderen, fremde Stöcke, wie Forager, die mit jedem Knoten auf dem Pfad zum Ziel auch einen anderen Stock besuchen. 3) Sammlerinnen haben kein vorgegebenes

Ziel, stattdessen wählen sie ein Ziel aus den annoncierten Futterquellen zufällig, entsprechend gewichtet nach dessen Qualität. Forager hingegen haben ein vorgegebenes Ziel aufgrund des Zielwunsches des Fahrers. 4) Die Bienenkommunikation vermöge des Schwänzeltanzes dient eigentlich der Arbeitsaufteilung, nicht der Pfadzuweisung von einem Start- zu einem Zielknoten in der technischen Umsetzung. Nicht die Lenkung von Arbeiterinnen über optimale Pfade hin zu Futterquellen steht in der Natur im Vordergrund, sondern die Anzeige der dynamischen Änderung der Effizienz des Futtersammelns.

Da das Narrativ der Futtersuche sich, ob dieser strukturellen Unterschiede, nur bedingt auf das Problem des Routings übertragen lässt, wurden unterschiedliche Brücken geschlagen. In BeeHive bspw. bewegen sich Scouts, anders als in vorstehender Übertragung, nicht von einer Futterquelle zum Ursprungsstock. Stattdessen fliegen Scouts von Stock zu Stock und verbreiten die Paket-Transmissionszeiten in Computernetzen zwischen den Stöcken. Ein Verhalten, das keinenfalls natürlichen Ursprungs ist.

Ergänzung der Übertragung

Obschon die Konzeptübertragung dieser Arbeit versucht, möglichst nahe am natürlichen Vorbild zu sein, treten vorgenannte Diskrepanzen zum Vorschein. Insbesondere die Eigenschaft der Scouts, an jedem Knoten ggf. erneut einen „Richtungswechsel“ durchzuführen, lässt sich im Vorbild der Futtersuche nicht belegen. Während des Schwarmtriebs hingegen ist solch ein Verhalten durchaus beobachtbar: Während einer Rast werden erneut Informationen ankommender Kundschafterinnen inkorporiert und die Richtungsentscheidung dementsprechend gefällt. Die Schwarmtraube bzw. die Königin wird so über Etappen hin zum Ziel geleitet. Man könnte folgende Übertragung wählen: Ein Königin-Agent traversiert den Graph und erhält an jedem Knoten, entsprechend einer Rast, aufgrund der Kollaboration zwischen Kundschafterinnen und Spurbienen eine neue Richtungsanweisung.

Dieser Ansatz erklärt zumindest die kontinuierliche Neuorientierung, erlaubt damit das Konzept von Alternativpfaden und kennt Hierarchie. Im Folgenden wird der Einfachheit halber dennoch weiter von der tradierten Scout-Forager-Kollaboration wie in BeeHive die Rede sein.

6.4. Grundlegende Agenten-Funktionsweise

Das BeeJamA-Protokoll wird in dieser Arbeit unterteilt in ein MAS-Basisprotokoll und zusätzliche Erweiterungen. Dieses *Basisprotokoll* wird in diesem und den folgenden drei Abschnitten ausgearbeitet. Die nachstehende Beschreibung diskutiert schrittweise die einzelnen Komponenten des Basisprotokolls und verdeutlicht so die Modularität des Ansatzes. Nachfolgende Erweiterungen des Basisprotokolls fügen sich durch Erweiterung oder Austausch einiger Komponenten harmonisch in das Gesamtkonzept ein. Einige zentrale Komponenten werden durch *Routingoperationen* (kurz: Operation) dargestellt, wobei jede Operation einen genau definierten Teil im MAS-Ablauf darstellt.

Dies stellt eine Verallgemeinerung des Operatorenkonzepts dar, wie es der Metarouting-Ansatz [57] popularisiert hat. Im Metarouting wird von einem festen Ablaufschema (i.d.R. dem Distanzvektorprotokoll entsprechend) ausgegangen und einige, üblicherweise als fix angenommene, mathematische Operatoren des Ausgangsprotokolls wie Additionen der Linkkosten als abstrakte algebraische Operatoren aufgefasst. Sodann werden Voraussetzungen für die Operatoren abgeleitet, unter denen gewünschte Eigenschaften, z.B. Konvergenz, beweisbar eintreten. Dieser Aspekt wird in dieser Arbeit nicht weiterverfolgt, die Einführung von Operationen

(nicht Operatoren) als Beschreibung des Programmablaufs zentraler Teile des MAS dient hier ausschließlich der Unterstützung der Dekomposition des Basisprotokolls zur Erhöhung der Modularität.

Nachstehend werden nacheinander die folgenden Aspekte des MAS vorgestellt: Das Versenden der Agenten, deren Aufgabe und Ziel, das Verhalten bei Erreichen eines Knotens, sowie die Koordinierung mehrere Agenten an einem Knoten, das Weiterversenden von den erreichten Knoten (die sogenannte Propagierung) und den Umgang mit vorhandenen Tabelleneinträgen. Anschließend wird das Forwarding erläutert.

Versenden der Agenten Grundlegend für die Beschreibung des Protokolls ist folgendes Verhalten: Jeder Knoten o versendet regelmäßig Scout-Agenten an inzidente Vorgänger, im Folgenden sei dies als „Starten“ und o als (Scout-)Ursprung bezeichnet. Das einmalige Starten von Scouts von allen Knoten wird *Generation* genannt, die Zeitspanne zwischen zwei Generation *Startintervall*. In dieser Arbeit wird ein homogenes Startintervall betrachtet, d.h. alle Knoten starten Scouts in dem gleichen Abstand. Aufgrund der Asynchronität muss dies aber nicht an jedem Knoten zwangsläufig gleichzeitig geschehen. Eine Generation kann daher als eine asynchrone Variante des synchronen Rundenkonzepts betrachtet werden.

Ziel des Scoutings Aufgabe der Scouts ist es, die CCP-Fahrzeit zum Ursprungsknoten zu verbreiten. Dazu traversieren die Scouts die Links entgegen der Fahrtrichtung (Upstream), damit Fahrzeuge mit Ziel o später genau die Links in üblicher Fahrtrichtung (Downstream) nutzen können, um den Ursprungsknoten des Scouts zu erreichen. Die Scouts werden daher auch als *Upstream-Scouts* bezeichnet[1]. An jedem Knoten wird die Routingtabelle T entsprechend des zurückgelegten Pfades aktualisiert. Abbildung 6.3 verdeutlicht das Konzept. Die gestrichelten Linien von a zu b stellen gerichtete Pfade mit beliebig vielen Knoten dar. Wie in allen folgenden Darstellungen, werden der Übersicht halber nur die Scouts eines einzigen Ursprungs (fett dargestellt, hier: b) eingezeichnet. Scouts werden zum Einen mit Buchstaben, zum Anderen mit Zahlen annotiert sein. Im ersten Falle wird damit verdeutlicht über welchen Knoten ein Agent weitergeleitet wird. Im Zweiten, welche aggregierten Kosten mitgeführt werden.

Aufgrund der Upstream-Verbreitung der Scouts gilt die folgende Proposition:

Proposition 6.1. Für jeden Knoten i den ein Upstream-Scout mit Ursprung o über den Pfad $o \to v_1 \to \ldots \to v_n \to i$ erreicht, existiert der umgekehrte Downstream-Pfad $i \to v_n \to \ldots \to v_1 \to o$.

Ziel ist, dass die Routingtablle T im konvergierten Zustand, dann bezeichnet als T^*, zu jedem Scoutursprung in einer, später noch zu spezifizierenden, Umgebung, die LCP-Kosteneinträge enthält. Die Scouts verbreiten Schätzungen dieser Kosten, so dass bis zur Konvergenz ggf. *ELCP-Kosten* (Estimated LCP) in den Tabellen gespeichert sind. Die Vorgänger $s \in S_i$ von i über die ein Scout mit Ursprung o ankam, werden wegen Proposition 6.1 als *Pfadknoten* $\overrightarrow{S}_i^o \subseteq S_i$ von i zu o bezeichnet. Wenn der Kontext eindeutig ist oder im Allgemeinen von Pfadknoten gesprochen wird, wird im Folgenden auch $\overrightarrow{S}$ notiert. Die Pfadknoten, welche zum jeweiligen Zeitpunkt laut Tabelle die geringsten Kosten besitzen, werden im Folgenden als ELCP-Pfadknoten bezeichnet. Ferner wird aus sprachlichen Gründen im Kontext günstigster Pfade stets die Singularform verwendet. So wird der Einfachheit halber bspw. von nur einem LCP zwischen zwei Knoten und nur einem ELCP-Pfadknoten die Rede sein. Das spiegelt auch

[1] Dies wird später zur Unterscheidung zu Downstream-Scouts relevant.

die Praxis wieder, in komplexen Straßennetzen gibt es selten mehrere günstigste Pfade. An Stellen an denen es aus definitorischen Gründe notwendig ist, wird deutlich unterschieden.

Verhalten bei Erreichen eines Knotens Jeder Scout $\mathcal{S}$ führt eine Kostenvariable $C_{\mathcal{S}}$ mit, in welcher die Kosten des bereits zurückgelegten Pfades gespeichert werden. Beim Erreichen eines Knotens werden diese Kosten, als auch die Routingtabelle des Knotens aktualisiert und der Scout anschließend weiterpropagiert. Die Schritte im Einzelnen:

1. Erreicht $\mathcal{S}$ Knoten i über den Link (s, i), wird zunächst $C_{\mathcal{S}}$ mittels der *Link-Aggregierungsoperation*[2] $\ominus$ aktualisiert. In der Standardbelegung werden die Linkkosten ω_{is} addiert: $C_{\mathcal{S}} \leftarrow C_{\mathcal{S}} + \omega_{is}$. Dabei werden die Kosten $i \to s$ addiert (statt ω_{si}), um die Pfadkosten $i \to s \rightsquigarrow o$ zu aggregieren.

2. Gilt $C_{\mathcal{S}} < T_i^{so}$ wird anschließend wird die Routingtabelle T_i für Ziel o über Nachfolger $s \in S_i$ aktualisiert: $T_i^{so} \leftarrow C_{\mathcal{S}}$.

3. Danach wird $C_{\mathcal{S}}$ mittels der *Knoten-Aggregierungsoperation*[3] $\circledcirc$ und der Routingtabelle $C_{\mathcal{S}}$ aktualisiert. In der Standardbelegung: $C_{\mathcal{S}} \leftarrow \min_{s \in \overrightarrow{S}_i^o} T_i^{so}$. Die Operation extrahiert aus der (jüngst aktualisierten) Routingtabelle einen neuen Kostenwert, der die Kosten eines Pfades $o \rightsquigarrow i$ von repräsentiert. Dabei werden die Erfahrungen übriger Scouts, vermöge ihrer zuvor getätigten Tabellenaktualisierungen, einbezogen. Dies ist eine konstituierende Eigenschaft von SI-Verfahren. Das klassische DVP im Gegensatz dazu, aggregiert keine Kosteninformationen, die von anderen Nachrichten hinterlassen wurden.

4. Abschließend wird mittels der *Propagierungsoperation* $\bigcirc\!\!\rightarrow$ eine Menge von Knoten ermittelt, an die der Scout weitergeleitet wird. In der Standardbelegung wird $\mathcal{S}$ an alle Vorgänger $p \in P_i,\ p \neq s$, weitergesendet. Dieses Verhalten begründet sich in dem Umstand, dass zunächst mit lokalen Informationen durch den Agenten nicht entschieden werden kann, auf welchen der potentiellen, unexplorierten Pfaden sich die Fahrtkosten minimieren werden (vgl. Abschnitt 5.4). Daher werden $n = |P_i \setminus \{s\}|$ *Scoutklone* von $\mathcal{S}$ erstellt und versendet, woraus sich insgesamt ein *Flooding* ergibt (siehe Abbildung 6.4). Der Knoten s kann dabei ausgelassen werden, da durch ein Kreis keine günstigeren Fahrtkosten entstehen können (vgl. Split Horizon, Abschnitt 5.5.2). In der genannten Abbildung 6.4 sind die Agenten mit dem zuerst passierten Knoten annotiert.

Wie geschildert, wird an jedem Knoten, den ein Scout passiert, mittels der mitgeführten Kosten ggf. die Routingtabelle aktualisiert. Später wird dieses Konzept noch auf verschiedene (logische) Tabellen ausgeweitet, aber im Grunde entsteht eine Distanzvektortabelle wie in Tabelle 5.1 in Abschnitt 5.5.1 bereits dargestellt: Nachfolgeknoten sind in den Zeilen, Zielknoten in den Spalten eingetragen. Erreicht ein Scout einen Knoten, trägt er in der Spalte seines Ursprungsknoten und in der Zeile des Nachfolgers von welchem er upstream zu dem aktuellen Knoten versendet wurde, den aktuellen Wert seiner Kostenvariable ein.

Abbildung 6.5 erläutert das soweit geschilderte Disseminationprinzip der Upstream-Scouts. Der Knoten d startet einen Upstream-Scout, welcher über c zu b gelangt und anschließend zu a. Eingedenk der asynchronen Kommunikation, ist *a priori* nicht bekannt, wann genau der

[2] Das Symbol im Kreis soll einen Link darstellen.
[3] Das Symbol im Kreis soll einen Knoten darstellen.

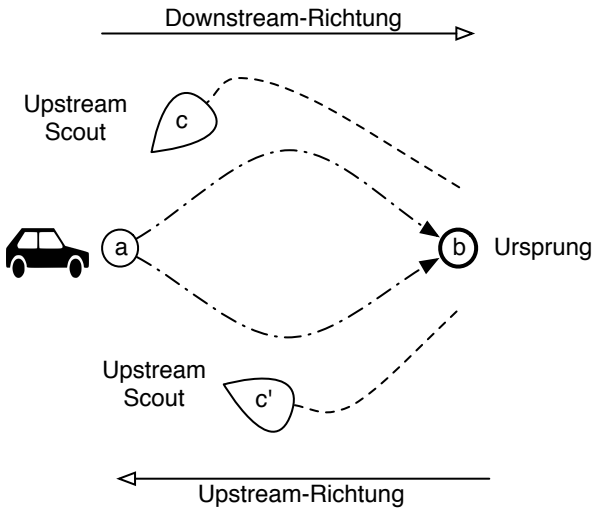


Abbildung 6.3.: Scout Terminologie

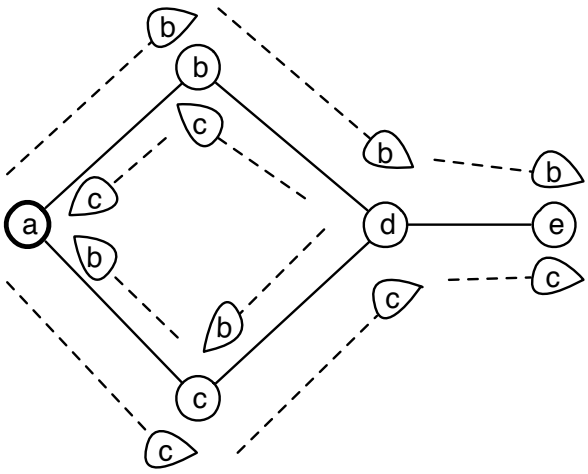


Abbildung 6.4.: Scout Flooding

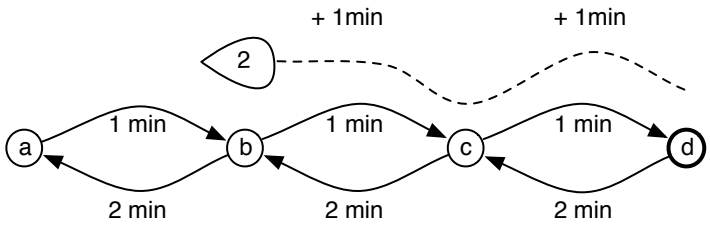


Abbildung 6.5.: Upstream-Scout

Scout die Knoten erreicht. Die einzige mögliche (und in diesem Fall triviale) Aussage, ist, dass der Scout erst c, dann b und schließlich a besucht.

Wie in der Abbildung angedeutet, traversiert der Scout die Links (c, d) und (b, c) in Upstream-Richtung und nicht etwa die Links (d, c) und (c, d) (in Downstream-Richtung). Nur so ist

gewährleistet, dass vom aktuellen Knoten des Scouts auch ein (gerichteter) Downstream-Pfad zum Scout-Ursprung besteht (vgl. Proposition 6.1). Auch muss über genau diese Upstream-Links die Fahrzeit aggregiert werden. In diesem Fall beträgt die Transitzeit von a zu d 3 Minuten; eine Aggregierung der Downstream-Link-Reisezeiten führte zu dem falschen Ergebnis von 6 Minuten.

Dieser Unterschied wird besonders dann relevant, wenn es zwischen zwei Knoten nur ein Link gibt. Am offensichtlichsten ist dies bspw. bei Einbahnstraßen. Allerdings sind in digitalisierten Karten, wie in Abschnitt 3.1 dargestellt, viele Intermediates und ob der Speicherungsart auch viele unidirektional verbundene Knoten vorhanden, obwohl sie bidirektionale Straßenabschnitte repräsentieren.

Auch die Knoten-Aggregierung sei exemplifiziert. In dem Beispiel der Abbildung 6.5, in dem nur ein Tabelleneintrag an c zu d vorhanden ist, wählt die Minimumsfunktion zur Knoten-Aggregierung den Eintrag, den der Scout selbst eingetragen hat. Wie Abbildung 6.4 verdeutlicht, können stattdessen die Einträge vorheriger Scouts vorhanden sein und eine angemessene Aggregierung kann durchgeführt werden. Die genau Vorgehensweise hängt dabei aber auch von der im Folgenden beschriebenen Scout-Koordinierung ab.

Koordinierung von Scouts an einem Knoten Geht man zunächst davon aus, dass 1) Scouts beliebig viele Links traversieren dürfen, folglich kein Hop Limit beachten müssen, 2) die Agenten sich gemäß des DVP verhielten, und 3), dass alle Knoten zum Zeitpunkt t_0 die erste Generation starten, dann kennen alle Knoten spätestens zum Zeitpunkt $t_0 + b \cdot \gamma$ einen Next Hop auf einem LCP zu jedem anderen Knoten im Netz (mit b als maximaler Transmissionzeit auf einem Kanal). Wie in Abschnitt 5.4 dargetan, würden dabei im Zweifelsfall pro Generation aber faktoriell viele Nachrichten, hier Agenten, entstehen. Ein Straßennetz einer Region umfasst aber leicht viele Tausende Knoten, so dass die Anzahl der Scouts folglich immens ansteigen könnte. Um diese Agentenflut einzudämmen, ist es sinnvoll, eine Koordinierung der Scouts untereinander anzustreben. Die zu beantwortende Frage ist, was passiert, wenn ein Scout einen Knoten erreicht, der zuvor schon von einem anderen Scout erreicht wurde. Hierbei sind mehrere Fälle zu unterscheiden:

1. Zunächst wird der Fall *unterschiedlicher Ursprungsknoten* betrachtet. In diesem Fall passiert ein Scout einen Knoten, der zuvor bereits schon von einem Scout eines anderen Ursprungsknoten besucht wurde. Da ein Scout nur Tabelleneinträge bzgl. seines eignen Ursprungsknotens vornimmt, konfligieren die beiden Agenten nicht. Anders ausgedrückt: beide müssen weitergeleitet werden, da keiner der beiden Scouts Informationen über den anderen mit sich führt und somit beide benötigt werden.

2. Im zweiten Fall stammen die Scouts vom *selben Ursprungsknoten*, aber aus *unterschiedlichen Generationen*. Um diesen Fall überhaupt diskriminieren zu können, verfügt ein Scout über eine Generationskonstante, die vom Ursprungsknoten in Abhängigkeit von der jeweiligen Generation gesetzt wird. Erreicht ein Scout einen Knoten, der bereits von einem Scout aus einer höheren Generation besucht wurde, wird der später eintreffende Scout als veraltet betrachtet und daher schlicht verworfen. Denn schließlich hat ein anderer Scout, der später von demselben Ursprungsknoten startete, bereits diesen Knoten erreicht. Der Ältere ist folglich im Kommunikationsnetz verzögert worden. Da prinzipiell nicht abschätzbar ist, wann in einem asynchronen System verzögerte Scouts eintreffen, würde ein Stop-and-Wait-Verfahren potentiell zu erheblichen Verzögerungen führen.

3. Nach Ausschluss der ersten beiden Fälle bleiben Scouts gleichen Ursprungs und gleicher Generation. Grundsätzliche Idee ist, dass *pro Generation nur ein Scout eines bestimmten Ursprungs einen Knoten verlässt.* So ist gewährleistet, dass ein Link nur von einem einzigen Scout vom selben Ursprung traversiert wird. Damit entstehen pro Ursprung für jeden Link ein Scout, damit insgesamt $|V| \cdot |L|$ Nachrichten, statt faktoriell vielen im Worst Case. In Abbildung 6.4 ist dargestellt, wie von Knoten d beide Scouts b, c zu Knoten e passieren. Das wird durch diese Einschränkung verhindert. Zu klären bleibt, welcher Scout seine Reise fortsetzen darf. Diese Frage sei nachstehend diskutiert.

Propagierungsstrategie Wenn nur eine Teilmenge der Scouts passieren darf (und hier sei nur der Spezialfall eines einzelnen Scouts diskutiert), um so die Nachrichtenkomplexität linear zu limitieren, kann nicht zwangsläufig garantiert werden, dass jedem empfangenden Prozess anschließend der LCP bekannt ist (da wie in Abschnitt 5.4 gezeigt, hierzu in einem asynchronen System ohne zusätzliche Synchronisation im Zweifelsfall faktoriell viele Nachrichten, hier Agenten, nötig wären). Die notwendige Scout-Selektionsoperation sei mit dem Symbol $\Subset$ gekennzeichnet[4]. Es seien zwei dieser *Propagierungsstrategien* diskutiert:

1. **First-Strategie:** *Weitersenden des ersten eintreffenden Scouts.* Hierbei wird der erste eintreffende Scout einer Generation weitergeleitet und der Knoten dementsprechend markiert. Später eintreffende Scouts derselben Generation dürfen zwar noch die Routingtabelle aktualisieren, werden danach aber unmittelbar verworfen, dürfen also insbesondere nicht weiterpassieren. Dieser Regelung wird bspw. bei BeeHive angewendet, obschon sie weitreichende Folgen hat. Dazu sei Abbildung 6.6 betrachtet. Problematisch ist dann nämlich der Umstand, dass die Ausbreitungsgeschwindigkeit im Kommunikationsnetz erheblichen Einfluss auf die Verbreitung des LCP hat. Angenommen, der untere Pfad über c ist bzgl. der Scout-Ausbreitungsgeschwindigkeit stets der schnellere der beiden Pfade vom Scoutursprung e zu a. Dann wird an Knoten a auch stets ein Tabelleneintrag zugunsten dieses unteren Pfades vorgenommen, mithin ist der LCP an a unbekannt. Vorteil ist, dass unmittelbar zwischen Weiterpassieren und Verwerfen entschieden werden kann.

2. **Best-Strategie:** *Weitersenden des besten Scouts der letzten Generation.* Angenommen, es gelte bei genügend geringen Startintervall folgende *Kontinuitätsannahme*: zwischen zwei Generationen ändern sich die Zustände im Straßennetz nicht signifikant. Dann ist auch die Annahme gestattet, dass die LCPs, die durch die letzte Scout-Generation verbreitet wurden, sich zu großen Teilen erhalten haben. Diese Heuristik leitet folglich einen Scout eines Ursprungs o genau dann weiter, wenn noch kein Scout von o dieser Generation weitergeleitet wurde und für ihn eine der beiden folgenden Bedingungen gilt:

   a) Der Scout traf über den besten Pfadknoten der letzten Generation ein, d.h. wenn der Scout über den Link den aktuellen Knoten erreichte, über den der Scout der letzten Generation den ELCP zum Ursprungsknoten erkundete (mit Ausnahme der ersten Generation, in der der erst ankommende Scout weitergeleitet wird).

   b) Der Scout führt geringere Kosten mit sich als der ELCP-Scout der letzten Generation.

[4] Das Symbol soll verdeutlichen, dass ein Scout gesucht wird, der „besser“ ist als die übrigen.

Jede dieser beiden Forwarding-Entscheidungsmöglichkeiten kann unmittelbar ohne Stop-and-Wait-Verfahren getroffen werden. Ein Knoten muss nur speichern, welche Kosten und welcher Nachfolger in der letzten Generation die Besten waren. Sieht man von der ersten Generation ab, ist diese Heuristik unabhängig von der Ausbreitungsgeschwindigkeit im Kommunikationsnetz, beachtet aber gleichzeitig die Auslastung im Straßennetz und weist eine lineare Nachrichtenkomplexität. Entscheidend ist natürlich die Dauer des Startintervalls. Allerdings werden in dieser Arbeit sehr kurze Zeitspannen im Sekundenbereich angestrebt, wodurch die Kontinuitätsannahme als gegeben betrachtet werden kann.

Abbildung 6.7 verdeutlicht diesen Ansatz. Die Annotation in den Scouts stellen den Wert ihrer Kostenvariable dar. Ganz links ist die Situation dargestellt, dass der linke Scout $\mathcal{L}$ mit Ursprung f und mit $C_{\mathcal{L}} = 10$ als erstes den Knoten b erreicht und weiter fliegt. Dadurch wird Vorgänger c als ELCP-Vorgänger an b zu f gespeichert. In der nächsten Generation, in der Mitte dargestellt, erreiche wieder der Scout über c als erstes den Knoten b. Obschon $C_{\mathcal{L}} = 15$, fliegt $\mathcal{L}$ weiter, und nicht der rechte, später eintreffende Scout $\mathcal{R}$ über e mit $C_{\mathcal{R}} = 5$, da c in der vorherigen Generation als ELCP-Pfadknoten markiert wurde. Wie erwähnt, liegt der Vorteil darin, dass nicht an b gewartet werden muss, bis alle Scouts angekommen sind, was sich in einem großen Netz zu Verzögerungen akkumulieren kann. Stattdessen kann unmittelbar lokal entschieden werden.

Würde an dieser Stelle die Empfangsannahme (vgl. Annahme 5.1) verletzt, könnte die Situation eintreten, dass über den markierten Vorgänger kein Scout einträfe, wodurch wiederum unter Umständen kein einziger Scout in der betreffenden Generation von Knoten b aus weiterfliegen würde. Dennoch würden die eingetroffenen Scouts einen Vorgänger markieren. In der nächsten Generation würde sodann wieder ein Scout weitergeleitet.

Die Situation ganz rechts stellt den Fall dar, dass ein Scout geringere Kosten ($C_{\mathcal{L}} = 4$) besitzt als der weitergeflogene Scout aus der vorherigen Generation ($C_{\mathcal{R}} = 5$). In dieser Ausnahme darf dieser Scout weiterfliegen (und markiert dementsprechend wieder den Vorgänger c). Angenommen, über alle Vorgänger stiegen die Kosten erheblich, dann könnte eine Situation ähnlich zur der im zweiten Teilbild entstehen: ein Scout mit nicht-minimalen Kosten könnte weiterfliegen, da er über den LCP-Vorgänger kam, obwohl seine Kosten höher sind als die anderer Scouts. *Eine Überapproximation der Kosten ist somit möglich, wird aber schnell in den folgenden Generationen korrigiert.*

Im Evaluationteil dieser Arbeit werden diese beiden Heuristiken miteinander verglichen. Es wird dabei gezeigt, dass letztere deutlich besser abschneidet. Unabhängig von der genauen Ausgestaltung, wird in beiden vorgestellten Varianten (unter der Empfangsannahme) genau ein Scout pro Generation pro Ursprung weitergesendet. Daraus folgt, dass das solch ein Flooding-Ansatz terminiert und zwar genau dann, wenn über jeden Link (des Straßennetzes) genau ein Scout von jedem Knoten verbreitet wurde, wodurch $|V| \cdot |L|$ Nachrichten insgesamt entstehen. Straßennetze bestehen jedoch leicht aus tausenden von Knoten und Links, die Anzahl der Scouts wäre dennoch erheblich. Im folgenden Abschnitt 6.5 wird daher ein Hierarchie-Konzept zur Reduzierung der (absoluten) Nachrichtenanzahl eingeführt.

Im Vergleich zu den genannten Strategien verwendet das klassische DVP eine Art „Better-Strategie“, bei der Nachrichten weitergereicht werden, sobald die Tabelle geändert werden durfte.

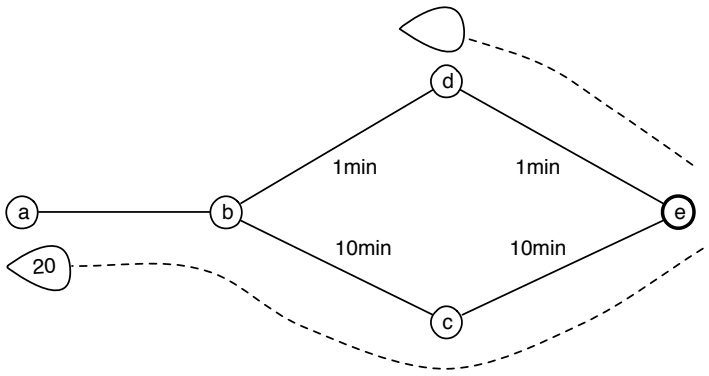


Abbildung 6.6.: Problem bei Weiterleitung des ersten eintreffenden Scouts

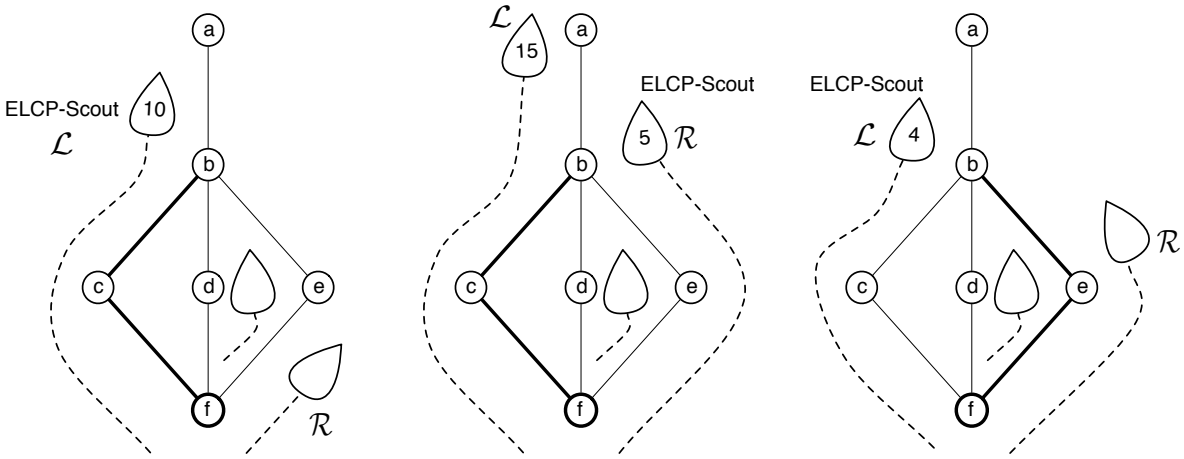


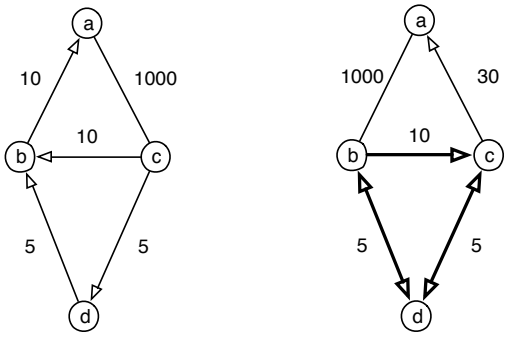
Abbildung 6.7.: Scout-Selektion

Es kommt ferner die Frage auf, wie mit alten Tabelleneinträgen umzugehen ist, die von vorherigen Generationen erstellt wurden.

Umgang mit alten Tabelleneinträgen Es seien wieder zwei Varianten betrachtet:

1. *Beibehalten alter Einträge.* Eintreffende Scouts, die die Tabellen ändern, überschreiben alte Einträge, sonst werden keine Modifikationen an den Tabellen vorgenommen. Dieses Verhalten wird bei BeeHive umgesetzt, obschon dies zu Tabellenschleifen führen kann, wie Abbildung 6.8[5] zeigt. Im linken Teilbild geben die gerichteten Links die LCP-Richtungen zu a an, die sich rein aus den Linkkosten ergeben. Das rechte Teilbild zeigt die Situation, nachdem sich die Linkkosten, wie angegeben, verändert haben. Nimmt man an, dass der Scout über $a \to c$ stets schneller über den unteren Link zu b gelangt und ferner dies passiert bevor der Scout von a über den direkten Link zu b gelangt. Dann werden die ELCP-Kosten von 20 von c über b zu a niemals geändert. Sie bleiben damit bestehen, da Knoten b immer den Scout, der über den unteren Link kam, weiterleiten wird – und keinen anderen Scout. Dadurch schließt sich ein Kreis: Fahrzeuge an b werden zu c

[5]Die leeren Pfeilspitzen (im Gegensatz zu den sonst ausgefüllten) bedeuten, dass Tabelleneinträge statt Linkrichtungen visualisiert sind.



(a) FFT-Tabelleneinträge

(b) Alte Kosten $\delta(c \rightsquigarrow a) = 20$

Abbildung 6.8.: Tabellenschleifen durch Erhalten von Tabelleneinträgen älterer Generationen.

gesendet und dann aufgrund von „veralteten“ Informationen zurück zu b, wie das rechte Teilbild verdeutlicht. Dieses Verfahren ist somit ohne weiteres nicht verwendbar.

Vorteil dieses Verhaltens ist allerdings, dass unter Umständen eine größere Anzahl von Pfaden zum Ziel an einem Knoten „bekannt“ ist, wenngleich die Tabellenreferenzen nicht kreisfrei sind. Statt einem In Tree zum Scoutursprung (vgl. Proposition 5.11), entsteht folglich ein gerichteter, nicht zwingend kreisfreier, Graph. Dieses Konzept wird später noch einmal aufgegriffen.

2. *Löschen der alten Einträge.* Sobald der erste Scout einer Generation an einem Knoten eintrifft, werden zunächst die Tabelleneinträge mit Ziel des Scoutursprung gelöscht, denn sie stammen aus vorherigen, veralteten Generationen. Tabellenschleifen wie zuvor werden dadurch ausgeschlossen. Im konkreten Beispiel würde der ELCP-Pfadknoten an Knoten c (über b zu a) aus der Tabelle gelöscht, sobald der Scout der neuen Generation auf dem Link $a \rightarrow c$ eintrifft.

Damit ist das grundsätzliche Disseminationsverhalten beschrieben, es folgt eine Darstellung potentieller Forwarding-Prinzipien, also das Weiterleiten der Fahrzeuge.

Fahrzeug-Forwarding Für das Forwarding wird die $\oplus$-Operation[6] eingeführt, die aus einer Menge von 2-Tupeln, bestehend aus einem Pfadknoten und zugehörigen Kosten einen Next Hop selektiert. Es bieten sich zwei Möglichkeiten an:

1. *Wahl des Knotens mit den geringsten Kosten als Next Hop.* Aus der Routingtabelle wird der Nachfolgeknoten gewählt, der auf einem bereits explorierten Pfad zum Ziel liegt und die geringsten Kosten aufweist. Selbst wenn die Next Hop-Richtungsangaben aus den Tabelleneinträgen einen Graph bilden, wird dadurch ein In Tree erzwungen. Dieses Konzept wird im Folgenden auch als min-Forwarding bezeichnet.

2. *Wahl eines Knotens proportional zu den assoziierten Kosten.* Hierbei kommt jeder Nachfolgeknoten (auf einem Pfad zum Ziel) mit einem Tabelleneintrag in Frage. In Abhängigkeit seiner Kosten wird probabilistisch selektiert: je höher die Kosten, desto

[6] Das Symbol im Kreis soll eine Kreuzung darstellen.

geringer die Auswahlwahrscheinlichkeit ϕ des Knotens als Next Hop. Im Folgenden wird dies als ϕ-Forwarding bezeichnet.

Möglichkeit 1 führt zu einem verteilten Ex-Ante-ELCP-Routing mit CCP-Kosten, bei dem an jedem Knoten für ein Fahrzeug neu entschieden wird, welcher Nachfolger auf dem gegenwärtigen ELCP liegt. Ist das Startintervall zwischen den Scoutgenerationen klein genug, dass zwischen zwei aufeinander folgenden Forwarding-Entscheidungen eines Fahrzeugs (an zwei aufeinander folgenden Kreuzungen), die beteiligten Tabellen aktualisiert werden, basieren die Next Hop-Entscheidungen auf aktuellen Verkehrsinformationen. Das BeeJamA-Basisprotokoll wählt genau diese Möglichkeit.

Stattdessen könnten auch suboptimale Knoten (antonym zu den optimalen Ex Ante-Kosten des Ex Ante-ELCPs) gemäß Möglichkeit 2 gewählt werden. Jedoch ist das wiederum mit zwei elementaren Problemen verbunden.

1. Erstens würden Fahrzeuge nicht über den für sie aktuell günstigsten Pfad geleitet. Würde man dies zumindest an die Auslastung knüpfen, könnte dadurch ggf. ein globaler Vorteil erhofft werden, d.h. im Sinne des System Optimums sinken die durchschnittlichen Fahrzeiten (wohingegen die individuellen Fahrzeiten partiell ansteigen könnten). Jedoch ist damit auch eine psychologische Fragestellung verknüpft: Möchte ein Fahrer, bspw. ein Pendler, morgens auf dem Weg zur Arbeit, wirklich statt auf dem schnellsten Wege lieber einen langsameren Weg wählen, dafür aber der Allgemeinheit helfen? Diesen Fragen seien weiter unten noch ausführlicher diskutiert.

   BeeHive verwendet solch eine stochastische Auswahl, verknüpft dies jedoch nicht mit einem Auslastungsgrad, wodurch der potentielle Vorteil nicht aufgegriffen wird. Anders ausgedrückt: Datenpakete werden über suboptimale Wege geleitet, obwohl überhaupt keine Notwendigkeit dafür bestünde, da die „optimalen“ Wege noch gar nicht ausgelastet sind. Auch dieser Aspekt wird noch deutlicher diskutiert.

2. Das zweite Problem hat ebenfalls erhebliche Auswirkungen. Probabilistisches Routing führt – sofern keine zusätzlichen Maßnahmen ergriffen werden – zu Kreisen. Man stelle sich zur Verdeutlichung einen Random Walk auf einem Graphen vor. Offensichtlich können dabei Knoten mehrfach besucht werden. Auch wenn die Auswahlwahrscheinlichkeit proportional zu den Kosten gewählt wird, können ebenfalls leicht Kreise entstehen. Maßnahmen, um in einem verteilten System solche Kreise a priori zu identifizieren oder gar zu eliminieren, erzeugen zusätzlichen Overhead.

Allein die Kombination dieser beiden Probleme lässt deutlich werden, dass ϕ-Forwarding nicht unbedingt geeignet für ein VRGS erscheint.

In diesem Abschnitt wurde einführend das Grundkonzept des Multi-Agenten System beschrieben. Jeder Knoten versendet kontinuierlich eine Generation von Upstream-Scouts, welche mittels Flooding Pfade explorieren und entsprechend die Tabellen aktualisieren. Fahrzeuge werden gemäß dieser Einträge downstream zum Ziel geleitet, wobei an jeder Kreuzung neu entschieden wird, im besten Falle basierend auf neusten Verkehrsinformationen. Problematisch ist die hohe, wenngleich linear limitierte, Nachrichtenkomplexität, da ohne Reichweitenlimitierung das Flooding fortgesetzt wird, bis alle Scouts nicht mehr weitergeleitet können. Der nächste Abschnitt geht darauf ausführlich ein.

6.5. Hierarchie

Im vorherigen Abschnitt wurden die grundlegenden Konzepte des BeeJamA-Basisprotokolls eingeführt, dabei jedoch der Spezialfall erläutert, in dem Scouts eine unendliche Reichweite erlaubt ist. Da dies für Netze realistischer Größen eine enorme Agentenflut erzeugt, ist ein wesentlicher Bestandteil des Verfahrens ein Hierarchiekonzept. Zwar lässt sich dadurch die Anzahl der Scouts reduzieren, jedoch geht damit unausweichlich eine Vergröberung der explorierten Pfade einher. Anders ausgedrückt: nicht an allen Knoten werden Pfade zu allen übrigen (Ziel-)Knoten bekannt seien. Hier seien zunächst die Konzepte dargelegt, Diskussion der einhergehenden Abwägungen finden später statt (und werden anschließend simulativ ausgewertet).

Inspiriert ist das umgesetzte Hierarchiekonzept durch das Verhalten des Bienenvolkes bei der Futtersuche, das zwischen einem Nah- und Fernbereich um den Stock unterscheidet. Im Nahbereich um den Bienenstock ist eine hohe Explorationsabdeckung zu beobachten, im Fernbereich eine deutlich geringere (vgl. Abschnitt 6.2). Durch die gewählte, geographisch-orientierte V2I-Architektur entsteht ein begrenzter Nahbereich quasi von alleine: die Areas. Areas sind zusammenhängende, disjunkte geographische Gebiete, zu denen jeweils eine Teilmenge aller Knoten des Straßennetzes gehört.

Die folgenden Betrachtungen (als auch die Implementierung der Protokolle) vereinfachen sich deutlich, wenn eine Area zusätzlich die Bedingung des starken Zusammenhangs erfüllt. Sollten die durch die Grid-Dissektion entstandenen Areas diese Bedingung nicht erfüllen, wurden sie für die später erläuterten Simulationen in ihre (starken) Zusammenhangskomponenten zerlegt, so dass im Folgenden diese Eigenschaft stets als gegeben betrachtet werden kann.

Areas bilden die unterste Hierarchiebene, den sogenannten *Area Layer*. Scouts des Area Layers, genannt *Area Scouts*, verlassen ihren *Ursprungsbereich*, d.h. die Area ihres Ursprungsknotens, nicht. Im Gegensatz zu den folgenden Hierarchiebenen, startet auf dem Area Layer jeder Knoten Scouts.

Startet demnach ein Area Scout an einem Knoten des Area Layers, so wird dieser gemäß Disseminationvorschriften des vorheringen Abschnitts das Netz der Area traversieren. Ein Scout wird zusätzlich genau dann nicht über einen Link (i, j) weitergeleitet, wenn j außerhalb des Ursprungsbereichs liegt. Abbildung 6.9 illustriert dieses Konzept: gestrichelte Links werden von Scouts nicht verwendet. Die Idee hinter diesem Vorgehen ist, zumindest in dem Nahbereich eines Knotens eine hohe Präzision bzgl. der Pfadkosten zu erzielen. Ein Hop Limit existiert auf dem Area Layer nicht, die Limitierung entsteht allein aufgrund der geographischen Beschränkung einer Area.

Die zweite Hierarchieebene ist der sogenannte *Net Layer*. Im Gegensatz zum Area Layer besteht der Graph des Net Layers nur aus einer Teilmenge von Knoten und Links. Genau die Knoten, die auf dem Area Layer Nachbarn in umliegenden Areas haben, sogenannte *Border Nodes* (oder Randknoten), bleiben auch auf dem Net Layer vorhanden. Analog dazu die Links, von denen genau die erhalten bleiben, die Randknoten in unterschiedlichen Areas miteinander verbinden. Abbildung 6.10 enthält den Net Layer entsprechend des Area Layer aus Abbildung 6.9.

Die Menge der Border Nodes B_A der Area A wird weiter unterteilt. Zum Einen in *Entry Nodes* über welche die Area erreicht werden kann, $E_A = \{v_1 | (v_0, v_1) \in L \wedge v_0 \notin A \wedge v_1 \in A\}$. Zum Anderen in *Forwarding Nodes* über die Fahrzeuge die Area in Richtung anderer Areas verlassen werden können, $F_A = \{v_0 | (v_0, v_1) \in L \wedge v_0 \in A \wedge v_1 \notin A\}$.

Diese Unterteilung macht eine Aufteilung der Tabellen notwendig: Die Scouts des Area

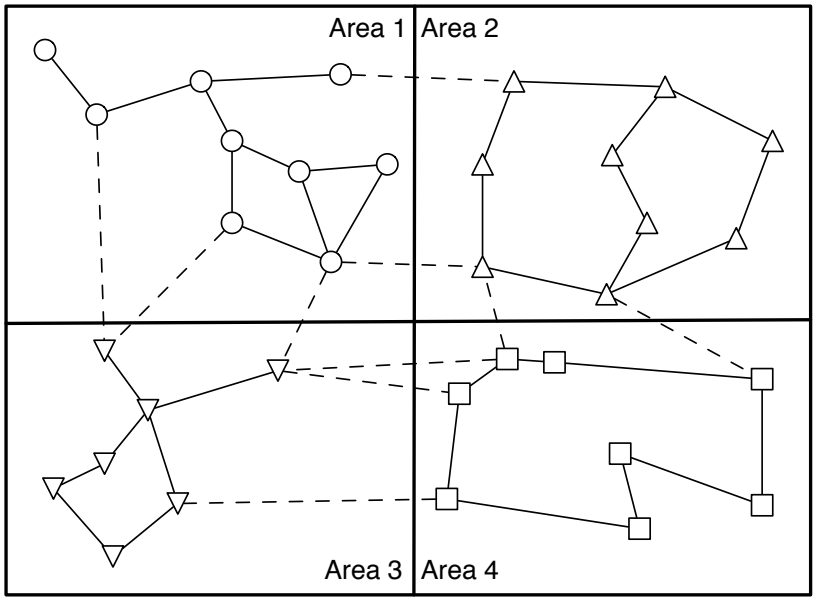


Abbildung 6.9.: Graph des Area Layers

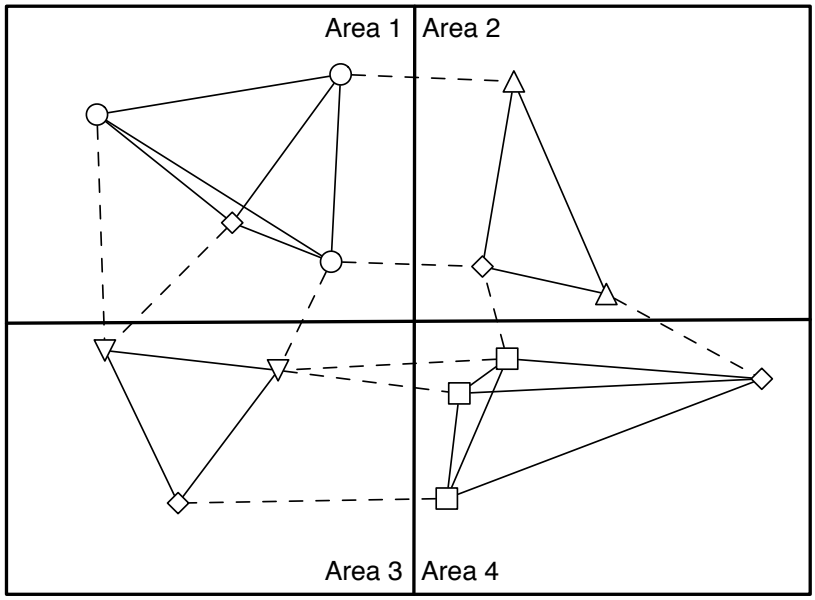


Abbildung 6.10.: Graph des Net Layers

Layers lesen und schreiben die *Area*-Tabelle $\mathfrak{A}_i$ des aktuellen Knoten i, die des Net Layers die *Net*-Tabelle $\mathfrak{N}_i$. Somit verwaltet der Navigator für jeden Knoten eine Area-Tabelle und falls der Knoten auch auf dem Net Layer vorkommt, ebenso eine Net-Tabelle.

Der Net Layer kann weiter in mehrere Hierarchieebenen unterteilt sein. Hier wird zunächst eine Grundunterteilung in zwei Ebenen eingeführt, die später noch verallgemeinert wird.

Unterschieden wird dabei zwischen den bereits erwähnten Short Distance und Long Distance

Scouts. Beide Scouttypen werden nach den selben Disseminationsregeln weitergeleitet, weisen jedoch unterschiedliche Hop Limits auf.

Aufgabe der *Short Distance Scouts* ist, den Nahbereich der Scout-Ursprungsarea abzudecken. So können sich diesem Bereich nähernde Fahrzeuge mittels möglichst präziser Informationen weitergeleitet werden. Dabei sollen die Fahrzeuge über einen möglichst günstigen Randknoten in den Zielbereich geführt werden. Dazu starten alle Randknoten des Net Layers Short Distance Scouts über die Upstream-Links, deren Startknoten in anderen Areas liegen. Der empfangende Navigator nimmt die Scouts entgegen, dekrementiert das restliche Hop Limit des Scouts und floodet den Scout seinerseits über alle Net Layer Upstream-Links, mit Ausnahme der Links mit Area-Ursprung gleich dem Scoutursprung. Innerhalb der empfangenden Area muss auf dem Net Layer kein Flooding durchgeführt werden, da die Kosten für innerbereichliche Pfade von dem zugrundeliegenden Layer, dem Area Layer, übernommen werden kann. Dadurch kann das Flooding von den Randknoten aus direkt in Richtung der weiteren, umgebenden Areas fortgesetzt werden.

Der Navigator geht dabei wie folgt vor. Für jeden Forwarding-Knoten f werden aus der Area-Tabelle die günstigsten ELCP-Kosten (welche im besten Falle denen des LCPs entsprechen) von f zum aktuellen Entry-Knoten r des Scouts entnommen, über den der Scout die aktuelle Area erreicht hat. Dann wird die Kostenvariable des Scouts um die Kosten dieses Eintrags erhöht und über den Forwarding Knoten an benachbarte Areas weitergesendet – sofern das verbleibende Hop Limit größer Null ist und es nicht zurück zur Area des Scoutursrpung geht. Ist kein passender Tabelleneintrag vorhanden, wird dieser nicht weiter behandelt, weil demnach kein explorierter Downstream-Pfad vom aktuellen Knoten zu dem Forwarding Knoten existiert.

Das von den Short Distance Scouts abgedeckte Gebiet wird als Foraging Zone bezeichnet.

Definition 6.2 (Foraging Zone einer Area)**.** Die Foraging Zone (FZ) einer Area sind genau die Areas, die von Short Distance Scouts der Area erreicht werden.

Short Distance Scouts werden im Folgenden auch entsprechend als FZ-Scouts bezeichnet. In Abbildung 6.11 stellt der blaue Bereich die FZ der Area A dar.

Long Distance Scouts dienen dazu, das „restliche“ Netz zu informieren. Die folgende Annahme drückt die umgesetzte Idee aus.

Annahme 6.3. Da die empfangenden Knoten der Long Distance Scouts weiter vom Scoutursprung entfernt sind, reichen weniger präzisere Informationen. Denn bis ein Fahrzeug den Scoutursprung erreicht, vergeht eine gewisse Zeitspanne, in der sich die Verkehrssituation noch mehrfach ändern kann, weswegen präzise Informationen nicht zwangsläufig notwendig sind.

Eine Präzisionsreduzierung kann bspw. über ein höheres Startintervall oder, wie hier umgesetzt, durch eine geringere Anzahl von Knoten, die Scouts starten, geschehen. Dazu werden *repräsentative Knoten* ausgewählt: im einfachsten Falle ein zufälliger Knoten pro Area. Nur repräsentative Knoten dürfen Long Distance Scouts starten, die prinzipiell, abgesehen von dem größeren initialen Hop Limit, genau wie Short Distance Scouts behandelt werden. Allerdings mit einem Unterschied beim Verhalten in der Ursprungs-Area. Während Short Distance Scouts nur über die Net Layer Upstream-Links des Ursprungsknoten in angrenzende Areas versendet werden, werden Long Distance Scouts auch an die übrigen Randknoten des Ursprungsbereich weitergereicht. Diese senden die Scouts dann ihrerseits über ihre Net Layer Upstream-Links. Dabei werden wieder die Kosten gemäß Area Tabelle für den günstigsten,

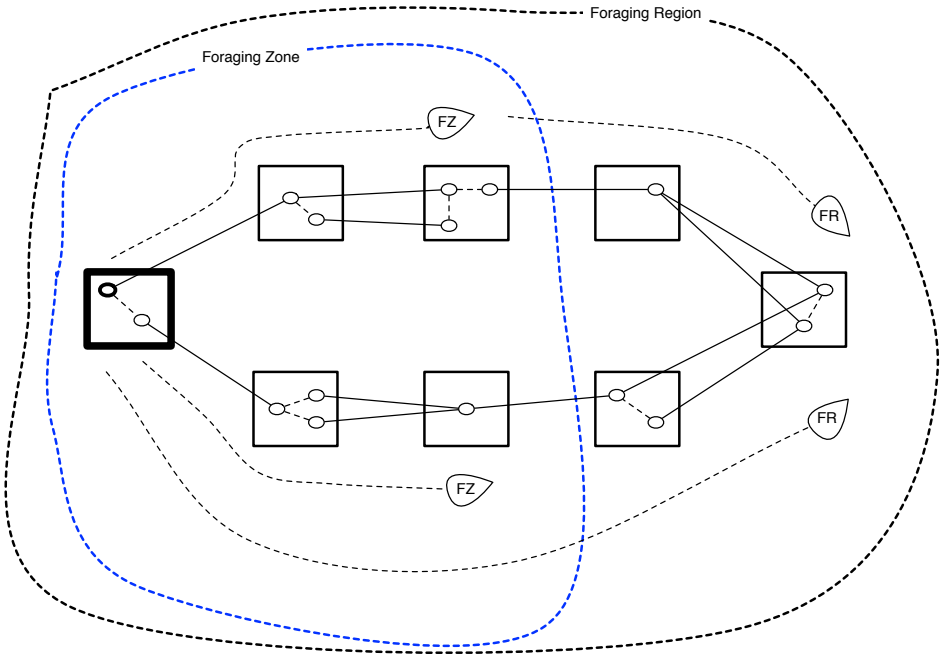


Abbildung 6.11.: Foraging Zone und Foraging Region

bekannte Downstream-Pfad auf die Kostenvariable aufgeschlagen. Dieses unterschiedliche Verhalten im Ursprungsbereich ist der einzige funktionale Unterschied zu den Short Distance Scouts. Der abgedeckte Bereich wird als *Foraging Region* bezeichnet.

Definition 6.4 (Foraging Zone einer Area)**.** Die Foraging Region (FR) eines repräsentativen Knotens sind genau die Areas, die von Long Distance Scouts des Knotens erreicht werden.

In dieser dichotomen Unterteilung müssen die im Folgenden auch als FR-Scouts bezeichneten Long Distance Scouts das gesamte Netz abdecken. Nur so ist es allen Areas im Netz möglich, Fahrzeuge in die Richtung des repräsentativen Knoten des Ziel weiterleiten zu können. Daher muss das Hop Limit der Long Distance Scouts zunächst unbegrenzt sein. In Abschnitt 6.8 wird dieses zweiteilige Konzept verallgemeinert.

Das notwendige Kommunikationsnetz zwischen den Navigatoren ist ungleich des Straßennetzes, da Nachrichten, d.h. Agenten, nur zwischen den Navigatoren ausgetauscht werden müssen. Zwar werden Agenten logisch über Links gesendet, physisch muss die Nachricht jedoch nur zu dem jeweiligen Navigator des Zielbereichs weitergereicht werden. Ein Navigator muss potentiell dann Nachrichten an einen anderen Navigator senden, wenn ein Link zwischen den assoziierten Areas existiert.

Das Konzept dieser Kommunikationsrelation lässt sich als ein Graph $G_C = (V_C, E_C)$ modellieren, der im Folgenden als Communication Graph bzw. *Communication Layer* bezeichnet wird. Pro Area A des Area Layers, existiert ein Knoten $a \in V_C$. Falls zwischen zwei Areas A und B mindestens ein Link im Routinggraph verläuft, so wird eine Kante zwischen den zugehörigen Knoten im Communication Graph hinzugefügt. Für den Net Layer Graph in Abbildung 6.10 enthielte dieser Graph die Knoten 1-4, jeweils einen für die gleichnamige Area, und die (bidirektionalen) Kanten $(1,2), (2,3), (4,3)$ und $(3,1)$.

Die vorstehenden Konzepte dieses Abschnitts bilden das Grundgerüst des BeeJamA MAS

und sind in dieser Form bereits geeignet Routinginformationen in Netzen verteilt zu verbreiten. Der nächste Abschnitt erläutert, wie anhand dieser das Forwarding umgesetzt wird.

6.6. Hierarchisches Routing

Prinzipiell lassen sich drei Hierarchie-Strategien unterscheiden:

Flach: Es wird stets in Richtung des Ziels geroutet. Dazu müssen im gesamten Netz Informationen verfügbar sein, wie das Ziel zu erreichen ist. Diese Strategie benötigt die größte Anzahl an Nachrichten.

Hierarchisch zu Zwischenzielen: Es wird nur mittelbar zum Ziel geroutet, auf dem Weg dorthin müssen (ggf. mehrere) Zwischenziele besucht werden. Es muss hierbei am aktuellen Knoten des Fahrzeugs nur bekannt sein, welches Zwischenziel als nächstes zu wählen ist und wie dieses zu erreichen ist. Dazu ist es ausreichend, wenn die Erreichbarkeitsinformation zu Zwischenzielen nur gerade so weit verbreitet werden, dass eingedenk der übrigen Zwischenziele, eine geschlossene Kette zu dem eigentlichen Ziel entsteht. Jedes Zwischenziel hat so einen Einflussbereich, in dem sich kein weiteres Zwischenziel befindet. Die Einflussbereiche sind somit disjunkt, eine Überlappung existiert nicht. Korrekt umgesetzt, benötigt solch eine Strategie am wenigsten Nachrichten. Nachteilig ist jedoch, dass die Zwischenziele von, im Zweifelsfall sehr vielen, Fahrzeugen besucht werden müssen, weswegen dort erhöhte Staugefahr dräut.

Hierarchisch in Richtung von Zwischenzielen: Auch hierbei sind Zwischenziele vorhanden. Diese müssen aber nicht zwingend besucht werden, da die Einflussbereiche der Zwischenziele sich überlappen. Sobald ein Einflussbreich des nächstes Zwischenziels erreichbar ist, wird das Fahrzeug in diese Richtung weitergeleitet, statt zum bisherig aktuellen Zwischenziel. So müssen nicht alle Fahrzeuge in der Umgebung zwingend ein und dasselbe Zwischenziel besuchen, jedoch wird eine größere Anzahl an Nachrichten benötigt.

Das Routing in BeeJamA gehört zu der letztgenannten Kategorie. Fahrzeuge werden immer soweit in Richtung eines repräsentativen Knotens geleitet, bis präzisere Informationen vorliegen. Damit bekannt ist, welcher repräsentative Knoten anzusteuern ist, wird eine *Membership*-Tabelle $\mathfrak{M}$ benötigt, in der eine Zuordnung von allen (Ziel-)Knoten zu *ihrem* repräsentativen Knoten aufgeführt ist. Da diese Zuordnung statisch ist, kann solch eine Tabelle initial einmalig erstellt und an alle Fahrzeuge verteilt werden. Diese Tabelle wird demnach im Folgenden als gegeben betrachtet.

Für die nachstehenden Betrachtungen sei angenommen, dass das Fahrzeug sich gegenwärtig an Knoten i in Area I befindet mit Ziel d in Area D. Insgesamt entstehen drei *Routingfälle*, die bei einer Forwarding-Entscheidung auftreten können:

1. Der Zielknoten ist in der aktuellen Area, d.h. Zielknoten ist in der Area-Tabelle des aktuellen Knoten vorhanden, $d \in \mathfrak{A}_i$.

2. Die aktuelle Area liegt in der Foraging Zone des Zielbereichs, d.h. mindestens ein Forwarding-Knoten der aktuellen Area hat ein Net-Tabelleneintrag zu mindestens einem Entry-Knoten des Zielbereichs, $\exists f \in F_I, e \in E_D\colon e \in \mathfrak{N}_f$.

3. Sonst.

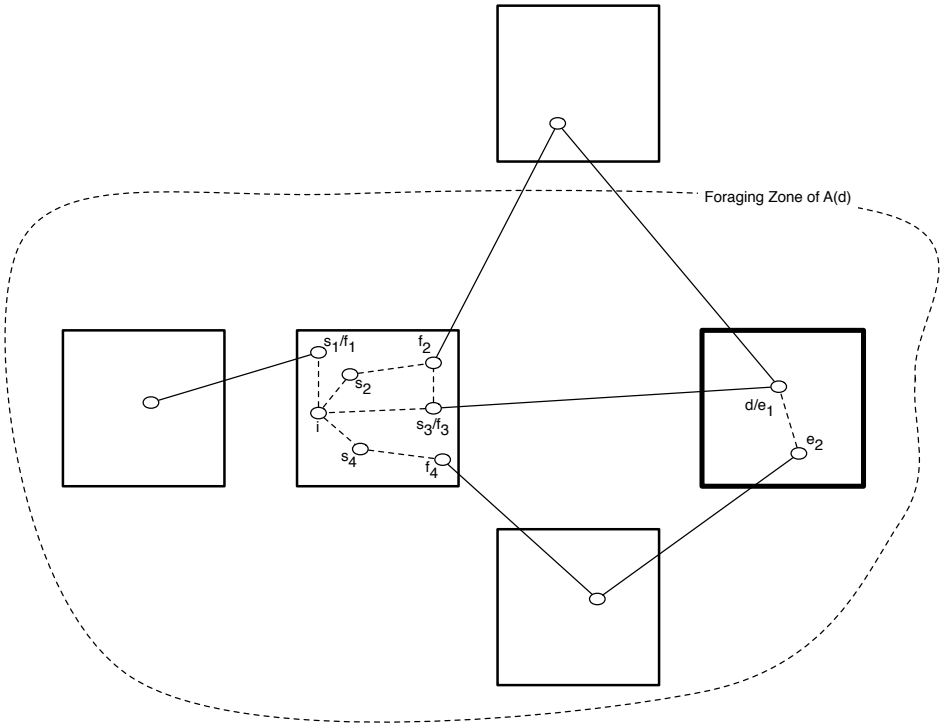


Abbildung 6.12.: Routingfall 2: d kann nicht über alle Pfade erreicht werden

Diese drei Fälle spiegeln Situationen wieder, in welcher die zur Verfügung stehenden Informationen weniger präzise werden. In der Regel, wird die Entfernung zwischen aktuellem und Zielknoten dabei immer größer.

Im ersten Fall, $I = D$, das Fahrzeug befindet sich schon in der Zielarea, wird aus der Area-Tabelle der Nachfolger ausgewählt, der für das Ziel den geringsten Kosteneintrag enthält. In diesem Fall liegen die präzisesten Informationen vor, da keine Zwischenziele angesteuert werden. Hingegen verbreiten die Scouts die (Ex-Ante-)ELCPs direkt zum Ziel.

Der zweite Fall tritt auf, wenn sich das Fahrzeug außerhalb der Zielarea, aber noch in der Foraging Zone des Ziels befindet. In dieser Situation existiert kein Area-Tabelleneintrag des Ziels am aktuellen Knoten, aber immerhin ist mindestens eine Teilmenge von Entry-Knoten des Zielbereichs in den Net-Tabellen der Forwarding-Knoten des aktuellen Bereichs eingetragen. Ein typische Situation hierzu ist in Abbildung 6.12 dargestellt.

Die möglichen Pfade vom aktuellen Knoten zum Zielknoten setzen sich im zweiten Fall aus drei Teilpfaden zusammen:

1. Einem Teilpfad vom aktuellen Knoten i über ein $s \in \overrightarrow{S}_i^f$ zu einem Forwarding-Knoten $f \in I$
2. Einem Teilpfad von dem Forwarding-Knoten $f \in I$ zu einem Entry-Knoten $e \in E_D \subseteq D$
3. Einem Teilpfad von dem Entry-Knoten $e \in E_D$ zum Zielknoten $d \in D$

Diese drei Teilpfade werden demnach durch drei „Stützknoten“ (s, f, e) determiniert. Aus allen möglichen Kombinationen $\overrightarrow{S}_i^f \times F_I \times E_D$ dieser Stützknoten, wird die günstigste gewählt. Dazu werden zunächst alle Forwarding-Knoten f des aktuellen Bereichs betrachtet und genau

die aussortiert, welche nicht vom aktuellen Knoten innerhalb des aktuellen Bereichs über Pfadknoten erreichbar sind (es existiert kein Area-Tabelleneintrag am aktuellen Knoten). Ähnlich wird mit den Entry-Knoten verfahren, denn ist kein Pfad von diesem zum Zielknoten bekannt (kein Area-Eintrag beim Entry-Knoten zum Ziel), so werden auch diese aussortiert. Die erste Überprüfung erfolgt rein lokal (einfacher lokaler Tabellen-Lookup), die zweite Überprüfung muss mit Hilfe des entfernten Zielnavigators erfolgen, da prinzipiell nur dort bekannt ist, ob ein Pfad vom Entry- zum Zielknoten existiert. Dazu wäre eine Anfrage an den Zielnavigator notwendig, jedoch wird in dieser Arbeit von einem statischen Netz ausgegangen, so dass entweder die erste Anfrage für spätere Zwecke zwischengespeichert werden kann oder schlicht das a priori dem Navigator bekannte globale Netz geprüft werden kann.

Für jedes verbleibende Forwarding-/Entry-Knotenpaar wird dann der günstigste Eintrag aus den Net-Tabellen der Forwarding-Knoten betrachtet und zu diesen Knoten der günstigste Kosteneintrag aus der Area-Tabelle des Entry-Knoten (zum Ziel) addiert. Die dazu notwendige Anfrage bei dem Zielnavigator hingegen kann nicht durch globales Wissen (wie bei der vorherigen Abfrage bzgl. der Existenz eines Pfades zum Ziel) beantwortet werden. Deutlicher ausgedrückt: Jedes Forwarding im Fall 2 erfordert eine Anfrage beim Zielnavigator, sowie dessen Rückantwort.

Der dritte Fall ist dann gegeben, wenn das Fahrzeug sich außerhalb der Foraging Zone befindet. Das Forwarding findet daher in Richtung des repräsentativen Knotens r_d des Ziels d statt. Der Pfad setzt sich aus den beiden folgenden Teilpfaden zusammen:

1. Einem Teilpfad von i zu einem Forwarding-Knoten $f \in F_I \subseteq I$
2. Einem Teilpfad von $f \in F_I$ zu dem repräsentativen Knoten r_d des Ziels d, welcher der Membership-Tabelle $\mathfrak{M}$ entnommen werden kann

Das Vorgehen ist analog zu dem des zweiten Falls. Es werden für jeden erreichbaren Forwarding-Knoten des aktuellen Bereichs die Kosten zum repräsentativen Knoten r_d aus der Net-Tabelle entnommen und der Pfadknoten s in Richtung des Forwarding-Knotens f mit den geringsten Kosten aller Möglichkeiten $\overrightarrow{S}_i^f \times F_I$ gewählt. Eine Anfrage bei einem entfernten Navigator ist daher in diesem Fall nicht notwendig.

Wichtig beim einem Next Hop-Protokoll ist, dass stets ein Next Hop aus den lokalen Tabelleneinträgen abgeleitet werden kann. Diese Eigenschaft sei im Folgenden als *Forwarding-Kette* bezeichnet. Für das oben geschilderte Konzept ist die Eigenschaft gegeben: Angenommen, ein Fahrzeug startet in Fall 3 und wird dann solange weitergeleitet, bis Fall 2 eintritt. Fall 2 muss schließlich eintreten, denn bevor der repräsentative Knoten erreicht wird, muss eine Area passiert werden, in der von mindestens einem Entry-Knoten des Zielbereichs ein FZ-Scout eintraf – und sei es von dem repräsentativen Knoten selbst, da dieser ebenfalls einen Entry-Knoten darstellt. Tritt Fall 2 ein, kann das Fahrzeug zu einem Entry-Knoten des Zielbereichs geleitet werden, von wo ab Fall 1 gilt und das Fahrzeug somit schließlich zum Ziel gelangt. Ist das Ziel der Entry-Knoten über den der Zielbereich betreten wird selbst, so entfällt Fall 1.

Das geschilderte Forwarding-Prinzip ist in dieser Form funktionstüchtig, jedoch kann eine Situation wie in Abbildung 6.13 eintreten. Das Fahrzeug befindet sich im Zielbereich (Fall 1), der LCP führt allerdings aus dem Bereich hinaus. Würden die drei Fälle bei Fall 1 beginnend evaluiert und abgebrochen sobald ein Fall zutrifft, könnte der Zielbereich nicht mehr verlassen werden. Daher wird beim Zutreffen von Fall 1, ebenfalls noch Fall 2 geprüft. Entstünden durch das Verlassen des aktuellen Bereichs über einen Forwarding-Knoten und den Wiedereintritt über einen Entry-Knoten an anderer Stelle geringere Kosten, würde diese Alternative gewählt.

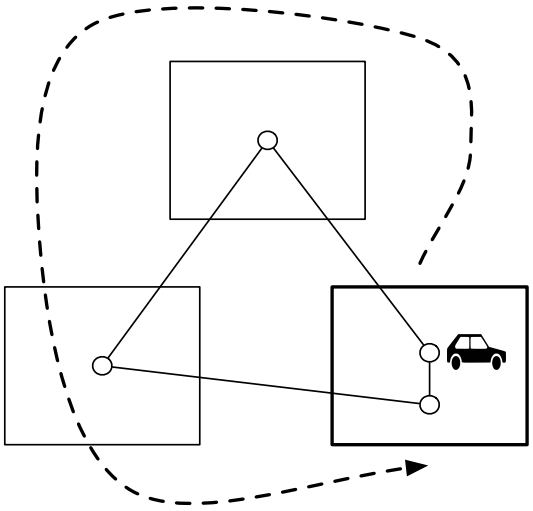


Abbildung 6.13.: LCP führt aus der Zielarea heraus

6.7. Das Basisprotokoll

Aus den Schilderungen der drei vorherigen Abschnitte wird nun ein funktionstüchtiges Basisprotokoll hergeleitet, dass in den nachfolgenden Abschnitten sukzessive erweitert wird.

Initial wird eine Offline-Aufteilung des Straßennetzes in geographisch zusammenhängende Gebiete, wie in Abschnitt 6.1 dargestellt, durchgeführt. Jedes Gebiet wird weiter in seine (starken) Zusammenhangskomponenten zerlegt, wobei jede Komponente einer Area entspricht. Jedem Bereich wird genau ein Navigator zugeordnet. Durch Überprüfen der Nachbarn aller Knoten, bestimmt ein Navigator die Menge der Randknoten, weiter unterteilt in Entry- und Forwarding-Knoten.

Die Scout-Dissemination geschieht dann, bei gegebenen Startintervall Δs, wie folgt:

1. Zu Beginn und jeweils nach Verstreichen von Δs Zeiteinheiten starten von jedem Knoten Scouts einer neuen Generation. Dabei startet pro Generation von jedem Knoten aus jeweils ein Area-Scout (unendliche Reichweite, aber auf den Ursprungsbereich begrenzt), von den Entry-Randknoten jeweils ein FZ-Scout (endliche Reichweite, geringer als der Durchmesser γ) und von den Repräsentativen Knoten jeweils ein FR-Scout (Reichweite gleich des Durchmessers bzw. unendlich).

2. Erreicht ein Scout einen Knoten (vgl. Abbildung 6.14), wird zunächst geprüft, ob bereits ein Scout jüngerer Generation und desselben Ursprungs diesen Knoten erreicht hat. Falls ja, stirbt der Scout umgehend.

3. Danach wird geprüft, ob der Scout der Erste einer neuen Generation ist. Fall ja, werden alle alten Tabelleneinträge bzgl. des Scoutursprungs gelöscht.

4. Dann aktualisiert der Scout seine Kosten, indem die Kosten des letzten Links zu seinen bisherigen Kosten addiert werden.

5. Anschließend aktualisiert der Scout die Routingtabelle (je nach Typ entweder die Area- oder Net-Tabelle).

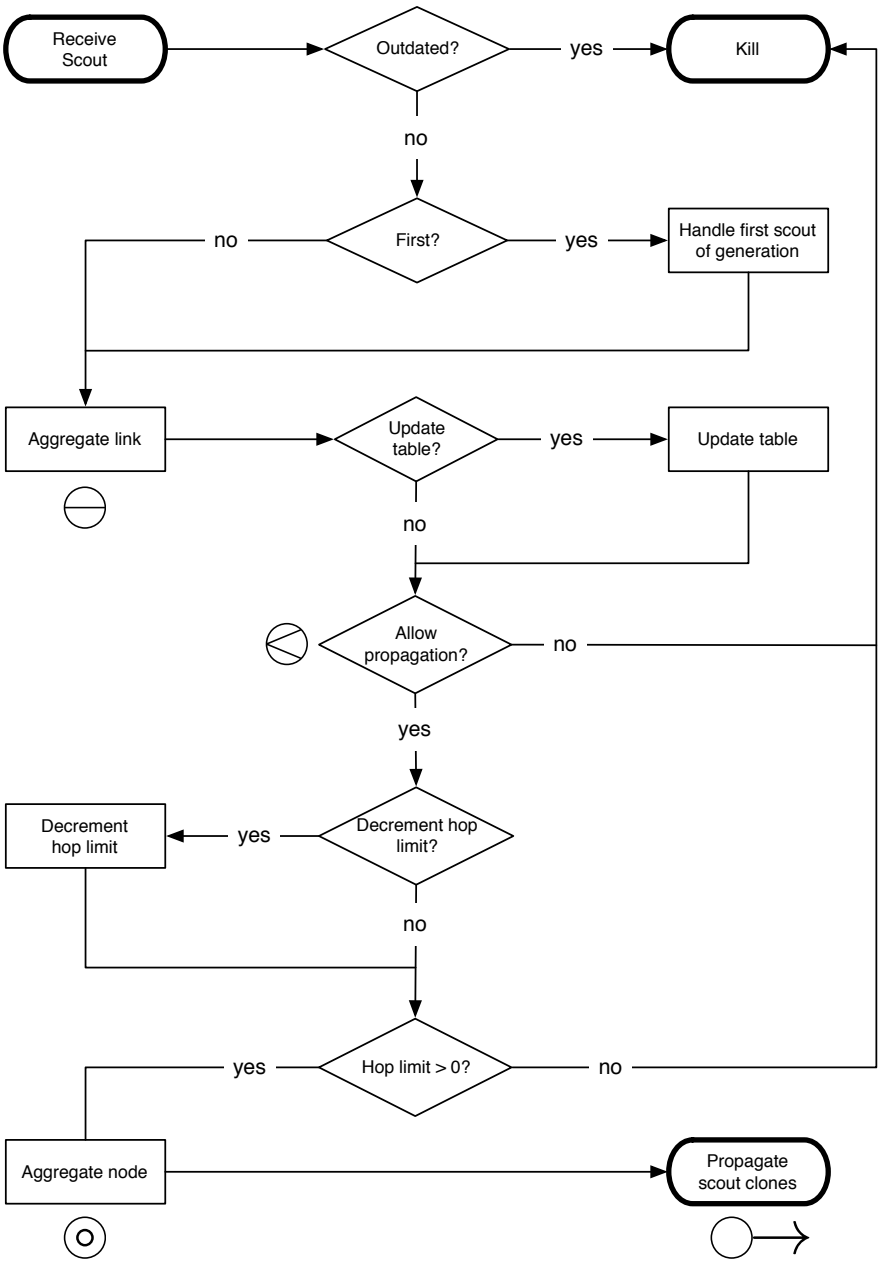


Abbildung 6.14.: Ablauf beim Empfang eines Scouts

6. Anschließend wird geprüft, ob ein Scout desselben Ursprungs und derselben Generation bereits von diesem Knoten aus weitergeleitet wurde, oder ob die Reichweite des Scouts erschöpft ist. Falls nur eins zutrifft, stirbt der Scout.

7. Der Scout darf weiter fliegen, falls entweder er über den besten Vorgänger der vergangenen Generation kommt, oder die Scoutkosten geringer sind als die besten Kosten der vergangenen Generation.

8. Darf der Scout weiterfliegen, wird die Menge der nächsten Ziele bestimmt: Die Vorgängerknoten

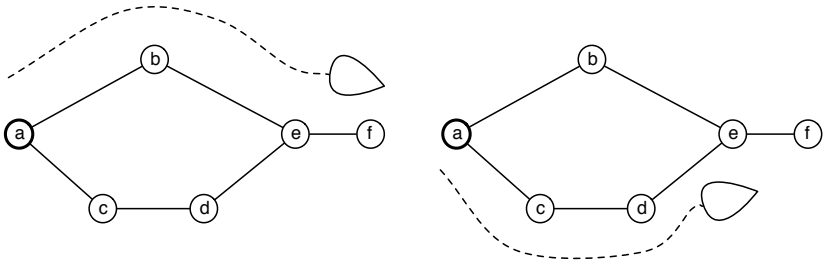


Abbildung 6.15.: Disseminationsproblem bei gleichbleibender Reichweite

exklusive des Knoten von dem der Scout kam. Anschließend fliegen neue, replizierte Scout-Klone zu diesen Knoten weiter.

Wirklich über das Kommunikationsnetz versendet werden Scouts im letzten Schritt allerdings nur, wenn der Link zu einem Vorgänger in einer anderen Area führt. Area-Scouts bleiben z.B. gänzlich im Ursprungsbereich, der zuständige Navigator muss somit nur rein intern vermerken, an welchem Knoten ein Scout sich befindet. Bevor ein Scout weitergesendet wird, muss ggf. noch sein Hop Limit reduziert werden. Auf dem Area Layer, wird pro traversiertem Link, das Limit dekrementiert (wobei in dem Basisprotokoll die Reichweite dort immer unlimitiert ist). Auf dem Net-Layer wird die Reichweite allerdings über die Anzahl der zurückgelegten Areas bestimmt. Der Grund hierfür ist einfach: erreicht ein Scout eine Area, kann dieser vom Navigator ohne weitere Inanspruchnahme des Kommunikationsnetzes innerhalb dieser verbreitet werden kann. Der Agent kann sich innerhalb einer Area folglich rein lokal verbreiten, ohne die Nachrichtenkomplexität zu erhöhen.

Zur Aktualisierung der Tabellen des Area-Layers könnte sogar pro Area vom jeweiligen Navigator ein zentraler Algorithmus eingesetzt werden. Für das Forwarding des Basisprotokolls sind nämlich nur die Tabelleneinträge mit den geringsten Kosten relevant, so dass leicht ein All-Pairs-LCP-Algorithmus genutzt werden kann, um die Tabellen des Area-Layers zu füllen. Für große (Area-)Graphen ist das etwas schneller, da eine verteilte Implementierung des MAS, wenn auch nur lokal genutzt, größeren Overhead erzeugt und praktisch eine größere Laufzeit aufweist als bspw. der Floyd-Warshall-Algorithmus[7].

Für die reibungslose Ausführung des oben genannten Vorgehens sei zusätzlich gefordert, dass eine *atomare Abarbeitung*, zumindest bzgl. des Ursprungs, der Scouts stattfindet. Anders ausgedrückt: Es werden zu keinem Zeitpunkt an einem Knoten zwei Scouts desselben Ursprungs gleichzeitig bearbeitet. So ist sichergestellt, dass in Schritt 3 nur ein Scout pro Generation als Erster erkannt wird (andernfalls kann es durch Race Conditions zu dem Entfernen bereits eingetragener Kosten kommen). Haben zwei Scouts jedoch einen unterschiedlichen Ursprung, ist eine parallele Bearbeitung erlaubt.

Das so beschriebe Scout-Verhalten weist jedoch noch ein Problem auf. Dazu sei Abbildung 6.15 betrachtet. Dort abgebildet ist ein einfaches Netz, bei dem zunächst (links dargestellt) der Scout mit Hop Limit $l = 3$ einen Pfad über Knoten b zu f etabliert. Es sei angenommen, dass in den folgenden Generationen (rechts dargestellt) der Scout über c und d als Erstes an e

[7] Technisch liegt das an der typischerweise auf Arrays basierenden Implementierung des Floyd-Warshall-Algorithmus. Das MAS allokiert für jeden Scout hingegen neuen Speicher. Man könnte das MAS für den Spezialfall des Area-Layers auch zentral umsetzen, was allerdings zu einem Vorgehen sehr ähnlich des zentralen Bellman-Ford-Algorithmus führte. Die Laufzeit zur Bestimmung aller LCPs in einem Bereich läge dann bei $\mathcal{O}(|V|^2|L|)$ (im Gegensatz zu $\mathcal{O}(|V|^3)$ des Floyd-Warshall-Algorithmus).

ankommt. In Schritt 6 des obigen Ablaufs würde festgestellt, dass die Reichweite erschöpft sei und der Scout nicht weiter zu f dürfte. Der Knoten f erführe so nicht von Änderungen der Kosten. In diesem sehr einfachen Beispiel hat das keinerlei Auswirkungen, da f Fahrzeuge zu e weiterleiten muss, da es keinen anderen Weg gibt. Aber angenommen, es gäbe Alternativpfade $f \overset{\pi}{\rightsquigarrow} a, e \notin \pi$, dann würden Forwarding-Entscheidungen an f basierend auf veralteten Informationen getroffen. Dann könnten langsamere Pfade gewählt werden als nötig oder im schlimmsten Falle sogar Kreise entstehen. Die Reichweiten-Entscheidung in Schritt 6 muss dies berücksichtigen. Dazu annotiert jeder weiterfliegende Scout den aktuellen Knoten mit der Länge seines zurückgelegten Pfades $o \rightsquigarrow i$ ausgehend von Ursprung o, sofern diese Länge besser ist als die bisherig bekannte Länge. Weiterfliegende Scouts erhalten entsprechend einen Bonus zu ihrer verbleibenden Reichweite, um alle jemals bereits angeflogenen Knoten in jeder Generation erreichen zu können. Im Beispiel würde der Scout über b an Knoten e die Länge 2 hinterlegen. Später, wenn der Scout über c und d einträfe, ersetzt der Scout seine zurückgelegte Pfadlänge durch 2 und erhält somit die Möglichkeit auch noch Knoten f zu erreichen. Wäre die Situation umgekehrt, zunächst erreicht immer der untere Scout e, hinterließe der obere Agent erst die kürzere Länge, nachdem er das erste Mal zu f weiterflöge (folglich wenn die Pfadkosten $e \to b \to a$ besser sind als über $e \to d \to c \to a$). Ab diesem Zeitpunkt wird f stets erreicht – auch wenn der untere Scout e zunächst erreicht. Man beachte, dass dadurch der Bereich des Netzes, der über einen Scoutursprung o informiert ist, dennoch nicht größer wird als dessen l-Umgebung (mit l als Hop Limit). Zwar legen einzelne Scouts ggf. einen längeren Weg zurück als ihr individuelles Limit erlauben würde, jedoch maximal zu Knoten, die von anderen Scouts desselben Ursprungs ebenfalls erreichbar sind. Im Rahmen dieses Basisprotokolls kann diese Problematik jedoch nur bei FZ-Scouts auftreten, da die beiden anderen Scouttypen jeweils mit unendlicher Reichweite (bzw. entsprechend des Durchmessers) ausgestattet sind.

Der Automat 7 zeigt den TIOA des Empfangsprozesses des Basisprotokolls (siehe auch Automat 8).

Da pro Generation über jeden Link nur eine Scout gesendet wird, ist die Nachrichtenkomplexität linear beschränkt pro Generation. Es soll ergänzend gezeigt werden, dass bei konstanten Linkkosten unter realistischen Voraussetzungen eine linear beschränkte Anzahl von Generationen zur Konvergenz notwendig ist.

Satz 6.5. Angenommen, das Startintervall Δs ist groß genug, dass Scouts der Generation n erst nach denen der Generation $n-1$ an den Knoten ankommen und ferner, dass die Linkkosten in der Zwischenzeit konstant bleiben. Dann ist Konvergenz (ausgehend von beliebigen Tabelleneinträgen) zu LCP-Einträgen nach spätestens γ Generationen an allen Knoten eingetreten, mithin das verteilte All-Pairs-Shortest-Path-Problem gelöst.

Werden sekündlich Scouts gefloodet, hat eine Generation folglich eine Sekunde „Vorsprung“ zur Ausbreitung, was selbst im öffentlichen Internet für transatlantische Round Trips genügend ist. Es kann demnach davon ausgegangen werden, dass ob des Vorsprungs i.d.R. Scouts in der Reihenfolge ihrer Generation ankommen, trotz der Asynchronität.

Bevor der Satz bewiesen wird, zunächst folgendes Lemma:

Lemma 5. Auf einem LCP $\pi = (v_0, \ldots, v_n)$ ist nach spätestens n Generationen an jedem Knoten $v_i, 1 \leq i \leq n$ ein LCP-Pfadknoten bekannt.

Beweis. Induktion über n. Nach der ersten Generation ist v_1 aufgrund des eingegangenen Scouts von v_0 ein LCP-Pfadknoten, nämlich $v_1 \to v_0$, in die Tabelle an v_1 eingetragen, weswegen

Automaton 7 NonRoot-Prozess des Basisprotokoll

Automaton BeeJamANonRoot(i) **extends** Flooder(i, P_i)

States:

$\forall v \in V : minHopCount[v] \leftarrow \infty$

$\forall v \in V : latestGen[v] \leftarrow 0$

$\forall v \in V : visited[v] \leftarrow false$

Transitions:

Input receive(scout, j, i)

Effects

if scout.gen $<$ $latestGen[scout.src]$ **then** return ▷ Outdated?

if scout.type = AREA **then** $T \leftarrow \mathfrak{A}$ **else** $T \leftarrow \mathfrak{N}$

if scout.gen $>$ $latestGen[scout.src]$ **then** ▷ First?

$\hat{s} \leftarrow \arg\min_{s \in \overrightarrow{S}_i} \{T[scout.src][s]\}$

lcpSucc[scout.src]$\leftarrow \hat{s}$

lcpCost[scout.src]$\leftarrow T[scout.src][\hat{s}]$

$\forall s \in \overrightarrow{S}_i$: $T[scout.src][s] \leftarrow \infty$

$latestGen[scout.src] \leftarrow scout.gen$

$\forall v \in V$: $visited[v] \leftarrow false$

$scout.cost \leftarrow scout.cost + cost(scout, i, j)$ ▷ Aggregate link $\ominus$

$T[scout.src][j] \leftarrow scout.cost$

if visted[bee.src] = false $\wedge$ (scout.gen = 1 $\vee$ scout.last = lcpSucc[scout.src]

$\vee$ scout.cost $<$ lcpCost[scout.src]) **then** ▷ Allow propagation $\ominus$?

if A(i) $\neq$ A(scout.last) **then** ▷ Different area?

scout.limit $\leftarrow$ scout.limit - 1 ▷ Decrement hop limit

if minHopCount[scout.src] $<$ scout.limit **then**

scout.limit $\leftarrow$ minHopCount[scout.src]

else

minHopCount[scout.src] $\leftarrow$ scout.limit

if scout.limit > 0 **then** ▷ Hop limit > 0?

$scout.cost \leftarrow \arg\min_{s \in \overrightarrow{S}_i} T[scout.src]$ ▷ Aggregate node $\circledcirc$

floodQueue $\vdash [scout, \{i\}, scout.limit]$ ▷ Propagate $\bigcirc\!\!\rightarrow$

visited[bee.src]$\leftarrow$ true

End

der Induktionsanfang für $n = 1$ gilt. Angenommen, bis v_n sei nun ebenfalls jeweils ein LCP-Pfadknoten eingetragen. Dann wird auf einem längeren LCP mit $n + 1$ Knoten während der Generation $n + 1$ aufgrund der Best-Propagierungsstrategie ein Scout über (v_n, v_{n+1}) mit Ursprung v_0 den Knoten v_{n+1} erreichen, wodurch dort ebenfalls ein Pfadknoten mit LCP-Kosten in die Tabelle von v_{n+1} eingetragen wird und der Induktionsschluss durchgeführt ist. □

Automaton 8 Hilfsfunktion

```
1: procedure COST(scout, i, j)
2:     if scout.type = AREA then
3:         return ω_ij
4:     else
5:         if j ∈ A(i) then
6:             return min_{s∈S_i} 𝔄_i^{sj}
7:         else
8:             return ω_ij
9: end procedure
```

Automaton 9 Root des Basisprotokolls

```
1: Automaton BeeJamARoot(i, Δs, d, type) extends Flooder(i, P_i)
2:
3:     States:
4:         gen ← 1
5:
6:     Transitions:
7:         Internal flood()
8:             Preconditions
9:                 clock modulo Δs = 0
10:            Effects
11:                scout ← [gen, i, 0, d, type]        ▷ d: Initial Hop Limit, type: AREA|NET
12:                floodQueue ⊢ [scout, ∅, scout.limit]
13:                gen ← gen + 1
14:
15: End
```

Beweis zu Satz 6.5. Es genügt o.B.d.A., ein einzelnen Scoutursprung v_0 zu betrachten, da Scouts unterschiedlicher Ursprünge nicht interagieren. Es wird ferner zunächst ein vollständig flaches, hierarchieloses Netz vorausgesetzt. Dann existiert zu jedem $v \in V$ mindestens ein LCP $v_0 \rightsquigarrow v$, weswegen Lemma 5 gilt. Für das flache Netz folgt wegen $n \leq \gamma$ die Behauptung. Da der Area Layer ebenfalls flach ist, gilt die Behauptung somit für alle Areas. Auf dem Net Layer werden Scouts von Rand- zu Randknoten weitergeleitet, wobei innerbereichliche Linkkosten aus den Tabellen des Area Layers stammen, vom Prinzip aber ein genauso flaches Netz existiert. Insgesamt folgt die Behauptung des Satzes. □

Aus der faktoriellen Anzahl an Nachrichten des AsyncBF ist so durch eine Art Runden-Synchronisierung ein Protokoll mit linearer Anzahl an Nachrichten pro Generation und maximal $\gamma \leq |V|$ Generationen geworden. Der Vorteil ist, dass zu jedem Zeitpunkt die Anzahl der Nachrichten konstant ist und die Genauigkeit der Einträge sich von Generation zu Generation verbessert, wie auch eine spätere simulative Evaluation zeigt.

Die Automatenbeschreibungen für den Root-Prozess, das Forwarding, den Navigator, das Fahrzeug bzw. den Forager und des BeeJamA-Systems finden sich in den Automaten 9, 10, 11, 12 und 13.

Automaton 10 Forwarding des Basisprotokolls

1: **Automaton** Forwarding(i)
2: **States:**
3: $vehicleID$ ▷
4:
5: **Transitions:**
6: **Input** requestNextHop(i, d, vID) ▷ d: destination
7: **Effects**
8: $nh1 \leftarrow \oplus \left(\bigcup_{s \in \overrightarrow{S}_i^d} \left(s, \mathfrak{A}_i^{sd} \right) \right)$ ▷ Case 1
9: ▷ Case 2 ($\min \mathfrak{A}_e^d$ has to requested from d's navigator)
10: $nh2 \leftarrow \oplus \left(\biguplus_{\substack{e \in E_{A(d)} \\ f \in F_{A(i)} \\ s \in \overrightarrow{S}_i^f}} \left(s, \mathfrak{A}_i^{sf} + \min \mathfrak{N}_f^e + \min \mathfrak{A}_e^d \right) \right)$
11: nextHop $\leftarrow \left(\underset{nh \in \{nh1, nh2\}}{\arg\min} \{(nh)_2\} \right)_2$ ▷ $(...)_2$ refers to the 2nd element
12: **if** nextHop = nil **then**
13: nextHop $\leftarrow \left(\oplus \left(\biguplus_{\substack{f \in F_{A(i)} \\ s \in \overrightarrow{S}_i^f}} \left(s, \mathfrak{A}_i^{sf} + \min \mathfrak{N}_f^{\mathfrak{M}^d} \right) \right) \right)_2$ ▷ Case 3
14: vehicleID $\leftarrow$ vID
15:
16:
17: **Output** receiveNextHopResponse(nextHop, vehicleID)
18: **Preconditions**
19: nextHop $\neq$ nil
20: **Effects**
21: nextHop $\leftarrow$ nil
22:
23: **End**

Die Beschreibungen dieses Abschnittes zeigen, dass das Basisprotokoll kein Count-to-Infinity-Problem aufweist (vgl. Abschnitt 5.5.2), da nicht wahllos andere Kosten aus der lokalen Tabelle herangezogen werden, um die Fahrzeiten zum Ziel abzuschätzen bzw. um diese vermeintlichen Kosten sogar zu disseminieren. Auch entstehen keine Kreise: ein Fahrzeug kann erst über einen Pfadknoten geleitet werden, wenn ein Scout der aktuellen Generation diesen Pfadknoten als einen Teil eines Pfades erkundet hat. Durch diese Erkundung liegen konsistente Start-Ziel-Tabelleneinträge an allen Knoten des Pfades vor. Da nur günstigste Pfadknoten selektiert

Automaton 11 Navigator

Automaton Navigator($navID, A, B, r, d$)
  States:
    $\mathfrak{A} \leftarrow [][], \mathfrak{N} \leftarrow [][]$

  $\forall v \in A$: BeeJamARoot(v, $\Delta s, \infty$, AREA), BeeJamANonRoot(v) ▷ Area nodes
  $\forall v \in B \setminus \{r\}$: BeeJamARoot(v, $\Delta s, d$, NET) ▷ Border nodes
  BeeJamARoot(r, $\Delta s, \infty$, NET) ▷ Representative node
  Forwarding(navID)

End

Automaton 12 Vehicle

Automaton Vehicle($vehicleID$)
  States:
    nextHop ← nil

  Transitions:
    Output requestNextHop(i, dst, vehicleID)
      Preconditions
        nextHop = nil
      Effects

    Input receiveNextHopResponse(nextHop, vehicleID)
      Effects
        nextHop ← nextHop ▷ Display next hop on PNA

    Internal approachingJunction()
      Preconditions
        "Approaching junction?" ▷ External event
      Effects
        nextHop ← nil
        navID ← "local navigator id"

End

Automaton 13 BeeJamA System

Automaton BeeJamA($Areas, d$)
  $\forall A \in Areas$: $B \leftarrow A \setminus \{v_0 | \nexists v_1 \in V \setminus A : (v_0, v_1) \in L\}$, Navigator($navID{+}{+}, A, B, d, B[0]$))
  $\forall (i, j) \in L, A(i) \neq A(j)$: $TimedChannel(i, j)$
End

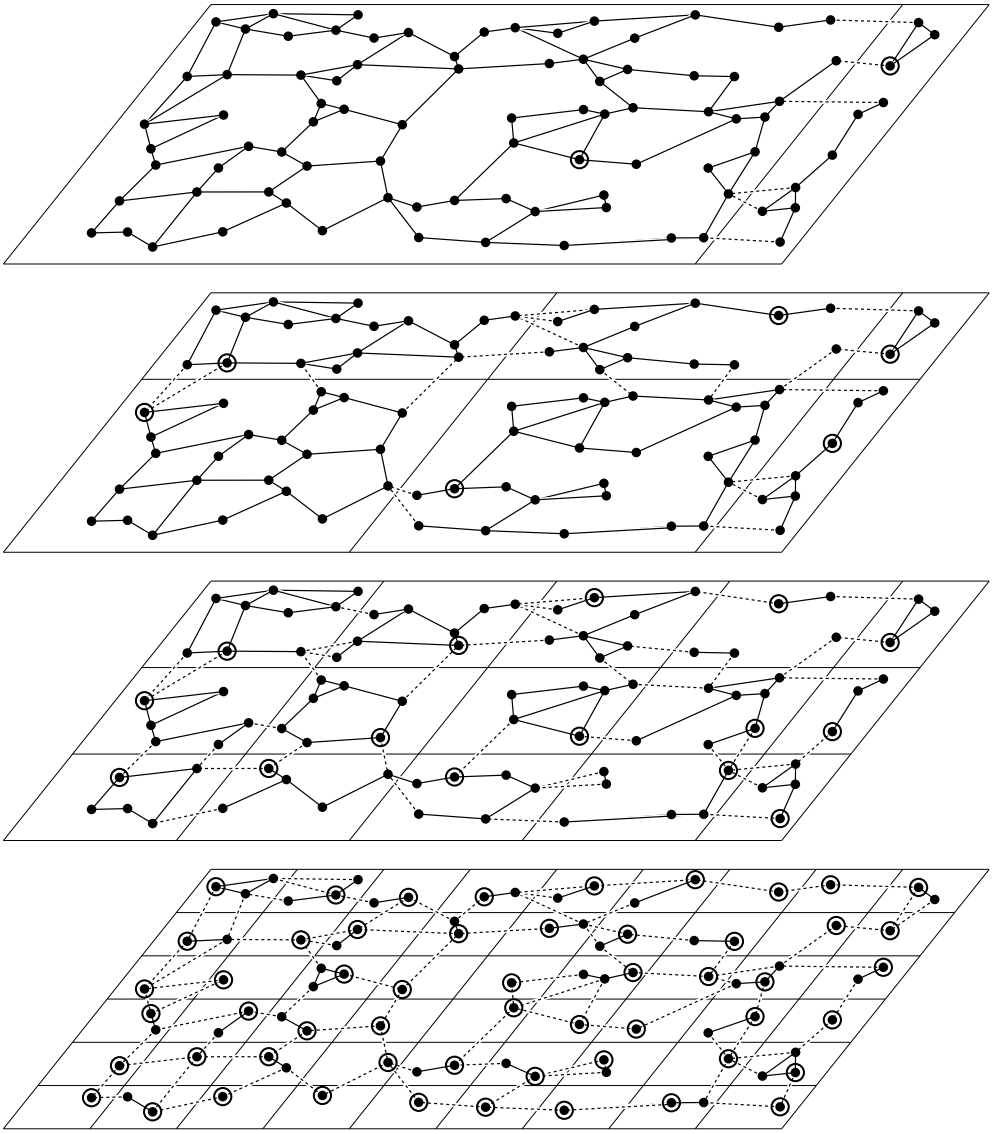


Abbildung 6.16.: Konzept der zusätzlichen Hierarchie

werden, erreichen Fahrzeuge in endlichen Schritten kreisfrei ihr Ziel. Da an jeder Kreuzung neu auf Basis aktueller Fahrzeiten entschieden wird, entsteht ein Forwarding gemäß dynamisch-konsekutiver Kosten (vgl. Gleichung 1.5).

Damit ist das Basisprotokoll beschrieben, der nächste Abschnitt führt zusätzliche Hierarchie ein.

6.8. Zusätzliche Hierarchie

Im Folgenden sei das dichotome Hierarchiekonzept derart erweitert, dass beliebig viele Zwischenabstufungen, genannt *Level*, möglich sind. Auf jedem Level wird die Anzahl der repräsentativen Knoten reduziert und die Reichweite der emittierten Scouts erhöht. Der entstehende Vorteil liegt in der Reduzierung der benötigten Nachrichtenanzahl, da eine geringere Anzahl von

(repräsentativen) Knoten Scouts mit großen (ggf. unendlichen) Reichweiten versenden muss. Zeitgleich stellt dies jedoch auch eine Vergröberung der verbreiteten Informationen dar. *Schlussendlich handelt es sich um eine Abwägung zwischen Präzision und zur Verfügung stehender Bandbreite.*

Um zusätzliche Hierarchie hinzufügen zu können, wird das oben eingeführte Prinzip beibehalten: ein repräsentativer Knoten ist für einen geographischen Bereich „verantwortlich“ und informiert die Knoten in diesem Einflussbereich mittels Scouts über seine Erreichbarkeit. Auf dem untersten Level soll das zugehörige Gebiet eines repräsentativen Knotens die geringste Ausdehnung aufweisen. Ebenso soll die Reichweite der Scouts, die von diesem Knoten auf diesem niedrigsten Level emittiert werden, am geringsten sein. Beides, Gebietsausdehnung und Reichweite, erhöht sich sukzessive mit jedem höheren Level, wobei schließlich auf dem höchsten Level die Scout-Reichweite wieder unlimitiert ist, wie bei den FR-Scouts im dichotomen Fall.

Um die Level online zu erstellen, wird der lokale Leader Election Mechanismus nach Abschnitt 5.3 verwendet. Ausgehend von einem untersten Level 1 – beispielsweise mit einem repräsentativen Knoten pro Area und endlicher Reichweite l – führen die repräsentativen Knoten dieses Levels untereinander eine Leader Election mit Reichweite l auf dem Communication Layer durch. Die so bestimmten Leader sind dann die repräsentativen Knoten des nächsten höheren Levels und haben eine Reichweite $l' > l$.

Dieser Prozess kann fortgesetzt werden, wie jedoch bereits erwähnt, muss die Reichweite des höchsten Levels stets unlimitiert, $l = \infty$, sein. Prinzipiell kann jeder Knoten ein repräsentativer Knoten werden (das ließe sich a priori nur mit globalem Wissen entscheiden), somit müssen an jedem Knoten (vermöge des zugehörigen Navigators) die entsprechenden Reichweiten aller Levels bekannt sein, d.h. in Form einer Konfiguration vorliegen. Konzeptionell könnte dieser Leader Election-Prozess zur Ausformung der Hierarchie mehrfach zur Laufzeit durchgeführt werden, bspw. wenn Navigatoren und damit repräsentative Knoten ausfallen oder Lastspitzen auftreten und dabei auch die Reichweiten variable an zur Verfügung stehende Bandbreite oder Rechenkraft angepasst werden. Im Evaluationsteil dieser Arbeit wird allerdings von einer einzigen, initialen Durchführung ausgegangen, so dass die Level-Konfiguration während der gesamten Laufzeit statisch ist.

Abbildung 6.16 verdeutlicht das Konzept. Auf der untersten Ebene sind die üblichen Areas mit jeweils einem repräsentativen Knoten angesiedelt. Auf den höheren Ebenen sind Bereiche tiefer liegender Ebenen jeweils zusammengefasst. Dadurch reduziert sich die Anzahl der repräsentativen Knoten, wodurch die Anzahl der emittierten FR-Scouts sinkt, die Ungenauigkeit beim Forwarding aber zunimmt.

Elementar für die Funktionsweise zusätzlicher Ebenen ist die Aufrechterhaltung einer *Forwarding-Kette*, d.h. dass über die verschiedenen Level hinweg stets garantiert ist, dass an dem aktuellen Knoten ein Fahrzeug weiter in Richtung des Ziels geroutet wird, dass also entsprechende Tabelleneinträge vorhanden sind. Dies ist gewährleistet, da die Leader Election immer so ausgeführt wird, dass an einem Leader des Levels n alle Leader des Levels $n-1$ in l-Reichweite bekannt sind. Spätestens wenn das Fahrzeug demnach einen Level n Leader erreicht, ist dort bekannt, welcher Leader als nächstes angesteuert wird, bis schließlich die Foraging Zone des Ziels erreicht ist und somit Fall 2 eintritt. In der Regel wird das aufgrund der überlappenden Einflussbereiche aber schon früher der Fall sein, bevor der repräsentative Knoten (Leader) erreicht ist.

Da zusätzliche Hierarchieebenen dynamisch zur Laufzeit entstehen bzw. sich ändern können, müssen die Membership-Tabellen dynamisch angepasst werden. Die repräsentativen Knoten verbreiten vermöge ihrer Long Distance Scouts ihren Leader-Status inkl. Level, wodurch

„unterlegene“ Areas abhängig vom Level die Knotenzugehörigkeit zu den repräsentativen Knoten lernen. Für die Membership-Tabelle wird ein zusätzlicher Index l eingeführt, $\mathfrak{M}_l$, der den assoziierten repräsentativen Knoten für Level l entspricht.

Die Änderung im Forwarding sind dann leicht vorzunehmen: In Zeile 10 des Automaten 10 muss die Zugehörigkeit in $\mathfrak{M}_{\check{l}}$ statt in $\mathfrak{M}$ nachgesehen werden, mit $\check{l} = \arg\min_{l \in \mathbb{N}} \mathfrak{M}_l^d \neq nil$.

Bei den Beschreibungen der nächste Abschnitte wird aus Gründen der Übersicht auf die Berücksichtigung von Hierarchie verzichtet. Technisch lassen sich die vorgestellten Hierarchiekonzepte leicht hinzufügen, nur verkompliziert es die Darstellungen deutlich, statt den Fokus auf die Kernelemente richten zu können. Daher werden die Verfahren so erläutert, als ob es eine flache Struktur gäbe. Dementsprechend gibt es am Knoten i nur eine Tabelle T_i.

6.9. Suboptimales Forwarding und Kreise

Die Forwarding-Operation $\oplus$ des zuvor beschriebenen Basisprotokolls selektiert den günstigsten Tabelleneintrag als Next Hop auf einem Ex Ante-LCP (min-Selektion). In diesem Sinne handelt es sich bei dieser Greedy-Wahl um ein optimales Forwarding, da der lokal beste Eintrag verwendet wird. Aufgrund der potentiellen Diskrepanz zwischen Ex Ante- und Ex Post-LCP, ob der flussabhängigen Linkkosten, muss sich solch eine Entscheidung weder für den einzelnen Fahrer individuell (i.S.d. UE), noch global (i.S.d. SO), als optimal bzgl. der Fahrzeiten herausstellen. Denn für einen einzelnen Fahrer hätte es in der Retrospektive sinnvoller sein können, erst eine gestaute Route vorzuziehen, wenn diese später entlastet würde und so eine schnellere Ankunft ermöglicht hätte. Nur ist diese Voraussicht in dynamischen und offen Systemen wie dem Straßenverkehr nicht leicht realisierbar. Zumal es sich um eine selbstzerstörende Prophezeiung handeln kann, denn sofern die Vorhersage öffentlich wird, richten sich Fahrer danach und der Stau verlagert sich. In Abschnitt 6.11 wird ein Pfadreservierungskonzept vorgestellt, dass einen Versuch darstellt, Voraussicht in einem verteilten System zumindest im Ansatz umzusetzen.

Dennoch gilt im Allgemeinen, dass es sinnvoll sein kann, dass Fahrer schlechtere Routen bevorzugen, in dem suboptimale Einträge gewählt werden. Wie das Braess-Paradoxon (vgl. Abschnitt 4.2.2) zeigt, kann dies von erheblichem Vorteil sein.

Eine Möglichkeit suboptimale Pfade zu wählen, ist die stochastische Auswahl. Im natürlichen Vorbild der Bienen findet sich interessanterweise eine ähnliche Kompromisssituation, die dort ebenfalls durch eine stochastische Selektion angegangen wird. Sammlerinnen wählen nicht zwangsläufig die am intensivsten beworbene Futterquelle aus, sondern eine proportinal zur Qualität der Quelle. So werden nicht zu viele Sammlerinnen für eine einzelne Quelle rekrutiert, man könnte sagen, es wird eine Art von Stauvermeidung betrieben, ein dezentrales Load Balancing. Mit geringerer Frequenz werden aber auch schlechtere Quellen anvisiert, wodurch die dortige Umgebung ebenfalls stetig exploriert und zu Sammelzwecken besucht wird. So maximiert zwar nicht die einzelne Sammlerin ihren Ertrag, aber das Volk insgesamt. Ameisen kennen bei der Pfadfindung ein ähnliches Konzept [31]: Dynamische Umgebungen erfordern, dass die Pfadexploration auch auf zuvor als schlechter identifizierten (Alternativ-)Pfaden nicht vollständig eingestellt wird. Daher wird nicht nur der höchsten Pheromonspur gefolgt, sondern ebenfalls in Abhängigkeit aller Möglichkeiten gewählt.

Für das Routing in Computernetzen wurden schon des Öfteren Protokolle entwickelt, die statt einer Greedy-Selektion, mittels stochastischer Routenwahl lokal suboptimale Forwarding-Entscheidungen treffen, mit der Intention der Durchsatzmaximierung. Diese Ansätze sind

dabei durch schwarmintelligente Ideen geprägt, wie BeeHive oder AntNet, oder wurden auch vollkommen losgelöst davon entwickelt.

Allerdings entsteht durch dieses suboptimale, stochastische Forwarding ein (gewichteter) Random Walk auf den Pfaden, die durch die bisherigen Tabelleneinträge möglich sind. Da stochastisches Forwarding erst sinnvoll ist, wenn auch mehrere Next Hop-Alternativen zur Verfügung stehen, ist daher inhärent mit Tabellenkreisen zu rechnen.

Der nächste Abschnitt zeigt auf, wie stochastisches Forwarding umgesetzt werden kann. Die beiden anschließenden Abschnitte erläutern Gegenmaßnahme zu auftretenden Kreisen.

Es sei allerdings schon an dieser Stelle darauf hingewiesen, dass das Ergebnis dieses Abschnitts negativ ist. Eine Verbesserung der Fahrzeiten gegenüber des Basisprotokolls konnte empirisch nicht festgestellt werden. Dennoch klären die folgenden Ausführungen eine der elementaren Designentscheidungen des Protokolls, nämlich gegen suboptimales Forwarding.

6.9.1. Stochastisches Routing

Für Computernetze wurden Routingprotokolle [22, 34, 45, 89, 116, 165] vorgeschlagen, die eine stochastische statt deterministische Forwarding-Operation $\oplus$ verwenden. Argumentativ wird dieses Vorgehen z.B. bei BeeHive dadurch bekräftigt, dass so Nachteile des Selfish-Routings – nicht optimale globale Fahrzeiten – abgemildert werden könnten. Würden Datenpakete über suboptimale Pfade geleitet, entstünde so ein Fluss mit geringerem Potential zur Überlastung von Links, da nicht alle Pakete über ein und denselben Pfad flössen. (Ob eine Senkung der individuellen Fahrzeiten erreicht wird, wird nicht explizit diskutiert in den genannten Quellen).

Die zentrale These: Der günstigste Pfad wird am häufigsten verwendet und wird daher am schnellsten überlastet. Der über einen suboptimalen Pfad geleitete Fluss, nutzt die begrenzten Ressourcen des optimalen Ex Ante-Pfades nicht und trägt somit nicht zur dräuenden Überlast bei. Dabei wird implizit die Vorhersage getätigt, das eine Überlastung eines optimalen Ex Ante-Pfades zu schlechteren Fahrzeiten führt und daher vermieden werden muss. Höhere Fahrzeiten eines Flussanteils wird dabei in Kauf genommen. Da weiterhin aktuelle Verkehrsinformationen verwendet werden, handelt es sich weiterhin um eine CCP-Strategie, die jedoch implizit prädiktiv interpretiert wird.

Diesem Ansinnen folgend, soll nun statt einer deterministischen Greedy-Wahl (z.B. min-Forwarding) die Selektion des Next Hops gewichtet nach Kosten des potentiellen Next Hops $s \in \overrightarrow{S}$ geschehen.

Dieses Konzept sei als ϕ-Forwarding bzeichnet, mit ϕ als Selektionswahrscheinlichkeit von s. Die Idee des stochastischen Forwardings: Je geringer die Kosten eines Next Hops auf einem explorierten Pfad zum Ziel, desto höher die Selektionswahrscheinlichkeit. Demnach kann als Bewertung eines Pfadknotens s zum Ziel d die reziproken Kosten $B_s = \frac{1}{T_i^{sd}}$ verwendet werden. So wird zumindest auch ein Teil des Flusses $s \rightsquigarrow d$ über suboptimale Pfadknoten weitergeleitet.

Die Next Hop-Selektion selbst kann bspw. mittels Rouletterad-Selektion umgesetzt werden, d.h. es wird eine (diskrete, kumulative) Verteilungsfunktion der reziproken, normierten ELCP-Kosten der Pfadknoten gebildet und die Inverse einer Zufallszahl $r \in [0, 1]$ bestimmt. Dadurch wird ein potentieller Next Hop proportional zu

$$\phi_s = \frac{B_s}{\sum_{s' \in \overrightarrow{S}_i^d} B_{s'}} = \frac{\frac{1}{T^{sd}}}{\sum_{s' \in \overrightarrow{S}_i^d} \frac{1}{T_i^{s'd}}} \tag{6.1}$$

selektiert.

Durch die Normierung ist $\phi_s \in [0,1]$ und wegen

$$\sum_{s \in \overrightarrow{S}} \phi_s = \sum_{s \in \overrightarrow{S}} \frac{B_s}{\sum\limits_{s' \in \overrightarrow{S}} B_{s'}} = \frac{\sum\limits_{s \in \overrightarrow{S}} B_s}{\sum\limits_{s \in \overrightarrow{S}} B_s} = 1,$$

auch ein Wahrscheinlichkeitsmaß.

Das BeeJamA-Protokoll in Gänze muss demnach nicht verändert werden, sondern nur die min-Selektion der $\oplus$-Operation durch diese ϕ-Selektion ersetzt werden.

Ein stochastisches Forwarding ist somit zwar leicht umsetzbar, jedoch spiegeln die bisher in ϕ_s einbezogenen Kosten von s dabei nicht tatsächlich die durch ein Fahrzeug erlebten wieder, da die Scouts aufgrund der bisherigen Umsetzung der Knoten-Aggregierungsoperation $\circledcirc$ nicht die potentielle Aufteilung des Verkehrsflusses berücksichtigen. Es bieten sich prinzipiell zwei Möglichkeiten an, wie ausgehende Scouts die Kosten eines Knotens mittels der $\circledcirc$-Operation bewerten können:

1. Kosten auf einem einzelnen Pfad, bspw. mittels min-Aggregierung, wie im bisherigen Basisprotokoll.

2. Gewichtete Kosten mehrerer Pfade zum Ziel, genannt ϕ-Aggregierung.

Der Unterschied wird an Abbildung 6.17 deutlich. Angenommen, Knoten a ist das Ziel, die aktuelle Fahrzeugposition sei e und die Linkkosten seien konstant. Wie im ersten Teilbild ersichtlich, bestehen die zwei Forwarding-Optionen $\delta((e,d,b,a)) = 11$, $\delta((e,d,c,a)) = 21$. Durch min-Aggregierung enthielte die konvergierte Tabelle T^* LCP-Kosten, insbesondere wären für $e \to d \rightsquigarrow a$ Kosten von 11 eingetragen. Das entspricht aber nicht zwangsläufig den erfahrenen Kosten eines Foragers über d bei ϕ-Forwarding. Denn an d könnte dann (im Gegensatz zum min-Forwarding) auch c als Next Hop gewählt werden, wodurch die erfahrenen Kosten 21 statt 11 betrügen. Stattdessen ist im Allgemeinen der Erwartungswert der Kostenzufallsvariable C_{ijd} zu betrachten, welche den Kosten von i über j zu d entspricht: $E(C_{ijd}) = \omega_{ij} + \sum\limits_{\pi \in \Pi_{jd}} \delta(\pi)\phi(\pi)$, mit Π_{jd} als Menge aller (einfachen) Pfade zwischen Next Hop j und einem Ziel d und mit $\phi(\pi)$ als Selektionswahrscheinlichkeit von π.

Es wird demnach ein Disseminationsverhalten benötigt, welches solch eine Knoten-Aggregierung ermöglicht. Das zweite Teilbild zeigt dazu die Situation, in welcher der linke Scout $\mathcal{L}$ über den Pfad $a \to b \to d$ als erstes Knoten d erreicht und zu Knoten e weitergeleitet wird. Angenommen, über den Pfad $a \to c \to d$ sei der rechte Scout noch nicht zu d gelangt. Daher ist an d nur ein Tabelleneintrag zu a vorhanden (von $\mathcal{L}$ selbst eingetragen). Die Bestimmung von $C_{\mathcal{L}}$ an d ist somit eindeutig und einfach die Summe der Pfadkosten $5+5$. Da jedoch nachfolgend auch der rechte Scout $\mathcal{R}$ den Knoten d schließlich erreicht, wären sodann zwei Einträge existent. Das zuvor beschriebene Basisprotokoll würde diese (veralteten) Einträge beim Erreichen des ersten Agenten einer neuen Generation löschen. Der hier beschrieben stochastische Ansatz speichert Einträge hingegen *generationsübergreifend*.

Im dritten Teilbild, welches die anschließende Generation von Scouts zeigt, wird das ausgenutzt. Es sei vereinfachend angenommen, dass die Linkkosten sich nicht verändert haben. Der

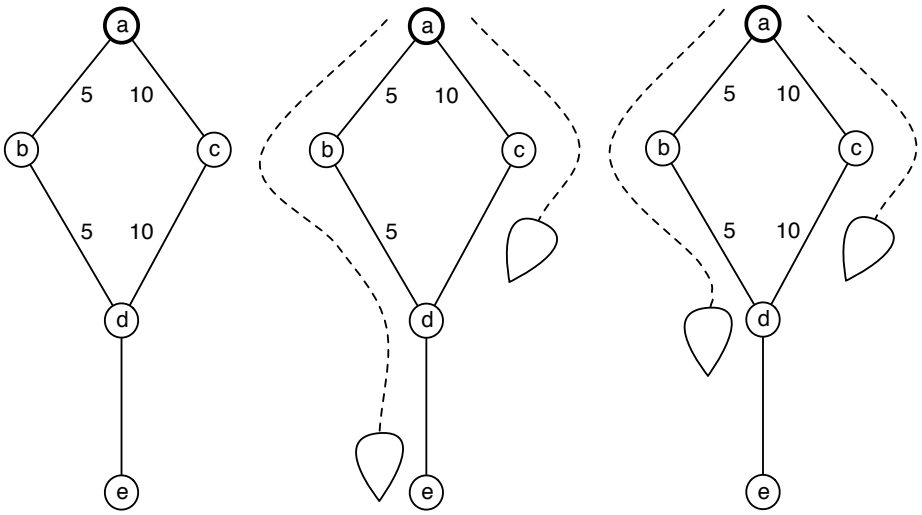


Abbildung 6.17.: Aggregierung bei mehreren möglichen Pfaden

Scout M wird von d zu e weitergeleitet, muss zuvor aber seinen Kostenwert C_M aktualisieren, kann dazu aber auch auf die Information $\delta((d,c,a)) = 20$ zurückgreifen.

Dazu werden zielgerichtete Tabelleneinträge gewichtet berücksichtigt, im Allgemeinen wird $C_\mathcal{S}$ für ein Scout $\mathcal{S}$ mit Ursprung o wie folgt bestimmt[8]:

$$
\begin{aligned}
C_\mathcal{S} = \circledcirc T_i^o := \sum_{s \in \overrightarrow{S}_i^o} \phi_i^{so} T_i^{so} = \sum_{s \in \overrightarrow{S}_i^o} \left(\left(\frac{\frac{1}{T_i^{so}}}{\sum_{s' \in \overrightarrow{S}_i^o} \frac{1}{T_i^{s'o}}} \right) \cdot T_i^{so} \right) \\
= \frac{\sum_{s \in \overrightarrow{S}_i^o} 1}{\sum_{s' \in \overrightarrow{S}_i^o} \frac{1}{T_i^{s'o}}} = \frac{|\overrightarrow{S}|}{\sum_{s \in \overrightarrow{S}_i^o} T_i^{so}}
\end{aligned}
\tag{6.2}
$$

Wie ersichtlich, bezieht solch ein Vorgehen nur Pfadkosten mit ein. Wünschenswert wäre es aber, wenn zusätzlich die Auslastungen berücksichtigt würden. Angenommen, es existieren zwei Pfadalternativen, beide mit ungefähr gleichen Kosten, jedoch sei einer gestaut, der andere leer. Mit obigem Vorgehen würde über beide Pfade ungefähr identischer Fluss geleitet, was zu einer weiteren Verschärfung des Auslastung des gestauten Pfades führen würde. Stattdessen wäre folgendes Verhalten sinnvoll:

[8] Dieses Vorgehen ist äquivalent zu BeeHive, wenngleich dort eine komplexere Vorgehensweise und gänzlich andere Herleitungserklärungen verwendet werden. Das ist in sofern verwunderlich, da die Autoren versuchen, mit möglichst wenigen CPU-Zyklen auszukommen, da dies beim Routing in Computernetzen durchaus Einfluss auf den Durchsatz haben kann, wohingegen dieser Aspekt im VRGS-Umfeld keine übergeordnete Rolle spielt. Die BeeHive-Berechnungen benötigen $\Omega(|\overrightarrow{S}|^2)$ Multiplikationen und Additionen (ableitbar aus den Darstellungen in [45, Abschnitt 3.4]), dieser Ansatz nur linear viele.

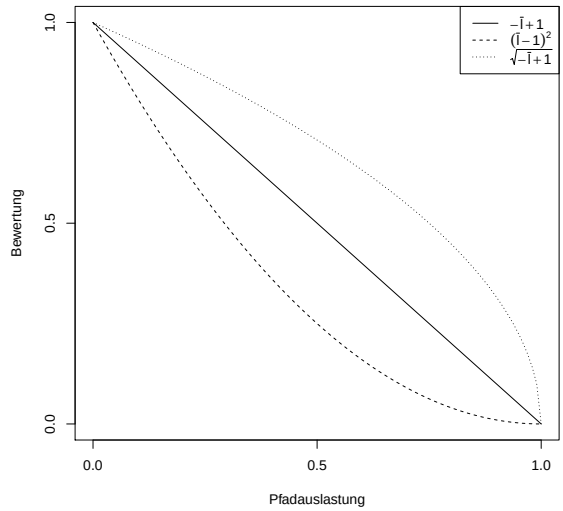


Abbildung 6.18.: Bewertungsvarianten der Pfadauslastung

- Je höher die Auslastung, desto schlechter sollte die Bewertung ausfallen und umgekehrt
- Bei voller Auslastung, wird kein zusätzlicher Fluss über ein Pfad geleitet
- Bei leerem Pfad, hängt es, wie bisher, von den Kosten ab
- Wenn auf mehreren Alternativpfaden die Auslastungen identisch sind, entscheiden die Kosten und umgekehrt

Dazu kann die *durchschnittliche Pfadauslastung* $\bar{l}$ definiert werden als die, mit den (normierten) Pfadkosten gewichtete Linkauslastung:

$$\bar{l}_\pi = \sum_{(i,j)\in\pi} L_{ij} \cdot \frac{\omega_{ij}}{\sum_{(i',j')\in\pi} \omega_{i'j'}}, \tag{6.3}$$

mit L_{ij} als Auslastung des Links (i,j) (vgl. Gleichung 3.6). Wird statt $B_s = B_{1,s} = \frac{1}{T_i^{so}}$ in Formel 6.1 bspw. eine der drei folgenden Varianten für eine gegebene durchschnittliche Pfadauslastung $\bar{l} \in [0,1]$ verwendet, sind die zuvor genannten Bedingungen erfüllt (vgl. Abbildung 6.18):

$$B_{2,s} = \frac{-\bar{l}+1}{T_i^{so}}, \qquad B_{3,s} = \frac{(\bar{l}-1)^2}{T_i^{so}}, \qquad B_{4,s} = \frac{\sqrt{-\bar{l}+1}}{T_i^{so}}. \tag{6.4}$$

Je nach Variante geht die Auslastung unterschiedlich stark in die Bewertung eines Pfadknoten mit ein. Um $\bar{l}$ allerdings bestimmen zu können, muss allerdings von einem DVP zu einem DPP gewechselt werden, wodurch die Nachrichtenlänge eines Scouts von $\mathcal{O}(1)$ auf $\mathcal{O}(\gamma)$ wächst. Ein aggregierender DVP-Scout kann nämlich in Gleichung 6.3 nicht retrospektiv auf l_{ij} und ω_{ij}

zugreifen. Stattdessen muss ein DPP-Scout ein Liste aller passierten Links inkl. dieser beiden Werte mitführen.

Die Umschichtung von Fluss – das ist ein zentraler Nachteil dieses stochastischen Ansatzes – geschieht demnach, in Abhängigkeit der aktuellen Kosten und/oder der aktuellen Auslastung. Bei einer bestimmten Kosten/Auslastung-Kombination wird ein entsprechender Anteil des Flusses über die zur Verfügung stehenden Pfade geleitet, ohne mit einzubeziehen, wie der zusätzliche Fluss tatsächlich die Transitkosten auf einem Pfad verändern würde. So könnte es sinnvoll sein, einen Next Hop auf einem Pfad zu wählen, der gegenwärtig schlechter bewertet wird, als andere. Wenn dieser Pfad aber, vorausschauend über die gesamte Fahrtzeit betrachtet, besser abschneiden würde, wäre es dennoch von Vorteil, Fluss über diesen Pfad zu leiten.

Die implizit prädiktive Hypothese des stochastischen Ansatzes, dass eine Reduzierung des Flusses auf einem hoch ausgelasteten Pfad ebenso zu einer Reduzierung der globalen Fahrzeiten führt, basiert somit, wie bereits erwähnt, auf der Genauigkeit der CCP-Strategie: Ex-Ante- und Ex-Post-Kosten müssen möglichst identisch sein. In Abschnitt 6.11 wird später ein (deterministischer) Ansatz zu einer explizit prädiktiven Bepreisung (Predictive Cost Pricing, PCP), basierend auf Pfadreservierungen, aufgezeigt, welche die genannten Nachteile verhindert.

Die Umsetzung der Überlegungen dieses Abschnitts führte aber zu Protokollen, welche in den Simulationen erhebliche Kreisprobleme aufwiesen: Fahrzeuge fuhren häufig im Kreis, die Fahrzeiten verschlechterten sich dramatisch.

Die Situation ist dabei in zweierlei Weise vom Routing in Computernetzen zu unterscheiden. Zum Einen sind Kreise dort nicht fatal, sie verringern nur die Übertragungsverzögerung und erhöhen unnötigerweise die Auslastung des Netzes. Ein Autofahrer hingegen frustriert ein Kreis im Zweifel erheblich, wodurch die Akzeptanz des Protokolls deutlichen Schaden erleiden kann. Zum Anderen weisen Straßennetze eine Struktur auf, die durchaus als anfälliger für Kreise bezeichnet werden kann. Ursächlich hierfür sind die engen Verzweigungen, z.B. in Kreuzungsbereichen. Ein Kreis verlängert die eigentlichen Kosten der Gesamtfahrzeit nicht unbedingt erheblich, folglich ist die Auswahlwahrscheinlichkeit für einen Next Hop der schließlich zu einem Kreis führt, fast identisch zu kreisfreien Alternativen. Die praktischen Umsetzungen der in diesem Abschnitt geschilderten Konzepte führte daher zu Effekten, an denen Fahrzeuge mehrfach pro Fahrt in einen Kreis gerieten, indes unbrauchbar scheinen. Im BeeHive Protokoll die Verwendung des stochastischen Routings zu Kreisen in ca. 5% aller Fälle [45, Abschnitt 9.2.2]. Ein Konzept zur Kreisvermeidung ist dort nicht implementiert. Zweifelsohne ist das ein unhaltbarer Zustand für den Kontext des Straßenverkehrs, der nächste Abschnitt diskutiert daher Auswege.

6.9.2. Auftreten von Kreisen

Einleitend werden zunächst zwei Fälle unterschieden in denen es zu Kreisen kommen kann. Der erste Fall tritt nur bei ϕ-Forwarding auf, der zweite hingegen auch bei min-Selektion.

Forwarding-bedingt: Aufgrund von ϕ-Forwarding kann das Fahrzeug bei vorliegenden Tabellenkreisen (vgl. Abschnitt 5.5.2) in Kreisen geroutet werden. Mit lokalen Informationen ist aber nicht einmal unmittelbar entscheidbar, wann Tabellenkreise entstehen, wie Abbildung 6.19 beispielhaft zeigt. Ausgehend von a werden Scouts gefloodet. Angenommen, alle Links seien bidirektional befahrbar. Der Scout über c im linken Teilbild erreiche zuerst d und wird deshalb auch an b weitergesendet. Da zu dem Zeitpunkt der Scout über b zu d schon unterwegs ist, sind bidirektionale Tabelleneinträge für (b, d) vorhanden

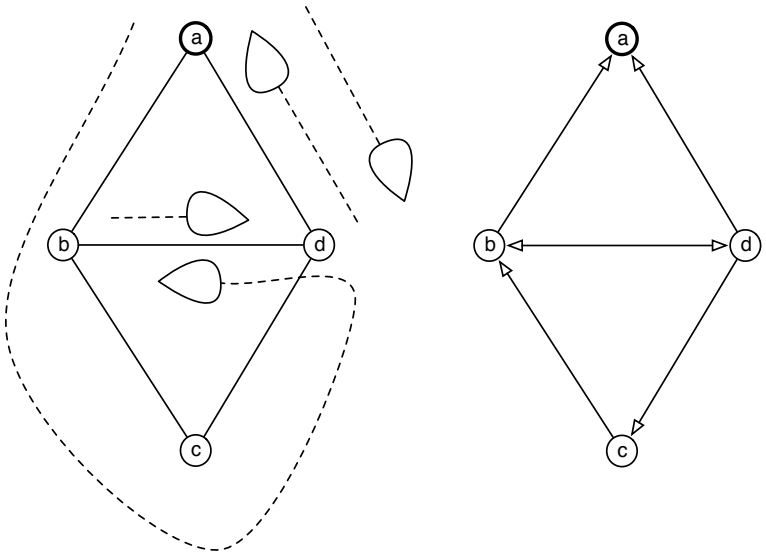


Abbildung 6.19.: Entstehung von Tabellenschleifen

(siehe rechtes Teilbild). Weder an b noch an d ist zu den Empfangszeitpunkten rein mit lokalen Informationen zu entscheiden, ob ein Tabellenkreis vorliegt. Die einfachste Lösung, nur den ersten Eintrag innerhalb einer Generation seitens eines Scoutursprungs zu erlauben, garantiert zwar Kreisfreiheit, eliminiert aber gerade auch alle anderen Alternativpfade. Bei deterministischem min-Forwarding würde ein Fahrzeug trotz Tabellenkreisen über den LCP (und damit kreisfrei) zum Ziel a geführt. Bei stochastischer Selektion wird jedoch ein gewisser Anteil ϕ_s über jeden Nachfolger $s \in \overrightarrow{S}$ gesendet. Die stochastische Selektion kann daher ein Fahrzeug theoretisch, zumindest ohne weitere Vorkehrungen, endlos in dem Kreis $b \to d \to c \to b$ leiten. In Allgemeinen entsteht aus den Tabelleneinträgen demnach statt einem In Tree ein gerichteter Graph mit Kreisen.

Einfachste Vermeidungsstrategie ist das Führen eine Tabu-Liste, besuchte Knoten dürfen kein zweites Mal besucht werden. Im einfachsten Fall (Tabu-Listenlänge von 1) ist es verboten, zum vorherigen Knoten zurückzukehren. So kann bei genügend langer Tabu-Liste vermieden werden, dass ein Kreis mehrfach durchlaufen wird, aber nicht, dass er überhaupt auftritt.

Verkehrsbedingt: Angenommen, das Fahrzeug befindet sich zum Zeitpunkt t_0 an Knoten d und sei weiter angenommen, das Fahrzeug würde bei gleichbleibenden Linkkosten über den LCP $d \to b \to a$ zum Ziel a geleitet. Ändern sich die Kosten während das Fahrzeug sich auf dem Link (d, b) befindet derart, dass der LCP auf $b \to d \to a$ wechselt, dann würde das Fahrzeug an b zu d umkehren. Dann wird der Knoten d ein zweites Mal passiert und es bildet sich der Kreis $d \to b \to d$. (Nimmt man die Regel hinzu, dass Fahrzeuge nicht zum letzten Knoten zurück dürfen, muss nur der Graph etwas anders gestaltet werden; das Problem bleibt erhalten, denn mittels einer Liste mit n Einträgen können nur Kreise der Länge $n + 1$ erkannt werden). In der Realität kann das eine richtige Entscheidung sein, denn herrscht Stau, lohnt es in gewissen Fällen eventuell wirklich umzudrehen. Kommt das Fahrzeug an Knoten d an und hätte sich die Auslastung im Netz wieder zu Ungunsten von b gedreht, dann könnte theoretisch ein

ewiges Pendeln zwischen d und b auftreten. Dazu müsste sich aber wirklich jedesmal die Netzsituation entsprechend ändern. Erstens treten solche Phänomene aber vermutlich nicht auf. Zweitens würde ein Fahrer solch ein Hin- und Herfahren auch nicht beliebig oft akzeptieren und sich irgendwann final für einen Weg entscheiden.

Der erste Fall sollte, wie bereits zuvor erwähnt, beim einem VRGS unbedingt vermieden werden, ist es einem Fahrer doch nicht zuzumuten, unnötigerweise Kreise zu fahren. Der zweite Fall hingegen ist prinzipiell erwünscht, sind doch potentiell Fahrzeitreduzierungen möglich. Allerdings könnte in seltenen (und konstruierten) Situationen dieser zweite Fall zu einem Endloskreis führen, oder zumindest zu einem mehrfachen Durchfahren ein- und desselben Kreises. In den durchgeführten Simulationen konnte so etwas jedoch nie beobachtet werden.

Der nächste Abschnitt zeigt Möglichkeiten zur Unterbindung von Forwarding-bedingten Kreisen.

6.9.3. Unterbindung von Kreisen

Es seien vier Lösungsmöglichkeiten diskutiert, das MAS so zu erweitern, dass Forwarding-bedingte Kreise trotz stochastischer Selektion und generationsübergreifenden Einträgen vermieden werden:

Ad-Hoc-Erkennung: Am geeignetsten wäre, direkt wenn ein Scout einen Knoten erreicht, entscheiden zu können, ob der Scout einen Eintrag in die Tabellen vornehmen darf. Dazu müsste aber entschieden werden können, ob durch den neuen Eintrag ein Tabellenkreis ermöglicht wird. Es lassen sich leicht Beispiele angeben, in denen aber lokal nicht über diese Eigenschaft entschieden werden kann, da es auch von anderen Agenten (mit demselben Ursprungsknoten und der selben Generation) abhängt (wie z.B. bei der besprochenen Situation in Abbildung 6.19). Zwar könnte mittels eines modifizierten Pfadvektorprotokolls, welches auch Informationen darüber verteilt, an welchen Knoten welche Scouts ankamen, in nachfolgenden Generation erkannt werden, das ein Kreis geschlossen wurde. Dazu bedarf es aber zusätzlicher Programmlogik und die Nachrichtenlänge würde erheblich anwachsen. Zudem wären an Knoten b und d in Abbildung 6.19 jeweils nur ein Pfad zum Ziel bekannt. Angenommen, beide Pfade weisen in etwa gleiche Kosten auf, dann müssten alle Fahrzeuge über einen Pfad fahren, statt dass auch der zweite Pfad zu ca. 50% genutzt würde.

A-Priori-Vermeidung: Die Idee hierbei ist, dass Fahrzeuge nur so geroutet werden, dass trotz existenter Tabellenkreise niemals die Gefahr eines Kreise besteht. Dazu wird eine Ordnung aller Knoten im Netz bzgl. des Ziels d implementiert. Für die Ordnung soll gelten, dass für einen Scoutpfad $(o, v_1, \ldots, v_n)$ gilt: $o < v_1 < \ldots < v_n$. Welche Ordnung gewählt wird, ist (zumindest) für die Kreisfreiheit irrelevant. In Frage kommen beispielsweise die beiden statischen Relationen „Entfernung in Hops“ oder „Entfernung in Metern“, oder die dynamische Relation „CCP-Fahrzeit“. Ferner wird das Forwarding so angepasst, dass das Fahrzeug nur zu Nachfolgern kleinerer Ordnung geschickt werden. Da die Ordnung von Knoten zu Knoten streng monoton abnimmt, muss ein Fahrzeug an v_i (mit Ziel o) in solch einer endlichen Kette in endlichen Schritten an o ankommen. Kreise können nicht auftreten, da Knoten höherer Ordnung nicht als Next Hop in Frage kommen. Jedoch werden auch hierdurch viele potentielle Pfade von Beginn an ausgeschlossen. Werden die Knoten bspw. nach „Entfernung in Metern“ geordnet, dann kann ein Fahrzeug

niemals zu einem Knoten weitergeleitet werden, der weiter entfernt liegt (im Sinne einer geographischen Distanz). Das Fahrzeug kann demnach keinen (im ursprünglichen Wortsinne) *Umweg* fahren. Die deterministische min-Selektion des Basisprotokolls ist als ein Spezialfall dieser Vermeidungsstrategie zu sehen, bei der nur ein einziger Knoten niedrigerer Ordnung (bzgl. der Fahrzeit) in Frage kommt. Würde sich ein Fahrzeug bspw. für den Link (b, d) in Abbildung 6.19 entscheiden und würde als Ordnung die Hop Distanz zu a gewählt, dann kann das Fahrzeug nicht mehr zu b zurück, da die Hop Distanz nicht sinken würde. Angenommen, das Fahrzeug befände sich an c und von c gelänge man auch über (c, d) zu d (im Gegensatz zu der Darstellung in der Abbildung). Dann könnte zufällig zwischen b und d entschieden werden, da beide Knoten eine geringere Hop Distanz zu a aufweisen als c selbst (tie break).

Posteriori-Vermeidung: Es wird ein neuer Agententyp eingeführt, der nur nach möglichen Kreisen sucht und diese „zerstört“, d.h. einige Einträge aus den Tabellen löscht, so dass Fahrzeuge nicht mehr im Kreis geroutet werden können. Diese Lösung hätte aber vier Nachteile. Erstens müsste dazu das Agentensystem erheblich erweitert werden, da ein komplett neuer Agententyp einführt würde. Zweitens wäre das wiederum mit einem Flooding verbunden, wodurch die Nachrichtenkomplexität erheblich anstiege. Drittens wäre ein instabiler Zustand möglich, denn Kreise wären erst dann nicht mehr möglich, wenn diese „Aufräum-Agenten“ das komplette Netz „gesäubert“ hätten. Bis dahin sind Kreise potentiell weiterhin möglich. Viertens werden wieder potentielle Pfade gelöscht, die zwar zu Kreisen führen können, aber nicht müssen.

Fahrzeug-spezifische Posteriori-Vermeidung: Bei diesem Konzept werden nicht die Tabelleneinträge gelöscht, die *im Allgemeinen* zu einem Kreis führen könnten. Stattdessen werden nur *spezifische Einträge für ein einzelnes Fahrzeug zeitweise deaktiviert*, d.h. für die Next Hop-Selektion nicht berücksichtigt werden. So wird verhindert, dass ein Fahrzeug Next Hops wählen kann, die schlussendlich zu einem Kreis führen könnten. Allerdings muss auch hierfür ein zusätzlicher Agententyp eingeführt werden. Dieses Vorgehen stellt eine Fahrzeug-bezogene Variante des Route Poisoning-Mechanismus für RIP dar (vgl. Abschnitt 5.5.2).

Im Rahmen dieser Arbeit wurde die A-Priori- und die Fahrzeug-spezifische Posteriori-Vermeidung umgesetzt.

Bei der umgesetzten *A-Priori-Vermeidung* darf ein Scout nur dann Tabelleneinträge vornehmen, wenn durch diese Änderung Einträge entstünden, die einem Fahrzeug einen Vorteil böten, d.h. wenn das Folgen eines Links, die restliche Fahrzeit zum Ziel verringert. Genau dann kann kein Forwarding-bedingter Kreis entstehen, da in einem Kreis die Restfahrzeit anstiege. Nachteilig bei dieser Vorgehensweise ist, dass dann nur wenige Knoten mehr als einen Eintrag in Richtung des Ziel aufweisen. Im Falle der später zu Evaluationszwecken verwendeten Netze sind es ca. 7%. Eine Verbesserung der Fahrzeiten lies sich damit nicht erzielen.

Die *Fahrzeug-spezifische Posteriori-Vermeidung* lässt zunächst Tabellenkreise durch die Scout-Dissemination zu, verhindert aber bei anschließenden Next Hop-Selektionen das Auftreten von Forwarding-Kreisen, indem Informationen einer zusätzlichen Agentenart inkorporiert werden. Dazu floodet ein Knoten von dem ein Fahrzeug v aufgebrochen ist, umgehend einen sogenannten *Depletion-Forager* zur Deaktivierung einzelner Tabelleneinträge. Neben seinem Ursprung o kennt dieser Forager auch das Ziel d des Fahrzeugs α. Da der Depletion-Forager sich

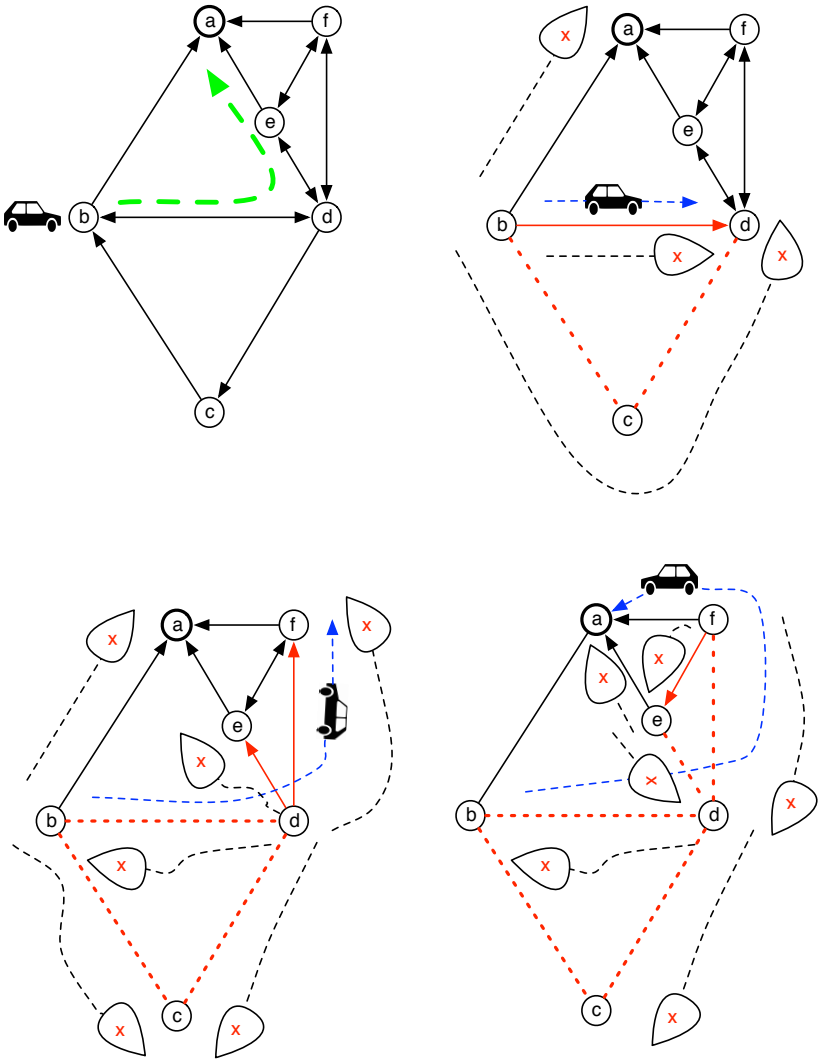


Abbildung 6.20.: Kreisvermeidung mittels Fahrzeug-spezifischen Link-„Deaktivierungen“

erheblich schneller bewegt als das zugehörige Fahrzeug (zumindest i.d.R. trotz Asynchronität), können so Kreise vermieden werden.

Das Vorgehen:

1. α verlässt o, weswegen o Depletion-Forager an alle Vorgänger $p \in P_o$ floodet.

2. Erreicht ein Depletion-Forager einen Knoten i über $s \in \overrightarrow{S}_i^d$, wird der Tabelleneintrag T_i^{sd} für α deaktiviert. Ein Request von α an i ignoriert fortan diesen Eintrag. Dadurch wird verhindert, dass α über i wieder zu s gelangt, wodurch ein Kreis geschlossen würde. Andere Fahrzeuge $\alpha' \neq \alpha$ können den Link (i, s) hingegen uneingeschränkt nutzen (im Gegensatz zu den anderen oben genannten Verfahren).

3. Der Forager fliegt nur weiter, wenn maximal ein anderer aktiver Eintrag $T_i^{s'd}, s' \in \overrightarrow{S}_i^d$ existiert. Sind mehr aktive Einträge vorhanden (es existieren daher noch Möglichkeiten von i zu d ohne einen Kreis zu schließen), oder ist d erreicht, oder der zu deaktivierende

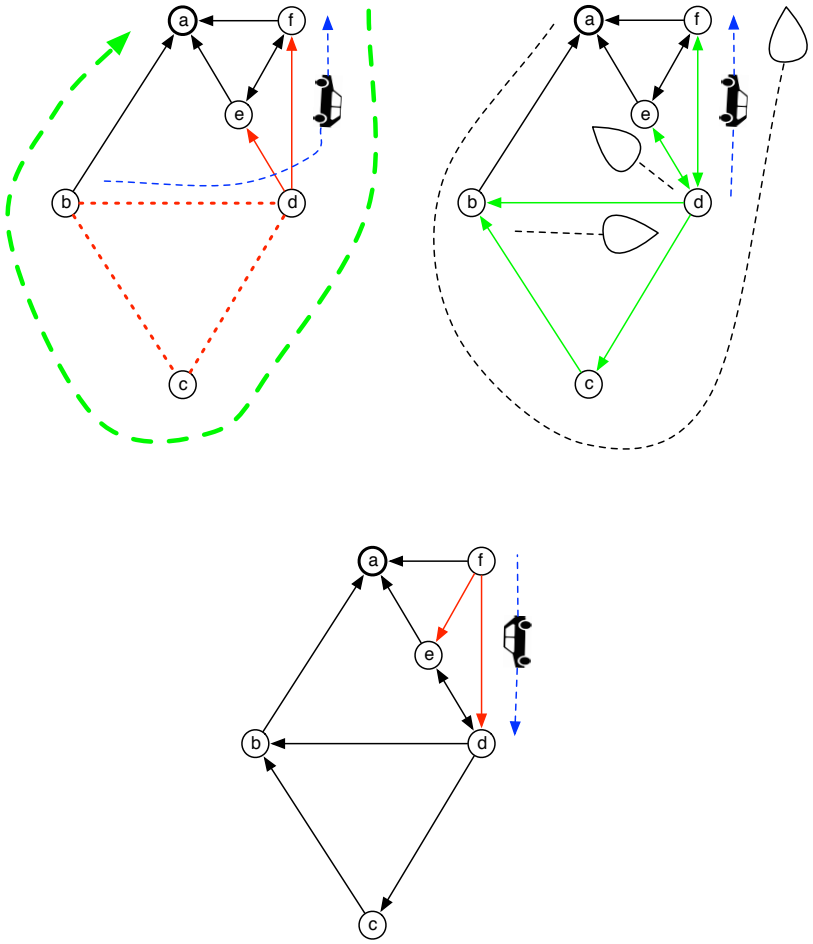
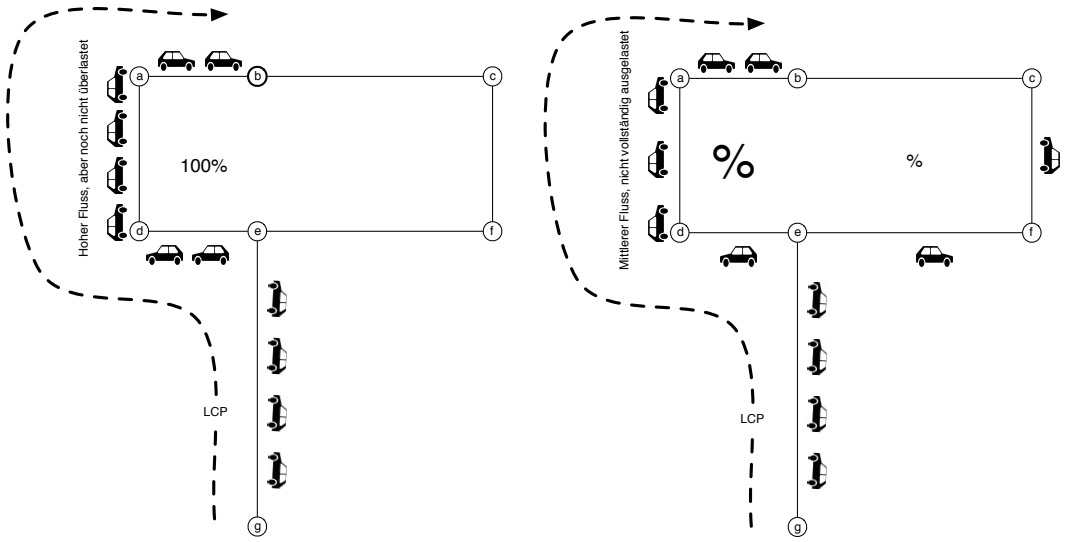


Abbildung 6.21.: Auflösung von Fahrzeug-spezifischen Link-„Deaktivierungen“

Link bereits deaktiviert, oder bereits ein Depletion-Forager mit Ursprung o wegen α weitergeleitet, wird der Agent getötet.

Abbildung 6.20 illustriert dieses Konzept. Das dargestellte Fahrzeug im ersten Teilbild befindet sich an Knoten b mit Ziel a. Der LCP sei durch die grüne Markierung angedeutet. In dem Beispiel nutzt das Fahrzeug den Link (b, d). Gleichzeit werden die mit einem „x“ gekennzeichneten Depletion-Forager gefloodet. Sie deaktivieren die Links $(d, b), (c, b)$ und (d, c) für das dargestellte Fahrzeug. Die beiden unteren Links kann das Fahrzeug gar nicht mehr nutzen, sie sind daher gestrichelt dargestellt. Der mittlere Link ist nur noch unidirektional zu befahren, so dass von d keine Möglichkeit mehr besteht, zu b zurückzukehren und damit einen Kreis zu schließen. Die Depletion-Forager werden an d und a nicht weiter geleitet, da mehrere Tabelleneinträge zum Ziel existieren bzw. das Ziel selbst erreicht wurde (siehe zweites Teilbild). Das Fahrzeug setzt seinen Weg über (d, f) und schließlich (f, a) fort und die Depletion-Forager deaktivieren die umliegenden Links entsprechend. Im dritten Teilbild wird dabei ein Depletion-Forager noch einmal über den Link (b, a) gesendet, da zuvor kein Depletion-Forager von d den Knoten b erreicht hatte. Die Redundanz könnte durch eine Pfadvektorerweiterung der Depletion-Forager (Mitführen und Speichern von Pfaden an Knoten) erkannt werden, ist aber nicht weiter verfolgt worden. Im vierten Teilbild muss der an b ankommende Depletion-Forager



(a) Deterministische min-Selektion (b) Stochastische ϕ-Selektion

Abbildung 6.22.: Effekte des stochastischen Forwardings

allerdings nicht mehr weitergeleitet werden, da der Link (d, b) bereits zuvor deaktiviert wurde. Die Depletion-Forager verbreiten sich demnach nur in einer lokalen Umgebung, abhängig von den vorherigen Depletion-Foragern, die aufgrund von v emittiert wurden.

Angenommen, wie im ersten Teilbild von Abbildung 6.21 angedeutet, der LCP ändert sich während α auf Link (d, f) ist. Es besteht keine Möglichkeit, zu d zurückzukehren und auf dem neuem LCP $d \to c \to b \to a$ das Ziel zu erreichen. Es wird daher ein Mechanismus benötigt, die Deaktivierungen abhängig von der aktuellen Netzsituation wieder aufzuheben. Dazu werden die üblichen Upstream-Scouts verwendet. Sobald ein Scout ein Knoten erreicht, aktiviert er dort deaktivierte Links, sobald er über einen günstigeren Pfad eintraf und demnach den Tabelleneintrag senken kann. Dadurch werden die Links auf dem LCP wieder befahrbar, wie im zweiten Teilbild dargestellt. Im dritten Teilbild ist ersichtlich, dass α an f in Richtung d umkehrte, und das die Links (f, e) und (f, d) durch von f startende (nicht eingezeichnete) Depletion-Forager (nun in entgegengesetzter Richtung) deaktiviert wurden.

6.9.4. Auswirkungen und Diskussion

Die Abbildung 6.22 verdeutlicht die Unterschiede des Basisprotokolls mit deterministischer Auswahl und dem stochastischen Forwarding. In Abbildung 6.22a ist verdeutlicht, dass alle Fahrzeuge über den LCP zu b geschickt werden. Es herrscht zwar ein hoher Fluss auf dem linken Pfad, jedoch sind die Kosten immer noch günstiger als auf dem rechten Pfad. Solange das so bleibt, wird kein Fahrzeuge über den rechten Pfad geleitet. In Abbildung 6.22a ist die gleiche Situation mit stochastischem Forwarding dargestellt. Die meisten Fahrzeuge werden ob der geringeren Kosten über den linken Pfad geleitet. Ein kleinerer Teil jedoch über den Rechten. Anders ausgedrückt: ein Teil der Fahrzeuge wird vollkommen ohne Not über einen Pfad geleitet, der längere Zeit zum Ziel benötigt, statt auf einen noch nicht ausgelasteten Pfad mit geringeren Kosten zurückzugreifen. Zumal dieses Umleiten auch zu Überlasten an anderer Stelle führen kann. Wie zu erwarten, führte dieser Ansatz in Simulationen zu keinen

Verbesserungen, stattdessen sogar zu Verschlechterungen (im Vergleich zum Basisprotokoll).

Eine wesentliche Einschränkung der stochastischen Selektion ist, dass sie rein auf den Ex Ante-Kosten basiert und bei der vorgestellten Pfadbewertung B_1 die Fahrzeugdichte auf den Pfaden vollkommen außer Acht lässt. Selbst wenn auf dem linken Pfad eine sehr geringe Dichte herrschte, würden proportional zu den Kosten Fahrzeuge auf den rechten Pfad umgeleitet, was noch weniger sinnvoll wäre. Denn bei einer hohen Dichte auf dem linken Pfad könnte zumindest argumentiert werden, dass mit einer gewissen Wahrscheinlichkeit sich dort in naher Zukunft durch zusätzlichen Verkehr (bspw. über einen zusätzlichen Link zu d aus anderen hier nicht eingezeichneten Teilen des Netzes), die Pfadkosten zu b sich aufgrund eines Staus drastisch erhöhen könnten. B_2, B_3 und B_4 berücksichtigen das, aber auch dies führte zu keinerlei Verbesserungen in Simulationen gegenüber des Basisprotokolls. Allgemein ist festzuhalten, dass die Flussumschichtung in unkontrollierter Weise geschieht. Wie hoch die erfahrenen Fahrtkosten ohne Umschichtung ausgefallen wären, wird nicht berücksichtigt.

Vorteile des stochastischen Ansatzes konnten zusammenfassend somit nicht festgestellt werden. Bleibt zu fragen, weshalb in Computernetzen eine Verbesserung zu beobachten ist. Für BeeHive (berücksichtigt nicht die Auslastung, sondern nur die Kosten nach B_1) existiert bspw. keine explizite Untersuchung, die das Protokoll mit deterministischer und stochastischer Selektion vergleicht. Stattdessen wird durchgehend nur die stochastische Selektion verwendet. Es ist daher schwer zu sagen, ob dadurch überhaupt ein Unterschied existiert wie behauptet. Indizien deuten aber darauf hin, dass es auch dabei keine signifikanten Auswirkungen hat. So wird in [45, S. 277] festgehalten, dass eine BeeHive-Variante, in der Pakete nur zu Nachbarn mit geringeren Kosten weitergeleitet wurden, im Vergleich keine negativen Ergebnisse zeitigte. Viele Knoten haben aber nur wenige Nachbarn mit geringeren Kosten. Wie oben bereits erwähnt, hat das zu Evaluationszwecken verwendete Straßennetz an nur 7% der Knoten mehr als einen Nachfolgereintrag mit geringeren Kosten zu einem Ziel. Für Computernetze ist nicht unmittelbar ersichtlich, weswegen das anders sein sollte. Daher wird bei dieser BeeHive-Variante vermutlich in den meisten Fällen ebenfalls deterministisch gemäß min-Selektion weitergeleitet, ohne das dies negative Auswirkungen hatte. Ggf. liegt hier ein Fall vor, in dem Occam's Razor gilt: die simple, deterministische Variante liefert (mindestens) gleich gute Ergebnisse wie die komplexere, stochastische.

Es sei für eine abschließende Bewertung noch einmal das Verhalten der Bienen betrachtet. Ein stochastische Auswahl der nächsten Futterquelle durch die Sammlerinnen erscheint in dem natürlichen Vorbild nämlich erklärbar. Denn eine Rückmeldung, dass an einer Quelle sich gegenwärtig zu viele Sammlerinnen tummeln, benötigt aufgrund des langen Fluges eine gewisse Zeit, bis sie im Stock verbreitet wurde. In der Zwischenzeit könnten viele neue Sammlerinnen rekrutiert werden, die die Überlastung ggf. noch verstärken. D.h. das System reagiert recht *träge* auf eine Veränderung der Umwelt. In Computernetzen mag die Situation eine ähnliche sein. Angenommen, ein Protokoll weist eine sekündliche Aktualisierungsfrequenz auf (wie BeeHive). In der Zeitspanne zwischen zwei Aktualisierungen können sehr viele Pakete entstehen und weitergeleitet werden. Das System ist in diesem Sinne ebenfalls träge, da, bis ein Hot Spot (überlasteter Knoten) im Netz bekannt gemacht wurde, relativ betrachtet, eine große Zeitspanne vergehen kann. Eine stochastische Selektion mag eventuell helfen, den relativ langen Zeitraum ohne Aktualisierungen zu überbrücken. Durch die kostenabhängige ϕ-Selektion werden die Pakete auf die zu verfügungstehenden alternativen Pfade aufgeteilt, statt alle über einen Pfad gesendet zu werden. BeeJamA hingegen weist relativ betrachtet eine *hohe Reaktionszeit* auf, da ebenfalls sekündlich Aktualisierungen angestoßen werden, die Fahrzeuge aber um Größenordnungen langsamer sind als Datenpakete. Entsteht daher an

einer Stelle eine Überlast, wird dies unmittelbar bekannt gemacht und trotz deterministischer Selektion wird daraus eine Umleitung, um gestaute Bereiche resultieren – jedoch nur, wenn wirklich die Kosten geringer wären (im Gegensatz zur ϕ-Selektion). In Computernetzen würden die Pakete bei deterministischer Wahl den entstehenden Hot Spot hingegen nicht umgehen, sondern noch verschärfen. Daher mag eine stochastische Selektion dort sinnvoll sein. Da aber nur die Kosten in dieser Selektion berücksichtigt werden (und nicht zusätzlich die Auslastung), können die umgeleiteten Pakete anderswo eine Verschärfung bedeuten. Wenn nach der zuletzt genannten Quelle tatsächlich keine Vorteile durch stochastisches Routing belegbar waren, könnte die Interpretation dessen sein, dass die Vor- und Nachteile sich gegenseitig eliminieren und es daher in Summe zu keiner Verbesserung kommt.

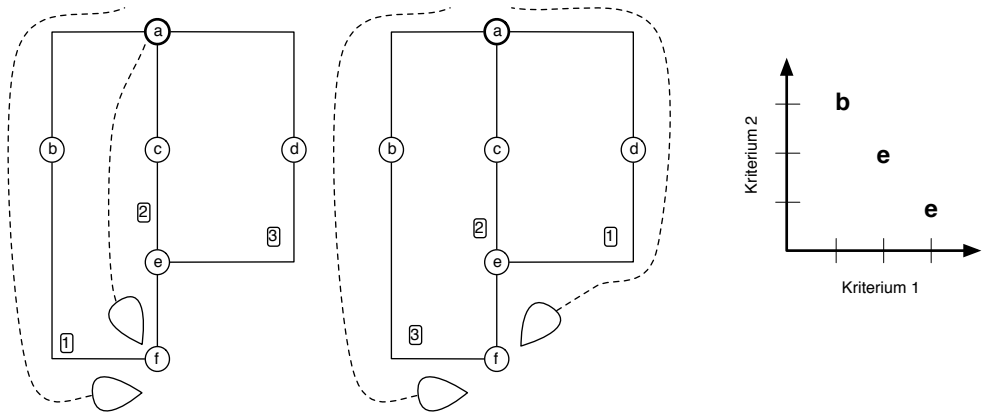
Es bedarf weiterer Untersuchungen, um diesen Aspekt genauer zu eruieren.

6.10. Mehrkriterielles Routing

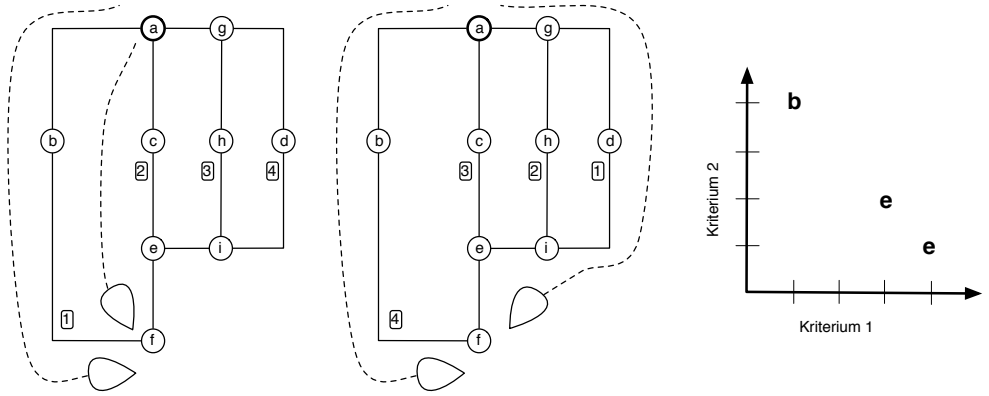
Bisher wurde als Routingmetrik ein einzelnes Kriterium, die aggregierte Transitzeit der Links des Pfades, betrachtet. Mitunter ist es jedoch von Belang, mehrere Kriterien in die Pfadfindung mit einzubeziehen. Als ein typisches Beispiel kann der Energieverbrauch dienen. Eventuell nimmt ein Fahrer Umwege zu Gunsten einer gewissen Kraftstoffersparnis in Kauf (indem auf bspw. auf eine kraftstoffsparende, längere Autobahnstrecke ausgewichen wird, statt kürzere kraftstoffintensive, innerstädtische Routen zu wählen). Für verteilte VRGS ist solche Mehrkriterialität bisher nicht untersucht, stattdessen lag allein die Fahrzeit im Fokus. In Computernetzen hingegen wurde dieser Aspekt bereits bearbeitet [87, 92, 134], bpsw. in Ad-Hoc-Funknetzen (wie Wireless Sensor Networks) bei denen ebenfalls der Energieverbrauch eine Rolle spielt. Eine typische Vorgehensweise ist die Link-Aggregierungsoperation $\ominus$ beim Eintreffen eines Scouts $\mathcal{S}$ über (j, i) und n Kostenriterien $\omega_{ij,1}, \ldots, c_{ij,n}$ wie folgt umzusetzen:

$$C_{\mathcal{S}} \leftarrow C_{\mathcal{S}} + \sum_{k=1}^{n} \alpha_k \omega_{ij,k}^{\beta_k} \tag{6.5}$$

Dabei stellen α_i, β_i subjektive Gewichtungen dar. Dadurch degeneriert das mehrkriterielle in ein einkriterielles Problem und das geschilderte MAS kann ohne weitere Änderungen eingesetzt werden. Zwar sind die Gewichtungen subjektiv, aber nicht im Sinne individueller Fahrer. Die Scouts des BeeJamA-Protokolls verbreiten Linkkosten ohne Vorwissen, welche Fahrzeuge diese Informationen später verwenden. Stattdessen werden die Daten in Tabellen abgelegt und können von da an, bei Forwarding-Entscheidungen für beliebige Fahrzeuge, verwendet werden. Im Umkehrschluss bedeutet dies, dass der Scout bei der Link-Aggregierung keine Fahrer-individuellen Gewichte berücksichtigen kann. Stattdessen kann nur ein globales (für alle identisches) Gewicht für das jeweilige Kriterium berücksichtigt werden. Ob ein Fahrer mehr Umweg in Kauf nimmt für eine bestimmte Kraftstoffersparnis als ein anderer, lässt sich so nicht modellieren. Das ist ein inhärenter Nachteil dieses Ansatzes, der sich zumindest abschwächen ließe, in dem „gängige“ Gewichtungen *a priori* bestimmt und festgelegt werden. Zu jeder Gewichtung wird dann ein separater Scout entsendet. Bei m Gewichtungen, steigt die Nachrichtenkomplexität um den Faktor m, genau wie die Anzahl der zur Speicherung benötigten Tabellen. Im Extremfall übermittelt jedes Fahrzeug die gewünschte, individuelle Gewichtung an den Zielknoten (bzw. den repräsentativen Knoten, der gegenwärtig angesteuert wird), so dass dort startende Scouts diese berücksichtigen können.



(a) Prinzip



(b) Unvollständige Exploration

Abbildung 6.23.: Mehrkriterielle Dissemination

Abgesehen von diesen Nachteilen, rein technisch müsste am verwendeten Protokoll, außer der Anpassung der $\ominus$-Operation, keine weiteren Änderungen am Disseminationsverhalten vorgenommen werden. Solche Fahrer-individuellen Gewichtungen werden in den folgenden Simulationen aber nicht weiter betrachtet, dafür allerdings globale Gewichtungen.

Allgemein handelt es sich um ein mehrkriterielles Problem [30] mit einer *pareto-optimalen Lösungsmenge* bestehend aus nicht dominierten Lösungen, d.h. aus Pfaden, deren mehrkriterielle Kostenvektor von keinem anderen Pfad bezüglich aller Kriterien unterlegen ist. Im Allgemeinen kann das Pareto-Optimum Elemente enthalten, die durch keine Gewichtsfunktion des zuvor genannten, einfachen einkriteriellen Vorgehens erzeugt werden kann.

Das Pareto-Optimum in einem verteilten System unter Realzeitbedinungen zu bestimmen, ist nicht leicht. Zumal es sich um eine dynamische Umgebung handelt und das (Pareto-)Optimum somit nicht statisch ist. Um die Schwierigkeiten zu verdeutlichen, sei ein einfaches Verfahren geschildert: Pro Kriterium wird ein Scouttyp gefloodet, welcher gemäß dieses Kriteriums jeweils mittels des Basisprotokolls (jedoch ohne das Löschen von nicht-optimalen Kosten) weitergeleitet wird (pro Kriterium darf ein Scout pro Knoten pro Generation weitergeleitet werden). Jeder Scout aggregiert wie üblich die Kosten zu diesem Kriterium, aber auch zu allen übrigen Kriterien. Im konvergierten Zustand kennt dann jeder Knoten zumindest den LCP

bzgl. jedes Kriteriums zu den Scoutursprungsknoten. Jedoch muss nicht jeder Knoten alle Pfadalternativen kennen und somit auch keine vollständige Pareto-Front, wie das folgende Beispiel zeigt.

Abbildung 6.23a, in der Knoten a das Ziel und f den aktuellen Knoten darstellt, verdeutlicht das Problem. Im ersten Teilbild sind die Kosten für das erste von zwei Kriterien dargestellt. Demnach ist der Pfad $f \to b \to a$ am günstigsten (markiert mit einer umrandeten 1), $f \to e \to c \to a$ am zweit günstigsten und $f \to d \to a$ am schlechtesten. Im konvergierten Zustand empfängt Knoten f Scouts über $a \to b \to f$ und $a \to c \to e \to f$, aber nicht über $a \to d \to e \to f$, da an Knoten e stets der Scout über c weitergeleitet wird (da er stets der Scout mit den geringsten Kosten aus der letzten Generation ist, vgl. Best-Propagierung). Analog verhält es sich für das zweite Kriterium im zweiten Teilbild, wo kein Scout über $a \to c \to e \to f$ Knoten f erreicht. Im Ergebnis hat f Kenntnis über alle Kostenkombinationen, da über jeden Link ein Scout f erreicht. In dritten Teilbild sind die an f bekannten Next Hops bzgl. der beiden Kriterien dargestellt, ähnlich einer Pareto-Front. So entstehen über b bzgl. des ersten Kriteriums die niedrigsten Kosten und bzgl. des zweiten Kriteriums die höchsten. Dem Fahrer kann entsprechend seiner Präferenzen vom Navigator ein Next Hop mitgeteilt werden.

In dem in Abbildung 6.23b dargestellten Graph ist mittels dieses Verfahrens an f jedoch nicht mehr jede Möglichkeit bekannt. Denn weder bzgl. des ersten noch bzgl. des zweiten Kriteriums erreicht ein Scout f über h, weswegen der Pfad mit den drittbesten Kosten bzgl. des Kriteriums 1 und den zweitbesten Kosten bzgl. Kriterium 1 unberücksichtigt bleibt. Unter Umständen wird das Fahrzeug folglich über b weitergeleitet, obwohl den Präferenzen des Fahrers nach auch eine Weiterleitung über $e \to i \to h \to g \to a$ möglich gewesen wäre.

Ursächlich für diesen Umstand ist, dass pro Knoten nur der beste Scout (der letzten Generation) weitergeleitet wird. Dadurch wird die Anzahl der Nachrichten zwar linear beschränkt, jedoch werden Alternativpfade (mit suboptimalen Kosten) ausgeblendet. Auch eine ϕ-Aggregierung löst im Allgemeinen nicht das Problem, da die über verschiedene Pfade aggregierten Kosten nicht eindeutig sind. Im Zweifelsfall würde so immer noch über b geroutet, statt über e.

Mit maximal $|V|$ Nachrichten ist es in diesem Beispiel demnach nicht realisierbar alle Möglichkeiten aufzudecken. Es scheint, als ob mit linearer Nachrichtenkomplexität das Problem im Allgemeinen nicht zu lösen ist, zumal das (nicht verteilt zu lösende) mehrkriterielle LCP-Problem Instanzen mit exponentieller Zeitkomplexität kennt [137]. Es könnte heuristisch vorgegangen werden, in dem nicht nur der beste Scout weitergeleitet wird und die Kosten generationsübergreifend gespeichert werden. So könnte f irgendwann auch einen Scout über h empfangen. Ggf. müssten zusätzliche die Kreisvermeidungsstrategien aus Abschnitt 6.9 hinzugefügt werden. Solche Verfahren wurden aber im Rahmen dieser Arbeit nicht weiter eruiert.

Im Evaluationsteil wird ein Spezialfall betrachtet, bei dem das geschilderte Problem nicht auftritt: Angenommen, ein Fahrer sei gewillt einen Kompromiss bzgl. der Fahrzeit einzugehen, d.h. einen langsameren Weg zu akzeptieren, wenn er dadurch helfen kann, überlastete Links zu entlasten.

Gesucht ist demnach der *schnellste Pfad* auf dem *noch genügend Platz* (i.S.d. Verkehrsdichte) ist, so dass die zusätzlichen Fahrzeuge sich zu keiner erheblichen Mehrbelastung für andere Fahrer entwickeln. Daraus ergeben sich zwei Kriterien, die sich durch den Einsatz zweier Scouttyps umsetzen lassen. Der erste Scouttyp wird wie üblich mit dem Kriterium der schnellsten Fahrzeit weitergeleitet. Der zweite Scouttyp wird zwar ebenfalls nach diesem Kriterium der Fahrzeit verbreitet, allerdings zusätzlich unter der Maßgabe, dass die Dichte

auf einem Link einen gewissen Wert nicht überschritten darf. Die empirischen Untersuchungen werden zeigen, dass durch die Akzeptanz von Nachteilen durch eine Minderheit, ein Vorteil für die Allgemeinheit entsteht.

Zu dem geschilderten Problem, das relevante Pfad-Kosten-Kombinationen unbekannt sind, kann es dabei nicht kommen, da nur die beiden LCPs für die Forwarding-Entscheidung benötigt werden und keine weiteren Kombinationen.

Die Abbildung 6.24 verdeutlicht das Konzept beispielhaft. Angenommen, Knoten b sei das Ziel und das Fahrzeug befände sich an a. Der LCP bzgl. der Fahrzeit würde über $a \rightarrow b$ führen, der LCP bzgl. „Fahrzeit und Auslastung nicht über einem Schwellwert“ hingegen führt über $a \rightarrow c \rightarrow d \rightarrow e \rightarrow b$. Die Links $(a, b), (c, b)$ und (d, b), dargestellt durch dickere Linien, sollen stark ausgelastet sein (oberhalb des Schwellwertes). Dem Fahrzeug stehen so zwei Pfade zur Verfügung, zwischen denen gewählt werden kann. Dazu übermittelt das Fahrzeug die Präferenzen des Fahrers mit jeder Routinganfrage an den lokalen Navigator, welche Toleranz für eine unausgelastete Route gegenüber des Fahrzeit-LCPs noch akzeptiert wird.

Allerdings muss es auch eine Belohnung für die Akzeptanz dieses Nachteils geben, denn der Vorteil liegt bei der Allgemeinheit: Nimmt der Fahrer einen Umweg in Kauf und verlegt seinen Weg zum Ziel auf weniger ausgelastete Links, so verringert sich die Verkehrsdichte auf den stärker frequentierten Links – zum Vorteil der Allgemeinheit. Außer für altruistisch orientierte Fahrer wäre dies in der Realität kein Ansporn, eine Fahrzeitverschlechterung in Kauf zu nehmen, weswegen ein Ausgleich geschaffen werden müsste. Die Zeit die der Fahrer der Allgemeinheit spart (dadurch, dass andere Fahrer über weniger belastete Straßen zum Ziel kommen) könnte finanziell ausgeglichen werden, oder aber dazu verwendet werden stark ausgelastete Straßen nutzen zu dürfen, ohne dafür einen Maut-Aufschlag zahlen zu müssen (siehe dazu die Ausführungen zur Randkostenbepreisung, vgl. Abschnitt 4.2.2).

Es soll nicht unerwähnt bleiben, dass in diesem Falle von Ausgleichszahlungen ein Abrechnungsmechanismus notwendig ist. Das würde schlussendlich personenbezogene Daten erfordern, wodurch die Anonymität leiden würde (vgl. Abschnitt 6.1).

Wenngleich dieser Abschnitt das Problem des mehrkriteriellen, verteilten Routings nicht allumfänglich lösen kann, zeigt es doch die Schwierigkeiten auf. Ziel in weiterführenden Arbeiten muss es sein, mit einer vertretbaren Anzahl von Nachrichten zumindest die wichtigsten Pfadalternativen bzgl. gegebener Kriterien zu explorieren. Der zuvor genannte Spezialfall wird im folgenden Kapitel simulativ erprobt und führt dabei, unter gewissen Umständen, zu Vorteilen für die „Allgemeinheit“.

6.11. Pfadreservierungen

In diesem Abschnitt wird mit dem Konzept der Pfadreservierungen, ein über die Navigatoren vermittelter Kooperationsmechanismus zwischen Fahrzeugen zur Absprache von Pfadselektionen diskutiert und eine Umsetzung zur Erweiterung des BeeJamA-Basisprotokolls vorgestellt. Statt wie bisher rein reaktiv Pfadkosten zu verbreiten, wird das Protokoll dadurch in die Lage versetzt, proaktiv zukünftige Überlasten einzubeziehen. So werden statt CCP-Kosten antizipierte dynamisch-konsekutive Predicted Cost Pricing-Kosten (PCP-Kosten) durch die Scouts verbreitet. Statt reaktiv wie das Basisprotokoll, verhält sich diese Variante *proaktiv*: Fahrzeuge werden um erst zukünftig entstehende Staus herumgeleitet.

Dabei beschreibt die *Pfadreservierung* den Vorgang, eine zukünftige, intendierte Verwendung eines Pfades anzumelden, um so andere Fahrzeuge (vermöge der Navigatoren) über die

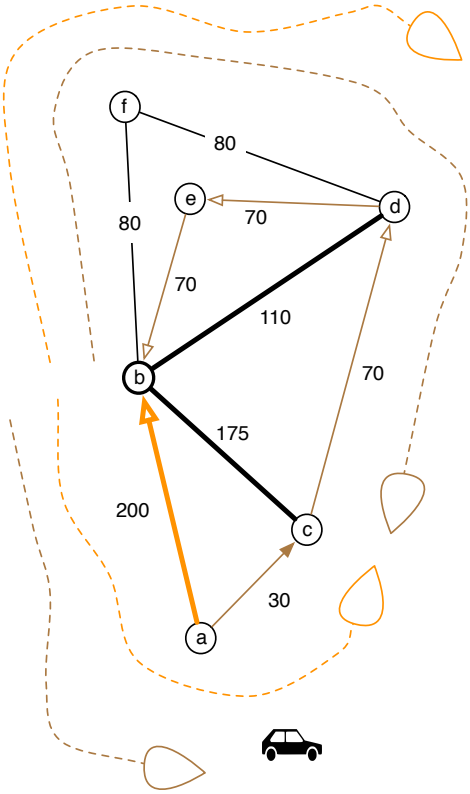


Abbildung 6.24.: Beispiel zur bikriteriellen Entscheidung zwischen Auslastung und Fahrzeit

beabsichtigte Route zu informieren. Dazu muss den Links eines Pfades bekannt gemacht werden, dass in einem bestimmten, zukünftigen Zeitintervall ein Fahrzeug die beschränkten Kapazitäten des jeweiligen Links beanspruchen möchte. So können nachfolgende Forwarding-Entscheidungen diese Informationen inkorporieren und schließlich Fahrzeuge, so die grundlegende Idee, um zukünftig entstehende Staus auf einem ursprünglich, beabsichtigten Pfad frühzeitig herum geleitet werden. Man erwäge hierzu folgendes Beispiel in Abbildung 6.25, in dem Knoten a das Ziel sei. Im ersten Teilbild befinden sich die beiden an d abgebogenen Fahrzeuge auf dem Link (d, c) und damit auf dem LCP zu a. Angenommen, das dritte Fahrzeug würde d erreichen bevor die beiden zuvor genannten Fahrzeuge den Link (c, d) erreichen. Dann sei aufgrund einer angenommenen genügend hohen Kapazität von (d, c) der LCP weiterhin der linke Pfad, weswegen das dritte Fahrzeug ebenfalls über den linken Pfad geleitet würde (siehe das zweite Teilbild).

Nun weise aber in diesem (artifiziellen) Beispiel der Link (c, b) eine so geringe Kapazität auf, dass sobald die beiden ersten Fahrzeuge diesen Link erreichen, die Transitzeit extrem ansteigt, so dass die Fahrzeuge sehr lange auf diesem Link verharren. Dadurch verlaufe der LCP nun über den rechten Pfad. Dann wäre es retrospektiv sinnvoller gewesen, das dritte Fahrzeug an d über den rechten statt über den linken Pfad zu leiten. Jedoch berücksichtigt das Basisprotokoll ausschließlich CCP-Linkkosten und vermeidet Vorhersagen absichtlich. Da aber zum Zeitpunkt der Forwarding-Entscheidung für das dritte Fahrzeug an d, die Linkkosten von (c, d) noch gleich der FFT ist, ist der linke Pfad im Vergleich zum rechten noch in Vorteil. Würde das Protokoll hingegen antizipieren, dass die beiden vorausfahrenden Fahrzeuge definitiv den

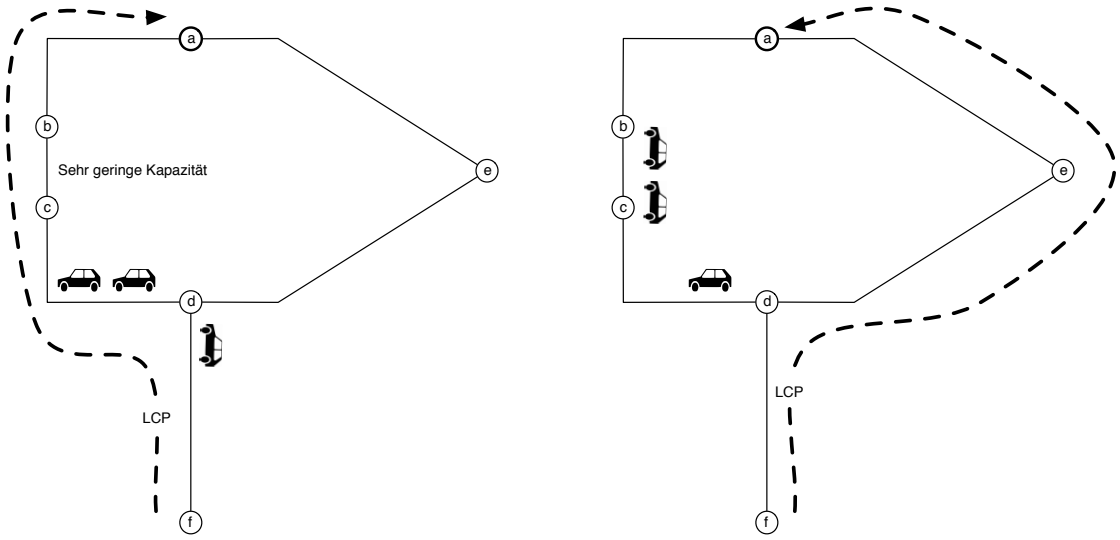


Abbildung 6.25.: Problem bei Forwarding bzgl. aktueller Pfadkosten

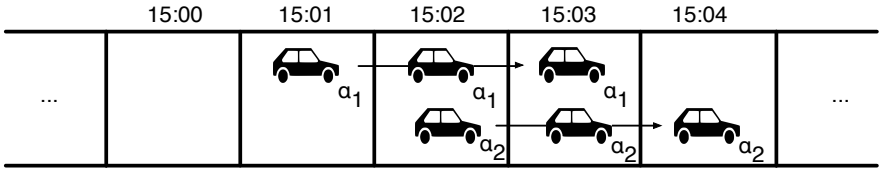


Abbildung 6.26.: Reservierungslog eines Links mit Eintragungen zweier Fahrzeuge

Link (c, b) passieren werden und das sich dort dadurch ein Stau bildet, der solange anhält, dass auch das dritte Fahrzeug dadurch in Mitleidenschaft gezogen würde, hätte an d eine andere Entscheidung getroffen werden können. Dazu sind aber PCP-Kosten notwendig, die vom Basisprotokoll nicht verbreitet werden. Solche Situationen können im Kleinen auftreten, wie in diesem Beispiel. Oder im Großen, z.B. wenn Stau auf einer Autobahn herrscht und zu viele Fahrzeuge auf ein und dieselbe Umleitung ausweichen.

An diesem Punkt setzt das Konzept der Pfadreservierung an, indem Fahrzeuge ihre intendierte, zukünftige Nutzungsabsicht durch Link-Anmeldungen ihrer Umgebung mitteilen. Aufbauend auf diesen Anmeldungen können Forwarding-Entscheidungen getroffen werden, die zukünftige Linkauslastungen antizipieren.

Es wird demnach eine Lösung benötigt, um aus Link-Anmeldungen eine Transitzeit schlusszufolgern. Das Werkzeug zum Ausdruck dieses Zusammenhangs zwischen erwarteten Fahrzeugen und Transitzeit sind empirische Dichte-Geschwindigkeits-Diagramme (vgl. Abschnitt 3.3.1), auch Link Performance Functions (LPF) genannt. Der Zusammenhang zwischen (makroskopischer) Geschwindigkeit und Dichte kann mit geeigneten Techniken maschinell gelernt werden ([150], [29], [106]) oder wie für die Simulationen dieser Arbeit durch die in Abschnitt 3.3.3 beschriebe MATSim-LPF (vgl. Definition 3.5) ausgedrückt werden.

Um Pfadreservierungen zu speichern, wird eine zusätzliche Datenstruktur, das *Reservierungslog*, pro Link benötigt. Dieses vom Navigator verwaltete Log speichert die angekündigten Aufenthalte der Fahrzeuge. In Abbildung 6.26 ist ein Log dargestellt, in welchem zwei Fahrzeuge für jeweils drei Minuten eingetragen sind. Es besteht aus *Slots* einer bestimmten Länge (z.B.

1 Minute) und pro Slot können beliebig viele Fahrzeuge anhand ihrer ID vermerkt werden. Der grundsätzliche Ablauf des Konzepts ist dann wie folgt:

Angenommen Fahrzeug α mit Ziel v_n befindet sich gegenwärtig zum Zeitpunkt t_0 an Knoten v_0 und ein Pfad $(v_0, v_1, \ldots, v_n)$ zum Ziel wurde bestimmt (wie dies geschieht, wird weiter unten diskutiert). Anhand der LPF wird zunächst die erwartete Transitzeit $t_{0,1}$ über den ersten Link (v_0, v_1) bestimmt und das Fahrzeug im Reservierungslog für das Intervall $[t_0, t_{0,1}]$ *registriert*, d.h. die ID α des Fahrzeugs wird in die zugehörigen Slots geschrieben. Dieser Prozess wird anschließend über den gesamten Pfad iteriert. Nachfolgende Pfadberechnungen anderer Fahrzeuge verwenden dann nicht mehr die aktuellen Fahrzeiten, sondern die per LPF bestimmten zukünftigen. Sobald ein Fahrzeug eine Online-Pfadneuberechnung durchführt, müssen vorherige Pfadreservierungen *aktualisiert* werden, d.h. auf den Links des alten Pfades müssen diese je nachdem verschoben oder sogar gelöscht werden.

Da das Basisprotokoll nur den Next Hop selektiert, statt einen ganzen Pfad zum Ziel, muss das Basisprotokoll u.a. in dieser Hinsicht angepasst werden. Der nächste Abschnitt zeigt auf, wie das geschehen kann.

6.11.1. Integration in das BeeJamA-Protokoll

Mit dem bisher beschrieben Agentenmodell lassen sich Pfadreservierungen nicht umsetzen. Denn zum Einen ist ein wesentliches Merkmal gerade, dass keine kompletten Pfade im Voraus berechnet werden. Zum Anderen, angenommen es würden vollständige Pfade berechnet, dann existiert noch kein Mechanismus die Reservierung auch durchzuführen. Denn bisher handelt es sich bei den Agenten um Upstream-Scouts. Die Reservierungen müssen aber in Downstream-Richtung vorgenommen werden, da die Fahrzeugankunftszeiten, und somit die zugehörigen Slots, durch die Transitzeiten in Downstream-Richtung determiniert sind. Insgesamt sind drei Änderungen am ursprünglichen Agentenmodell vonnöten:

1. **Änderung der $\ominus$-Operation:** Nicht mehr die gegenwärtigen CCP-Transitzeiten werden aggregiert, sondern genau die zukünftigen konsekutiven PCP-Transitzeiten, die ein Fahrzeug erführe, wenn es in Downstream-Richtung weiterfährt, eingedenk der intendierten, registrierten Pfadwahlen übriger Fahrer. Die Semantik solch eines (modifizierten) Upstream-Scouts bleibt somit erhalten: Er hat weiterhin die Aufgabe, an einem Knoten n eine Schätzung in die Tabellen einzutragen, wie lange es von n zum Scoutursprung in Downstream-Richtung dauert, wenn ein Fahrzeug zum aktuellen Zeitpunkt von n zum Scouturprung über den Pfad des Upstream-Scouts in Downstream-Richtung losführe.

2. **Einführung von Downstream-Scouts:** Da ein Upstream-Scout im bisherigen Modell nicht wissen kann, wie lange gegenwärtig Fahrzeuge von den noch zu explorierenden Knoten bis zum aktuellen Knoten des Upstream-Scouts benötigen, kann er auch nicht entscheiden, welche Einträge aus dem Log zu aggregieren sind. Zusätzliche Downstream-Scouts werden diese Informationen liefern und dabei ein identisches Disseminationsverhalten verwenden wie die Upstream-Scouts. Jedoch werden bei der Passage eines Links (i, j) die Linkkosten ω_{ij} aggregiert statt ω_{ji} (vgl. hierzu Abschnitt 6.4). An den Zielknoten ist dann dadurch somit auch bekannt, wie lange Fahrzeuge von einem bestimten Knoten benötigen, um sie zu erreichen. Zuvor war nur die gegenteilige Information vorhanden: wie lange benötigt man von dem aktuellen Knoten zu einem Zielknoten.

3. **Erweiterung der Forager-Agenten:** Zur Durchführung der Reservierung wird das

Forager-Konzept erweitert. Bisher bewegen sich die rein virtuellen Forager mit identischer Geschwindigkeit wie ihr assoziiertes Fahrzeug. Nun wird ein zusätzlicher, nicht mehr nur metaphorischer Forager-Agent eingesetzt. Dieser wird über das Netz versendet, um unmittelbar Reservierungen auf den Links des aktuellen Pfades zum Ziel gemäß der aktuellen Forwarding-Entscheidungen vorzunehmen.

Zunächst zur Erweiterung der Upstream-Scouts. Statt die aktuellen Linkkosten vom Navigator abzufragen, muss vermöge der LPF und den Einträgen des Reservierungslogs die zukünftige Transitzeit bestimmt werden.

Diese erste Modifikation ist leicht umsetzbar. Sei η die verwendete LPF. Statt $C_{\mathcal{S}} \leftarrow C_{\mathcal{S}} + w_{is}$ wird die Link-Aggregierungsoperation nach dem Empfangen des Scouts an i zunächst wie folgt umgesetzt:

$$C_{\mathcal{S}} \leftarrow C_{\mathcal{S}} + \eta_{is}(Log_{is}(t)), \tag{6.6}$$

mit $Log_{is}(t)$ als Anzahl der reservierten Fahrzeuge an Link (i, s) zum Zeitpunkt t. Schlechterdings kann der Scout aber bisher mit lokalem Wissen nicht entscheiden für welches t das Log ausgewertet werden soll. Denn es ist unbekannt, wann Fahrzeuge aus Upstream-Richtung den Knoten i erreichen.

Als Beispiel betrachte man Abbildung 6.27a. Sei e das Ziel des Fahrzeugs und a die aktuelle Position. Knoten e floodet wie gewohnt Upstream-Scouts. Der Upstream-Scout auf dem rechten Pfad, weiß bisher nicht, bspw. sobald d passiert wird, wann ein Fahrzeug von a den Knoten d erreichen würde. Welcher Log Slot ausgewertet werden soll, ist somit nicht bekannt.

Eine erste Idee zur Lösung könnte sein, diese Informationen von entgegenkommenden Upstream-Scouts hinterlegen zu lassen. So richtig diese Herangehensweise auch ist, löst es das Problem in dem Falle nur unidirektionaler Links bzw. Pfaden nicht. Aufgrund der Richtungen in dem Beispiel würde nie ein Upstream-Scout ausgehend von a aus gefloodet. Agenten, welche die Ankunftszeiten verbreiten, müssen daher die gleichen Pfade laufen, wie sie später Fahrzeuge führen und zwar auch in der richten Reihenfolge. Daher muss ein neuer Scout-Typ eingeführt werden, der *Downstream-Scout*. Der Downstream-Scout $\mathcal{D}$ traversiert die Links, wie der Name andeutet, in Fahrtrichtung, wird also stets an alle Nachfolger $s \in S_i \setminus \{j\}$ propagiert. Für die Link-Aggregierungsoperation wird dabei konsekutiv die LPF für genau den Zeitpunkt der Ankunft des Fahrzeuges vom Scoutursprung auf diesem Link addiert:

$$C_{\mathcal{D}} \leftarrow \eta_{is}(Log_{is}(C_{\mathcal{D}})). \tag{6.7}$$

Der Downstream-Scout wird ähnlich implementiert wie ein Upstream-Scout, mit einem Unterschied: Die Link-Aggregierung wird vor dem Versenden von i zu $s \in S_i$ durchgeführt, nicht nach dem Empfangen wie bei Upstream-Scouts. Dies ist notwenig, da die Kosten ω_{is}, genau wie die Logeinträge Log_{is} nur an i vorhanden sind.

An jedem Knoten i hinterlässt der Downstream-Scout das Tripel aus potentieller Ankunftszeit der Fahrzeuge, dem (Downstream-) Scoutursprung o, sowie die Distanz in Hops zu diesem als Eintrag in einer *ETA-Tabelle* (Estimated Arrival Time-Tabelle) $\mathfrak{E}_i^o$.

Dadurch kann der (modifizierte) Upstream-Scout auch entscheiden, ob dieser (Downstream-) Scoutursprung mit dem restlichen Hop Limit noch zu erreichen ist und ggf. die LPF bzgl. der Ankunftszeit auszuwerten und mitzuführen ist.

Ein Upstream-Scout muss demnach nicht mehr nur eine einzige Kostenvariable mitführen, sondern je eine für jeden zukünftig noch erreichbaren Knoten (statt der Link-Aggregierungsoperation in Gleichung 6.6):

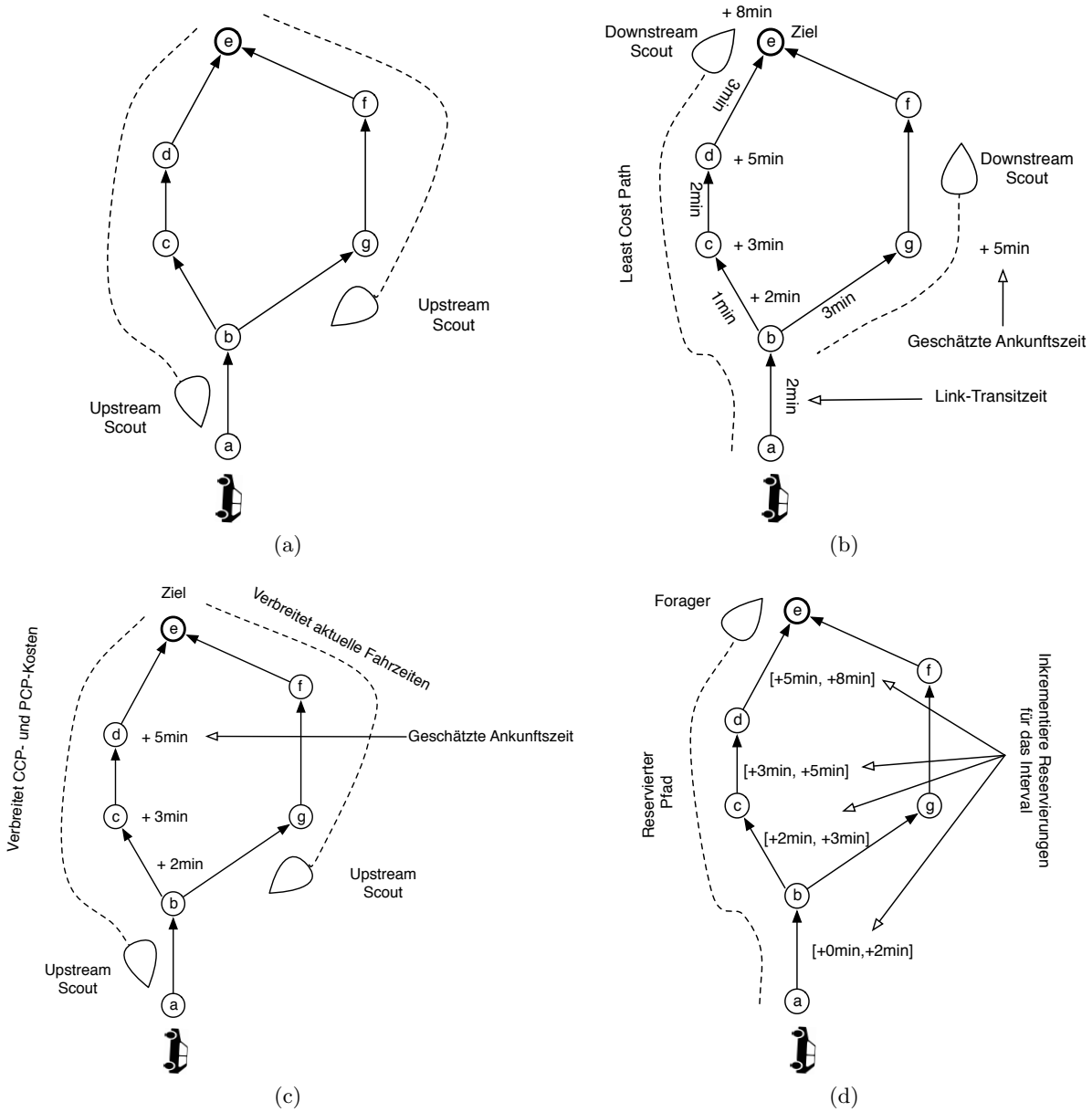


Abbildung 6.27.: Integration des Reservierungskonzeptes

$$C^o_{\mathcal{S}} \leftarrow C^o_{\mathcal{S}} + \eta_{is}(Log_{is}(\mathfrak{E}^o_i)) \qquad \forall o \in \mathfrak{E}_i \tag{6.8}$$

So aggregiert ein Scout konsekutive ELCP-Kosten nach Gleichung 1.6 unter Berücksichtigung der Routenwahl der übrigen Verkehrsteilnehmer.

In Abbildung 6.27b sind Downstream-Scouts ausgehend von a dargestellt, wodurch d bspw. erfährt, dass Fahrzeuge, die aktuell an a zu d aufbrechen in fünf Minuten einträfen. Der Upstream-Scout auf dem linken Pfad in Abbildung 6.27c kann somit an d die LPF mit der Anzahl der erwarteten Fahrzeuge auf dem Link (d, e) in fünf Minuten evaluieren und diese PCP-Kosten aggregieren – statt der aktuellen CCP-Linkkosten. Knoten a wird dadurch erfahren, wie teuer der linke Pfad abhängig von der Ankunft von Fahrzeugen von a auf den jeweiligen Pfadlinks ist – statt nur den aktuellen Pfadkosten.

Schließlich muss noch die *Registrierung* im Log selbst durchgeführt werden. Dazu wird das Verhalten des Foragers abgeändert, genauer gesagt ein zweiter, zusätzlicher Forager hinzugefügt. Dieser *Reservation Forager* begleitet das Fahrzeug nicht mehr mit identischer Geschwindigkeit, sondern bewegt sich mit Agentengeschwindigkeit. An jedem Knoten wird der Reservation Forager $\mathcal{F}$ gemäß $\oplus$-Selektion weitergeleitet, als ob es sich um ein Fahrzeug handelte. Die Kostenvariable wird dabei genau wie bei dem Downstream-Scout nach Gleichung 6.7 aggregiert. Vor jeder Linkpassage (i, s) wird das Log entsprechend der Dauer einer potentiellen Fahrzeugpassage aktualisiert:

$$\forall t \in [\check{t}, \hat{t}] : Log_{is}(t) \leftarrow Log_{is}(t) + 1, \tag{6.9}$$

mit $\check{t} = t_{now} + C_{\mathcal{F}}$, $\hat{t} = \check{t} + \eta_{is}(Log_{is}(C_{\mathcal{F}}))$ und t_{now} als aktuelle Zeit.

Ist der Zielknoten erreicht, wird das Fahrzeug mittels direkter Nachricht (im Gegensatz zum teuren Flooding) über den Pfad des Reservation Foragers informiert. Treten hierbei Änderungen gegenüber des vorherigen Pfades auf, müssen die Navigatoren der betroffenen Links mittels direkten Nachrichten darüber informiert werden, dass Änderungen (Löschen oder Verschieben von Einträgen) an den Logs vorgenommen werden müssen. In Abbildung 6.27d wird das Prinzip illustriert.

Insgesamt entsteht ein vollständig verteilter, asynchroner und dynamischer Pfadreservierungsmechanismus. Zwar wird die Nachrichtenkomplexität durch die Einführung eines neuen Agententyps (Downstream-Scouts) erhöht, sowie die Nachrichtenlänge der Upstream-Scouts vergrößert, da mehrere akkumulierte Transitzeiten mitgeführt werden müssen. (Im Endeffekt entsteht ein DPP.) Jedoch wird durch die Evaluation gezeigt, dass Reservierung die Fahrzeiten potentiell senken kann.

Dennoch bleibt ein erhebliches Problem bestehen. Was passiert bei einer geringen Durchdringung der reservierenden Fahrzeuge? Das vorstehende Reservierungskonzept geht davon aus, dass die LPF-Evaluation möglichst präzise die zukünftig zu erwartende Transitzeit approximiert, was jedoch nur funktionieren kann, wenn die Logs möglichst exakt sind. Was aber passiert, wenn nur ein Bruchteil der Fahrzeuge an dem System partizipiert? Offensichtlich ist zu erwarten, dass die möglichen Fahrzeiteinsparungen dadurch mindestens erheblich beeinträchtigt werden. Man könnte sogar eine Verschlechterung erwarten, denn Fahrzeuge die am Reservierungssystem teilnehmen, verlassen sich darauf, dass die gemeldeten Transitzeiten möglichst korrekt sind. Sie wählen einen Pfad, weil sie davon ausgehen, dass er frei ist. Ist dies aber nicht der Fall, weil Fahrzeuge den Weg versperren, die sich nicht registrierten, kann

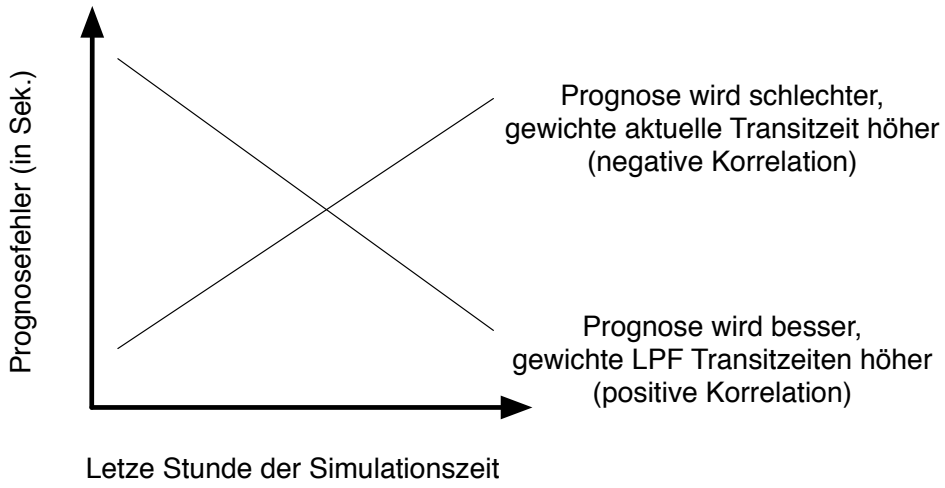


Abbildung 6.28.: Interpretation der Korrelation des Vorhersagefehlers

der Weg über den reservierten Pfad sogar länger dauern als über Alternativpfade, die laut Registrierungssystem eigentlich teurer wären. In Simulationen konnten beide Effekte belegt werden. Der nächste Abschnitt beschreibt daher einen hybriden Ansatz, der die CCP-Kosten ebenfalls berücksichtigt. So werden Approximation mittels Reservierung und aktueller Transitzeit mit unterschiedlicher Gewichtung eingebunden, um bei geringen Durchdringungsraten potentiell dennoch Vorteile aus dem Reservierungskonzept zu ziehen.

Der Ansatz dieses Abschnitts, in dem nur das Log ausgewertet wird, wird in Abgrenzung zu dem hybriden Verfahren in den späteren Evaluationen als *naive Reservierung* bezeichnet.

6.11.2. Hybride Pfadreservierung

Der in diesem Abschnitt vorgestellte Hybridansatz kombiniert die LPF-Vorhersage mit der gegenwärtigen Transitzeit. Hintergrund ist, dass bei geringen Durchdringungsraten der Reservierungsprotokolle, die Logs nur unpräzise die zukünftig zu erwartete Fahrzeuganzahl wiederspiegeln. In der Realität, insbesondere während der Einführungsphase solcher Systeme, ist aber gerade mit einer geringen Durchdringung zu rechnen. Selbst in einem gesättigten Markt müssen nicht zwangsläufig kompatible Systeme Verwendung finden, so dass es zu einer Fragmentierung der Reservierungsprotokolle kommen könnte. Kurzum, vollständige oder zumindest nahezu vollständige Durchdringung ist nicht zu erwarten, so dass Techniken zur Beherrschung dieses Szenarios besondere Bedeutung beikommt.

Die Grundidee dabei ist einfach: Da nicht alle Fahrzeuge eine Link-Registrierung durchführen, sind zukünftig vermutlich mehr Fahrzeuge als angemeldet auf einem Link. So kann vielleicht nicht die zukünftige Transitzeit exakt vorhergesagt werden, aber immerhin ist die aktuelle bekannt (darauf fußt ja gerade das Basisprotokoll). Geht man ferner von einer gewissen Trägheit des Verkehrs aus, so dass die aktuelle Verkehrssituation eine bestimmte Vorhersagekraft für zukünftige Situationen besitzt, dann liegt der Gedanke nahe, diese beiden Transitzeitapproximationen mit einander zu verknüpfen.

Technisch wird dabei die Approximation der zukünftigen Transitzeiten ω_l eines Links l durch eine gewichtete, lineare Kombination der LPF-Prognose ω_l^{lpf} und der aktuellen Transitzeiten

ω_l^{cur} bestimmt:

$$\omega_l = w^{lpf} \cdot \omega_l^{lpf} + w^{cur} \cdot \omega_l^{cur}, \qquad w^{lpf} + w^{cur} = 1 \tag{6.10}$$

Es bleibt die Frage, wie die beiden Gewichte gewählt werden sollen. Hierzu wird wieder ein Bienen-inspiriertes Konzept aufgegriffen: Wird eine zeitliche Veränderung einer Futterquelle seitens der Bienen festgestellt, wird der Tanz entsprechend verstärkt oder abgeschwächt. In der Übertragung lautet die Regel daher: Verbessert (oder verschlechtert) sich die LPF-Prognose über die Zeit, steigt (oder sinkt) das Gewicht w^{lpf} (und w^{cur} komplementär zur Summe 1). Es wird die einfache heuristische Annahme getroffen: *War die LPF-Vorhersagegenauigkeit in der Vergangenheit präzise, dann werden die zukünftigen Prognosen ebenfalls präzise sein.* Zweifelsfrei ist diese Heuristik diskussionswürdig, jedoch zeigt die Evaluation, dass unter Gewissen Umständen, damit Erfolge zu erzielen sind.

Technisch betrachtet muss folglich die Güte der LPF-Prognosen über die Zeit bewertet werden, d.h. der veränderliche Fehler zwischen der vorherigen Prognose und der anschließend wirklich beobachteten Transitzeit. Anders ausgedrückt, es wird eine *Korrelation* zwischen Prognosefehler und zeitlichen Verlauf gesucht. Um dies vergleichbar zu machen, bietet sich der Pearson'sche Korrelationskoeffizient zum Vergleich zweier Zufallsvariablen an. Dieser Koeffizient aus dem Intervall $[-1, +1]$, nimmt das Maximum $+1$ an, wenn ein vollständig positiver linearer Zusammenhang besteht, den Wert 0, wenn kein linearer Zusammenhang besteht und das Minimum -1, wenn ein negativer linearer Zusammenhang besteht. Die Steigung der Regressionsgeraden ist dabei hingegen für den Korrelationskoeffizient irrelvant, solange ein positiver oder negativer linearer Zusammenhang besteht.

Der *Pearson Produkt-Moment Korrelationskoeffizient* für zwei Zufallsvariablen X, Y ist definiert durch

$$\rho(X, Y) = \frac{cov(X, Y)}{\sigma_X \sigma_Y}, \tag{6.11}$$

wobei $cov(X, Y)$ die Kovarianz von X und Y, sowie σ_X, σ_Y die Varianzen von X und Y darstellen.

Im vorliegenden Kontext liegen jedoch Messreihen statt Zufallsvariablen vor. Einerseits sei der *Prognosefehler* $\Delta\epsilon^{l,i}$ für den Link l und den Zeitpunkt t gegeben durch:

$$\Delta\epsilon^{l,t} = \epsilon_{l,t}^{cur} - \epsilon_{l,t}^{lpf}, \tag{6.12}$$

mit $\epsilon_{cur}^{l,t}$ und $\epsilon_{lpf}^{l,t}$ als Messung der tatsächlichen bzw. Prognose der Transitzeit für Link l zum Zeitpunkt t. Andererseits wird die Zeit ebenfalls als diskrete Messreihe der letzten Stunde in Minuten (entsprechend der Slotlänge) betrachtet, $x_i = i, 1 \leq i \leq 60$. Drei Minuten nach Simulationsbeginn liegen bspw. drei Prognosefehler vor, sowie die Messreihe $(x_1 = 1, x_2 = 2, x_3 = 3)$, welche die Zeiten der Erhebung der Fehler ausrückt. Nach einer Stunde liegen 60 Prognosefehler vor und die temporale Messreihe ist $(x_1 = 1, \ldots, x_{60} = 60)$. Da nur die letzte Stunde betrachtet wird, bleibt die Zeit-Messreihe von da an konstant, die Prognosefehler-Messreihe bleibt im Umfang zwar ebenfalls konstant (stets 60 Elemente), jedoch jeweils weiterverschoben, um die letzte Stunde abzubilden. Zu betonen ist, dass die Zeit-Messreihe stets streng monoton steigend ist, $x_i < x_{i+1}$. Steigt der Prognosefehler ebenfalls in allen (maximal) 60 Messungen, liegt ein vollständig positiver linearer Zusammenhang vor und analog ein vollständig negativer linearer Zusammenhang bei sinkendem Prognosefehler.

Da statt Zufallsvariablen pro Link l die empirischen Messreihen $x_1, \ldots, x_n$ und $y_1 =$

$\epsilon^{l,1}, \ldots, y_n = \epsilon^{l,n}, 1 \leq n \leq 60$ verwendet werden, muss der *empirische Korrelationskoeffizient* statt Gleichung 6.11 verwendet werden:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}, \tag{6.13}$$

wobei $\overline{x}, \overline{y}$ die Mittelwerte der Reihen darstellen.

So kann für jeden Link anhand der Zeit-Messreihe und der zugehörigen Prognosefehler-Messreihe individuell für die letzte Stunde entschieden werden, ob eine (lineare) Korrelation zwischen Prognose und tatsächlicher Transitzeit vorlag. Fällt die Korrelation r positiv aus, bedeutet dies, dass der Fehler über die Zeit zunimmt (bzw. dessen lineare Regression). Fällt sie negativ aus, sinkt der Fehler hingegen. Im ersten Falle wird gefolgert, dass die Prognosen nicht zuverlässig sind und die gegenwärtigen Transitzeiten (bei folgenden Berechnungen) ein höheres Gewicht bekommen sollen (und umgekehrt). Dieser intuitive Zusammenhang wird für einen Link l durch folgende Fallunterscheidung umgesetzt:

$$\omega_l = \begin{cases} |r|\omega_l^{lpf} + (1 - |r|)\omega_l^{cur}, & r \in [-1, 0[\\ \omega_l^{lpf}, & r = 0 \\ (1 - |r|)\omega_l^{lpf} + |r|\omega_l^{cur}, & r \in]0, 1] \end{cases} \tag{6.14}$$

Die Gewichte sind dabei komplementär (vgl. Gleichung 6.10). Ist der Prognosefehler gleich Null und damit auch die Korrelation, wird die Prognose zukünftig ausschließlich verwendet. Wurde die Prognose in der letzten Stunde besser, gilt der erste Fall und analog der dritte Fall bei schlechter werdender Prognose. Abbildung 6.28 verdeutlicht diese Idee.

Die Simulationen des nächsten Kapitels werden die Tauglichkeit dieses Ansatzes zeigen.

6.11.3. Randkostenbepreisung

Mittels des Pfadreservierungskonzept lässt sich auch die Randkostenbepreisungsmethodik (Marginal Cost Pricing, MCP) aus Abschnitt 4.2.2 umsetzen. Prinzipielle Idee des MCP ist das Aufschlagen einer Maut abhängig von der zusätzlich induzierten Verzögerung durch das jeweilige Fahrzeug. Für den statischen Fall konnte gezeigt werden, dass das MCP ein UE in ein SO umwandeln kann (und umgekehrt). Auch ist die Vorgehensweise im statischen Fall recht simpel: Es ist bekannt, wie viele Fahrzeuge auf einem Link sind und modellbedingt ist auch bekannt, dass jedes dieser Fahrzeuge gleichsam die Verzögerung erfährt, welches sich aus der Ableitung einer LPF ergibt. Das Produkt aus Verzögerung und Anzahl der Fahrzeuge ergibt, unter diesen stark vereinfachenden Annahmen, den Mautaufschlag zu den gewöhnlichen Linkkosten.

Im dynamischen Fall ist die Situation deutlich komplexer. Denn zum Einen variiert die Anzahl der Fahrzeuge über die Zeit. Zum Anderen ist die Reihenfolge der Fahrzeuge relevant: nur Fahrzeuge die hinter dem jeweiligen Fahrzeug α auf dem Link sind, erfahren eine Verzögerung. Für eine praktische Umsetzung ist daher die Frage zu stellen, wie diese Anzahl bestimmt werden soll. Wirklich exakt lässt sich die Frage nur zu dem Zeitpunkt beantworten, an dem das Fahrzeug α für das der Mautaufschlag bestimmt werden soll, den Link überquert hat. Für eine vorherige Pfadplanung ist das folglich zu spät.

Mittels des Reservierungslogs besteht aber zumindest die Möglichkeit, die Anzahl im Voraus abzuschätzen. Dazu wird für den Zeitpunkt t, für den das Fahrzeug α auf einem

Link l registriert werden soll, genau die Anzahl n von Fahrzeugen bestimmt, die sich bereits für den Link l registriert haben und zwar genau für das Aufenthaltsinterval $[t, \eta_l(t)]$, in welchem sich α voraussichtlich auf l befinden wird (zumindest gemäß LPF-Prognose). Von der allgemeinen Ungenauigkeit des Reservierungsverfahrens (aufgrund der potentiell geringen Durchdringungsrate) abgesehen, ist das zusätzlich eine erheblich geringere Datenlage als im theoretischen Modell der statischen Situation. Berücksichtigt werden können nämlich nur Fahrzeuge die sich bis zum Zeitpunkt t registriert haben. Im theoretischen Modell hingegen ist stets exakt bekannt, wie viele Fahrzeuge einen Link verwenden werden.

Sei $[t', t'']$ das Aufenthaltsintervall des Fahrzeugs α auf dem Link l, dabei wird t' durch die Downstream-Scouts verbreitet und t'' ergibt sich aus der LPF-Prognose. Für ein $t \in [t', t'']$ sei dann $r^{>}(t)$ die Anzahl der Fahrzeuge vor dem Fahrzeug α auf den Link l zum Zeitpunkt t, analog $r^{<}(t)$ die Anzahl der Fahrzeuge hinter v. Der Upstream-Scout aggregiert mittels $\ominus$-Operation die folgenden Linkkosten:

$$\omega_l = \underbrace{\eta_l(Log(l, t'))}_{\text{LPF-Prognose}} + \overbrace{\int_{t'}^{t''} r^{<}(t) \cdot \frac{d\eta(r^{>}(t)+1)}{dx} \cdot \frac{1}{dt}}^{\text{MCP-Aufschlag}} \tag{6.15}$$

Dabei wird für den MCP-Aufschlag für jeden Zeitpunkt $t \in [t', t'']$ die LPF-Ableitung bzgl. der zu diesem Zeitpunkt sich vor dem Fahrzeug v befindlichen Anzahl von Fahrzeugen bestimmt und mit der Anzahl der zu diesem Zeitpunkt hinter v befindlichen Fahrzeuge anteilig multipliziert. Ansonsten muss keinerlei Änderung am MAS vorgenommen werden.

In den Simulationen konnten keine deutlichen Verbesserungen (im Sinne des System Optimums) durch diesen Ansatz festgestellt werden. Die verschlechterte Datenlage im Vergleich zum statischen Fall scheint hier durchzuschlagen.

6.12. Zusammenfassung

Vorstehend wurde das BeeJamA-Protokoll erläutert, wobei streng genommen von einer Art Baukasten zu sprechen ist: Ausgehend von einem Basisprotokoll der Upstream-Scouts, kann durch Austausch von Operationen und Hinzufügen von Agenten mit sehr ähnlichem Verhalten, den Downstream-Scouts, verschiedene Protokollvarianten erzeugt werden. Das Grundkonzept der Dissemination ist dabei stets gleich: Knoten flooden Agenten mit einer gewissen Reichweite, die den Knoten auf ihrem Pfad bekannt machen, wie teuer das Erreichen ihres Ursprungsknotens ist. Ob dabei gegenwärtige oder konsekutive Pfadkosten verbreitet werden, die zukünftig antizipierte Auslastungen berücksichtigen, ist abhängig von der Protokollvariante. Gemein ist allen Varianten, dass, anders als andere Flooding-basierte Ansätze, nur eine lineare Nachrichtenkomplexität entsteht. Erst dadurch und das Konzept der gestuften Hierarchie ergibt sich eine Skalierbarkeit jenseits bekannter Verfahren. *Das BeeJamA-Protokoll ist das erste Protokoll mit diesen Eigenschaften.*

Die Tabelle 6.2 fasst die verschiedenen Varianten zusammen[9]. In der ersten Zeile wird der klassische DVP-Ansatz dargestellt. Es folgt das BeeJamA-Basisprotokoll und deren Varianten. Leere Einträge bedeuten, dass das Verhalten identisch zu dem des Basisprotokolls ist.

[9] In der Darstellung steht η für eine LPF, n für die Anzahl der Knoten in der Reichweite des emittierenden Knotens, CDF für eine Verteilungsfunktion (cummulative distribution function), *rnd* für eine Zufallszahl, $Z_1, \ldots, Z_n$ für Zwischenziele.

6.13. Vergleich der Protokolle

Nachdem zuvor das BeeJamA-Protokoll beschrieben wurde, soll in diesem Abschnitt ein konzeptioneller Vergleich zu den drei, in Abschitt 2 erwähnten, verteilten Verfahren H-ABC, D-MAS und das WSN-basierte Verfahren, gezogen werden. Die Nomenklatur wird zum Zwecke der Vergleichbarkeit, soweit möglich, der dieser Arbeit angepasst.

6.13.1. H-ABC

Das Protokoll [139] orientiert sich an dem natürlichen Vorbild der Ameisen. Das Verfahren unterteilt das Straßennetz zunächst in Sektoren und es wird zwischen inneren Knoten und Randknoten (an den Sektorengrenzen) unterschieden. Eine Stadt soll dabei einem Sektor entsprechen. Innerhalb der Sektoren werden zielgerichtete Downstream-Agenten (genannt: Local Ants) mit Ziel d von einem Knoten s erzeugt, sobald ein Fahrzeug Knoten s mit Ziel d erreicht. Zielgerichtet bedeutet, dass ein Agent Pfade nur zu dem Ziel d explorieren soll. Dazu orientiert sich der Agent an den Erfahrungen vorheriger Agenten, welche in Form von Pheromonspuren an den Links gespeichert werden. An jedem Knoten entscheidet sich der Agent anhand dieser Stigmergenz-Information für einen einzigen, zu explorierenden Nachfolgeknoten. Der Agent kumuliert auf dem Weg die aktuellen Transitzeiten bis das Ziel erreicht wurde. Ist d innerhalb des aktuellen Sektors, wird bei dessen Erreichen ein Upstream-Agent (genannt: Backward Ant) erzeugt, welche auf demselben Pfad, nur in umgekehrter Richtung, des Downstream-Agenten zum Ursprungsknoten zurückläuft und die Routingtabellen an jedem Knoten i für das Ziel d über p entsprechend anpasst, wobei p der Vorgängerknoten auf dem Pfad des Downstream-Agenten ist. Gleichzeitig wird dabei die Pheromonkonzentration der passierten Links proportional zur Transitzeit erhöht. Mit der Zeit tritt eine Evaporation (Reduktion der Pheromonkonzentration auf unaktualisierten Links) ein. Um die Aktualisierungen der Links und Routingtabellen dürchführen zu können, müssen die Agenten dazu (analog zu einem Distanzpfadprotokoll) eine Liste der besuchten Knoten inkl. Transitzeiten mitführen. So können Fahrzeuge entlang der minimalen Einträge in den Routingtabllen zu Zielen innerhalb des aktuellen Sektors geleitet werden. Um Ziele außerhalb des Sektors erreichen zu können, versenden Randknoten Upstream-Agenten (genannt: Exploration Ants) zur Exploration von Pfaden zu anderen Sektoren. Solch eine Exploration Ant wird gesendet, sobald eine Local Ant mit Ziel d außerhalb des aktuellen Sektors auf einen Randknoten trifft. Deren Aufgabe ist die Exploration günstigster Pfade zu einem beliebigen Randknoten des Zielsektors. Dabei werden keine Pfadinformationen wie bei den Local Ants mitgeführt (Distanzpfad), sondern nur kumulierte Fahrzeiten (Distanzvektor). Die Next Hop-Auswahl geschieht zwar wie bei den Local Ants (basierend auf den Erfahrungen vorheriger Exploration Ants), die Tabelleneinträge werden allerdings unmittelbar ohne zusätzliche Backward Ants vorgenommen. Statt allerdings Einträge mit dem Ursprungsknoten als Ziel vorzunehmen, wird ein Zieleintrag für den gesamten Ursprungsvektor vorgenommen. Exploration Ants vereinen damit die Funktionalität der beiden zuvor genannten Agententypen. Supersummativität entsteht dadurch, dass die Tabelleneinträge des Ex-Ante-LCP von mehreren Agenten durch Teilpfade zusammengetragen werden kann. Anders ausgedrückt: das Fahrzeug wird nicht zwangsläufig über einen Pfad geleitet, der auch von einem einzigen Agenten in exakt dieser Reihenfolge traversiert wurde.

Insgesamt können bei diesem Verfahren drei Problemfelder identifiziert werden.

1. Erstens ist der Ansatz nachfragebasiert (on demand). Pfadexplorationen bzgl. eines Ziels finden nur statt, wenn ein Fahrzeug mit diesem Ziel einen Knoten passiert. Liegt das

Ziel innerhalb des Sektors, könnte noch davon ausgegangen werden, dass die Backward Ant schnell genug zurückkehrt bevor das Fahrzeug sich final einordnen muss. Liegt das Ziel jedoch in einem anderen Sektor, so kehrt die Backward Ant umgehend um, sobald die Local Ant einen Randknoten erreicht hat. Die Backward Ant nimmt dabei die letzt bekannten Fahrzeiten von diesem Randknoten zum Zielsektor mit. Wie alt diese Informationen sind, hängt gänzlich davon ab, wann zuletzt Fahrzeuge aus dem Zielsektor Routinganfragen mit Ziel im aktuellen Sektor gestellt haben. Ist dies längere Zeit nicht geschehen, oder sind die Exploration Ants stets an anderen Randknoten des aktuellen Sektors angekommen, dann sind die Informationen entsprechend veraltet. D.h., die Aktualität der Informationen kann nicht von dem anfragenden Fahrzeug beeinflußt werden.

BeeJamA hingegen erzeugt kontinuierlich Scouts zur Pfadexploration. Veraltete Informationen werden so verhindert. Prinzipiell kann das zu Overhead führen, im anschließenden Kapitel wird empirisch gezeigt, dass dies praktisch nicht der Fall ist.

2. Zweitens muss bezweifelt werden, dass das Verfahren hinreichend adaptiv ist. Ursächlich hierfür ist das Verhalten der Agenten, die bessere Pfade in der Umgebung von Pfaden mit hohen Pheromonkonzentrationen suchen (da die Auswahlwahrscheinlichkeit dieser Links höher ist). Verschlechtert sich aber bspw. der aktuell beste Pfad und ein weit davon abseits liegender Pfad verbessert sich stattdessen (absolut oder relativ), dann muss die Exploration dieses neuen besten Pfades nicht zwangsläufig unmittelbar bevor stehen. Es ist durchaus möglich, dass viele Agenten-Generationen benötigt werden, bis eine Konvergenz der neuen Situation entsprechend stattgefunden hat. Da Agenten aber nur nachfrageorientiert erzeugt werden, ist praktisch nur schwer vorhersagbar, wann über den neuen, besseren Pfad umleitet wird. Generell ist langsame Konvergenz in dynamischen Situationen ein Problem in Stigmergie-basierten Ansätzen [132].

   BeeJamA basiert auf einem Flooding-Konzept, d.h. im Sinne einer nebenläufigen Breitensuche werden alle Pfade (in der erlaubten, begrenzten Umgebung) exploriert. BeeJamA identifiziert so auch abseitig gute Pfade zuverlässig und schnell. Ameisen-basierte Ansätze funktionieren eher im Sinne einer probabilistisches Best First Search und nur ein einzelner Pfad (entsprechend der Auswahlwahrscheinlichkeiten) wird pro Agent erkundet.

3. Das Verhalten der Exploration Ants erinnert stark an Downstream-Agenten aus BeeJamA, da in beiden Fällen die Agenten sich vom Ziel wegbewegen und entsprechende Tabelleneinträge vornehmen. Es gibt allerdings einen erheblichen Unterschied. H-ABC konvergiert zum LCP und nur dieser ist als Pfad in Sektor d zum Ursprungssektor s des Agenten bekannt. Alle Fahrzeuge in d mit Zielen in s werden über den gleichen Pfad zwischen den beiden Sektoren geleitet.

   Auch hier bietet das Flooding-Konzept von BeeJamA Vorteile. Es wird nicht nur der beste Pfad zwischen zwei Sektoren erkundet, sondern die LCPs zum Ziel von allen Scout-empfangenden Randknoten der aktuellen Area. Demnach werden die Fahrzeuge nicht alle über denselben Pfad zum Ziel geschickt, sondern über den, der von aktuellen Position der gegenwärtig günstigste ist.

Den wesentlichen Kritikpunkten des Protokolls kann BeeJamA jeweils eine geeignete Gegenmaßnahme entgegenstellen.

6.13.2. WSN-Routing

Das Protokoll [23] betrachtet das Straßennetz als ein Wireless Sensor Network (WSN). Jede Kreuzung entspricht einem Knoten, jeder Link einer Kante. Die Sensoren an den Knoten kennen die Auslastungen der inzidenten Links. Fahrzeuge kommunizieren im Sinne einer V2I-Architektur mit den Sensorknoten. Es handelt es sich ebenfalls um ein nachfrageorientiertes Verfahren. Das Fahrzeug stellt eine Routinganfrage (genannt Interest) mit Ziel d an den direkt vorausliegenden Knoten i des WSN. Dieser Knoten floodet das Interest in einer ersten Phase durch das gesamte Netz, eine Hierarchie besteht nicht. Wurde ein Pfad zum Ziel entdeckt, wird in der zweiten Phase ein sogenanntes Advertisment vom Zielknoten gemäß Bellman-Ford-Ansatz gefloodet[10]. Schlußendlich erreicht über jeden Nachfolger von i ein Advertisment den Ursprungsknoten i. Statt einfach den günstigsten Pfad als Next Hop aus diesen Advertisments auszuwählen, wird ein Fuzzy Multi-Attribute Decision Making-Ansatz verwendet. Welcher Vorteil dieses Vorgehen bieten soll, wird allerdings nicht diskutiert (und ist auch nicht vordergründig ersichtlich). Eine simulative Evaluation findet nur bei vollständiger Durchdringung des Ansatzes statt.

Der Ansatz birgt daneben Skalierungsprobleme. Erstens, da auf Hierarchie vollkommen verzichtet wird. Zweitens, da in beiden Phasen ein Flooding stattfindet und der in der zweiten Phase eingesetzte Bellman-Ford-Algorithmus im Worst Case eine faktorielle Nachrichtenkomplexität aufweist (weswegen die fehlende Hierarchie noch schwerwiegendere Auswirkungen hat).

BeeJamA hingegen bietet lineare Nachrichtenkomplexität und benötigt (im Verwendungssinne des Ansatzes) nur eine „Phase“, denn es werden nur Scouts auf ihrem Rückweg zum Stock im Basisprotokoll modelliert. Ferner wird das potentiell mögliche Auftreten von Kreisen in diesem Verfahren nicht diskutiert. Simulationen finden bei diesem Ansatz nur auf sehr kleinen Netzen statt. Die folgenden Simulationen für BeeJamA werden deutlich umfangreicher ausfallen.

6.13.3. D-MAS

Auch dieser Ansatz ist nachfrageorientiert [163]. Ein Fahrzeug stellt eine Nachfrage an den direkt vorausliegenden Knoten i. Dieser floodet die Anfrage durch das gesamte Netz, eine Hierarchie existiert nicht. Auch ist nicht exakt definiert, was mit „Flooding“ gemeint ist. Zugunsten des Verfahrens sei angenommen, dass damit der Bellman-Ford-Algorithmus gemeint ist (welcher effizienter ist als unkonditionales Flooding, was sonst stattdessen verwendet werden müsste). Der so ermittelte Ex Ante-LCP wird an das Fahrzeug zurückgemeldet. Routingtabellen existieren nicht. Dafür wird ein Reservierungsverfahren verwendet. Längere Zeit unbestätigte Reservierungn verlieren ihre Gültigkeit, da Reservierungen über Pheromonkonzentrationen abgebildet werden, die mit der Zeit evaporieren. Eine Evaluation findet nur bei vollständiger Durchdringung statt.

Zwei Nachteile sind zu nennen. Erstens, es existiert keine Hierarchie und der Bellman-Ford-Algorithmus besitzt eine faktorielle Worst Case Nachrichtenkomplexität. BeeJamA hingegen weist eine lineare Nachrichtenkomplexität auf. Zweitens wird der LCP aufgrund der Reservierungen berechnet. In den empirischen Untersuchungen wird gezeigt, dass das nachteilige

[10] Im Text ist die Rede von „Broadcast“. Vermutlich meinen die Autoren aber Flooding. Prinzipiell wäre der Bellman-Ford-Algorithmus auch mittels eines Broadcasts möglich, jedoch müsste der Algorithmus dann zuvor angepasst werden.

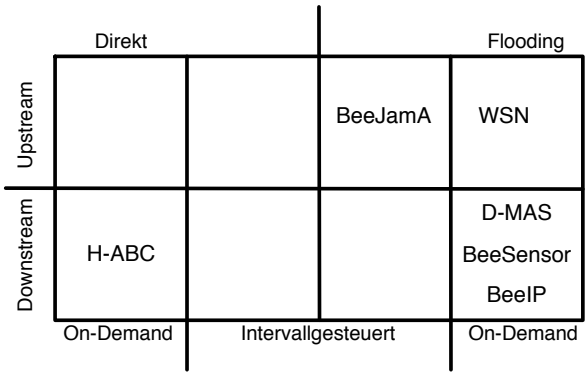


Abbildung 6.29.: Vergleich der Protokolle

Auswirkungen bei unvollständiger Durchdringung hat. BeeJamA ist in der Standardvariante unabhängig von jeglicher Reservierung, bietet mit der hybriden Reservierungsvariante aber die Möglichkeit, neben den Reservierungen auch aktuelle Auslastungen einfließen zu lassen.

6.13.4. Schlussfolgerung

Wie vorstehend aufgezeigt, sind die genannten Verfahren problembehaftet. Die unterschiedlichen BeeJamA-Protokollvarianten gehen diese Problemfelder an. Weiter interessant ist aber die Feststellung, dass H-ABC und D-MAS sich trotz deutlicher Unterschiede beide als Ameisenverfahren bezeichnen. Noch undurchsichtiger wird die Lage, wenn man zusätzlich das BeeIP-Routingprotokoll [54] für Computernetze betrachtet. Das nach eigener Aussage Bienen-inspirierte Verfahren ist dem (Ameisen-basierten) D-MAS-Verfahren nicht unähnlich. Nachfrageorientiert werden bei BeeIP Agenten ausgehend von dem aktuellen Knoten des Datenpakets gefloodet. Hat ein Agent das Ziel erreicht, wird ebenso ein Backward Agent zur Rückmeldung versendet. Reservierungen werden bei BeeIP nicht verwendet (da Paketweise Reservierungen in Computernetzen auch keinen Sinn ergeben). Auch BeeSensor [121] (von demselben Autor wie BeeHive) nutzt Forward und Backward Agenten, wie klassische Ameisenverfahren.

Das WSN-basierte Verfahren verzichtet auf eine Schwarmmetapher, dabei ist dessen zweite Phase aber nicht unähnlich zu einer BeeJamA-Variante ohne jegliche Hierarchie. In beiden Fällen wird vom Ziel ausgehend gefloodet. Demgegenüber basiert das Hierarchiekonzept in BeeJamA auf den Vorarbeiten des BeeHive-Protokolls, welches das Hierarchiekonzept als integralen Bestandteil des Bienenverhalten ansieht und es daraus ableitet. BeeIP verwendet allerdings keine Hierarchie, H-ABC hingegen schon (verzichtet dabei aber auf eine Herleitung aus einem Schwarmverhalten, da Ameisen keine Hierarchie in dieser Form kennen). Genuine, charakteristische Konzepte lassen sich scheinbar nicht unbedingt aus natürlichem Schwarmverhalten ableiten, unterscheiden sich auf dasselbe natürliche Vorbild beziehende Verfahren doch erheblich. Stattdessen lassen sich Konzepte, die viele Schwarmalgorithmen als integralen Bestandteil betrachten, auch vollkommen losgelöst davon entwickeln.

Abbildung 6.29 zeigt eine Einteilung der Verfahren.

Tabelle 6.2.: Übersicht der Protokollvarianten

	$\ominus$	$\circledcirc$	$\oplus$	$\bigcirc\!\!\rightarrow$	$\oslash$	Hop Limit	Nachrichten-komplexität
DVP	$+\omega_{is}$	$C_{\mathcal{S}}$	$\underset{s\in S_i}{\arg\min}\, T_i^{sd}$	$P_i \setminus \{j\}$	Better	∞	$\mathcal{O}(2^n)$
Basisprotokoll			$\underset{\tau\in S_i\times F_I\times E_D}{\arg\min}\quad \delta(\tau)$				
Upstream-Scout	$+\omega_{is}$	$\min T_i^o$		$P_i \setminus \{j\}$	Best	FZ/FR	$\mathcal{O}(n)$
zusätzliche Hierarchie			$\underset{\tau\in S_i\times F_I\times Z_1\times\ldots\times Z_n}{\arg\min}\quad \delta(\tau)$			$l_1,\ldots,l_n$	
stochastisches Routing		$\sum \phi_s \delta(s \rightsquigarrow d)$	$CDF^{-1}(rnd)$				
Mehrkriterielles Routing							
Aggregierung	$\sum \alpha\omega_{k,is}^{\beta}$						
zusätzliche Scouts	$+\omega_{k,is}$						
Reservierung							
Upstream-Scout	$+\eta$	$\min T^o_{i,\mathfrak{E}_d}$					
Downstream-Scout	$+\eta$	$\min T^o_{i,\mathfrak{E}_d}$		$S_i \setminus \{j\}$			
Reservation Forager	$+\eta$			$\oplus T_i^d$	Best	$\mathcal{O}(\gamma)$	$\mathcal{O}(\gamma)$
MCP	$+(\eta+\eta')$						

Kapitel 7

Evaluation

Ziel dieses Kapitels ist die Darstellung der simulativen Evaluationsergebnisse der zuvor beschriebenen Varianten des BeeJamA-Protokolls mittels des GRF und des Verkehrssimulators MATSim.

7.1. Straßennetze und Simulationsaufbau

Für die Evaluation wurden zwei Straßennetze verwendet: jeweils ein Ausschnitt des Ruhrgebiets und der chinesischen Stadt Shanghai. Die Daten stammen aus dem freien OpenStreetMap-Projekt und umfassen pro Netz deutlich mehr als tausend Knoten, was im Vergleich zu anderen Evaluationsstudien von zentralen und dezentralen Routingverfahren sehr groß ist[1]. Beide Netze stellen eine urbane Umgebung dar, welche die wesentliche Anforderung für erfolgreiches Gelingen einer dynamischen Online-Routingstrategie erfüllt: *es existieren stets mehrere Möglichkeiten ein Ziel zu erreichen und die alternativen Pfade weisen nur vertretbar höhere Kosten auf.*

Gäbe es nur wenige Pfade zum Ziel, im Zweifelsfall sogar nur einen, dann wären Bemühungen des Umleitens absehbar nicht von Erfolg gekrönt. Existieren Umleitungsmöglichkeiten, dürfen diese jedoch nicht zusätzlich so exorbitant die Ankunft verzögern, dass ein Beibehalten der aktuellen Route über eine stark verstaute Strecke dennoch vorteilhafter erschiene. Historisch entwickelte und gut erschlossene Straßensysteme weisen aus naheliegenden Gründen jedoch genau diese positiven Eigenschaften auf (denn andernfalls wären sie ein Hemmnis der Stadtentwickelung).

Die Abbildung 7.1 zeigt das verwendete Straßennetz des Ruhrgebiets, reichend von Unna (im Osten) bis Bochum (im Westen) und Dortmund in der Mitte gelegen. Die Abbildung 7.2 zeigt den zugehörigen physikalischen Graphen. Erkennbar sind die vielen alternativen Möglichkeiten, ein Ziel zu erreichen. Zu diesem physikalischen Graphen zeigt die Abbildung 7.3a den zugehörigen, von den Routingprotokollen verwendeten, Routinggraphen. Erkennbar ist, dass die verbliebenen Links geradliniger sind, da die rein formgebenden Knoten (bspw. für Kurven) eliminiert wurden.

Analog findet man die beiden Graphen für den Straßennetzausschnitt von Shanghai in den Abbildungen 7.4a und 7.4b. Abbildung 7.5 zeigt zusätzlich in grün markiert die Links des

[1] So wurde z.B. das H-ABC-Verfahren auf einem Netz bestehend aus 54 Knoten getestet.

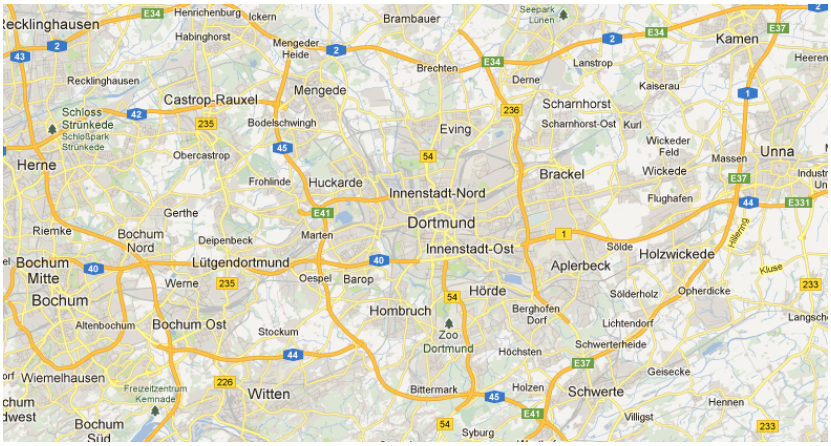


Abbildung 7.1.: Ausschnitt des Ruhrgebiets von Bochum über Dortmund bis Unna

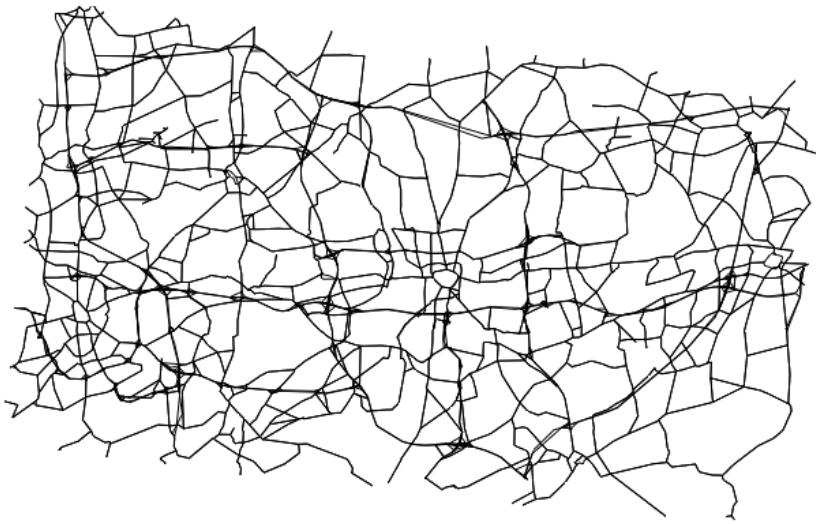
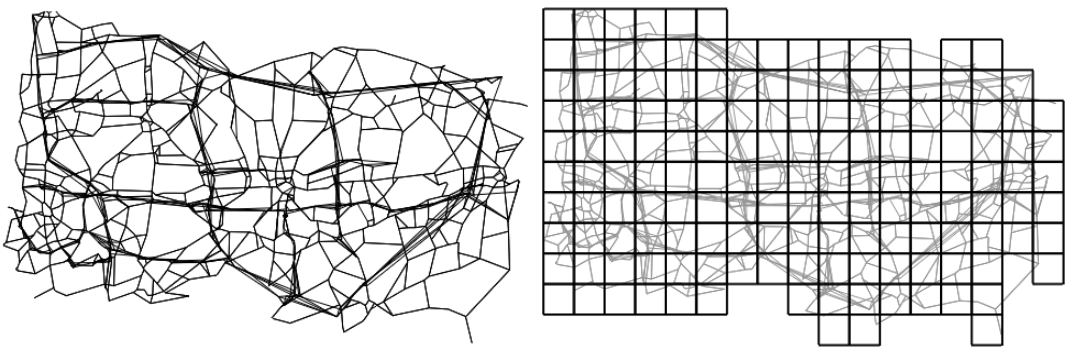


Abbildung 7.2.: Physikalischer Graph des Ruhrgebietausschnitts



(a) Routing Graph (b) Routing Graph mit Area-Einteilung

Abbildung 7.3.: Routing Graph des Ruhrgebietsausschnitts

(a) Physikalischer Graph (b) Routing Graph

Abbildung 7.4.: Shanghai-Netz

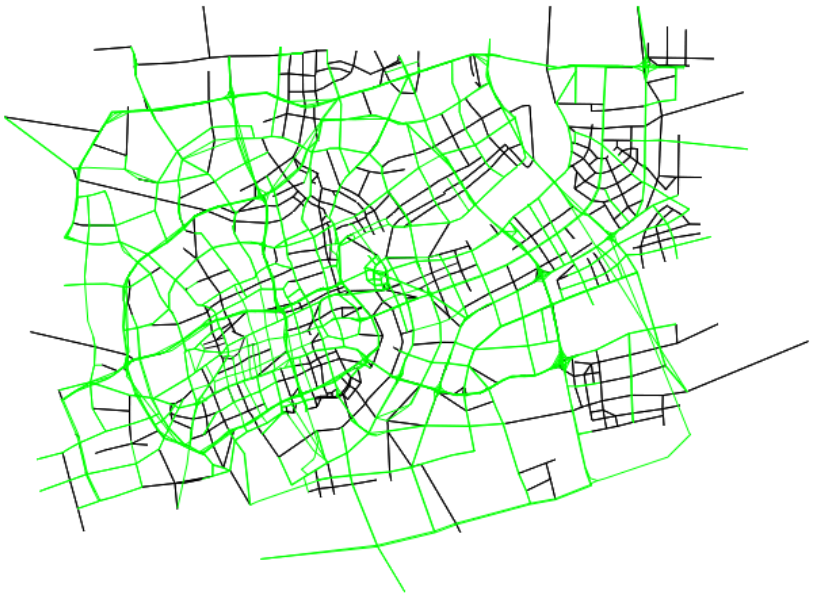
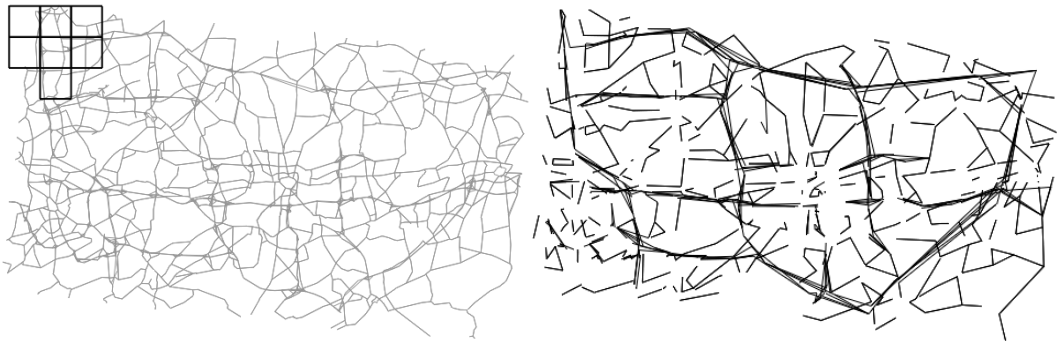


Abbildung 7.5.: Shanghai-Netz mit zur Laufzeit aktualisierten Links (in grün)

Shanghai-Netzes, welche mindestens über eine (Maximal-)Kapazität von 1200 Fahrzeugen / Stunde aufweisen. Idee dabei ist, dass bei einigen der folgenden Untersuchungen die zentralisierenden Protokolle nur Aktualisierungen der Kosten auf genau diesen Links erhalten. Das spiegelt die Situation wieder, dass gegenwärtig kommerziell erhältliche VRGS nur auf einem gewissen Teil der Straßen Kostenaktualisierungen zur Verfügung stellen. Auf den übrigen Links muss die FFT als rein statische Metrik verwendet werden. Eine Maximalkapazität von 1200 Fahrzeugen pro Stunde entspricht nach deutschen Verhältnissen in etwa einer Landstraße oder einer innerstädtischen Straße mit zwei Spuren pro Richtung. Anders ausgedrückt: die nicht markierten Links des Graphen entsprechen kleineren Straßen(-abschnitten). In dem Shanghai-Netz weisen ca. 64% der Links eine größere Kapazität auf, somit erhalten Fahrzeuge für die komplementären 36% keine Aktualisierungen.



(a) Foraging Zone von Area 1 (b) Graph des Net Layers

Abbildung 7.6.: Ruhrgebiet

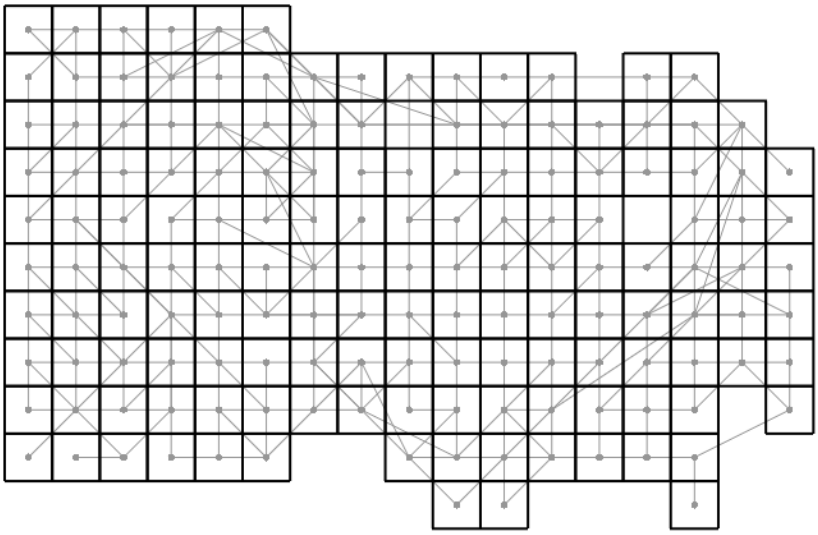


Abbildung 7.7.: Communication Layer des Ruhrgebiets

BeeJamA unterteilt das abgedeckte Gebiet in Areas, wozu an dieser Stelle eine einfache Unterteilung in Grid-Quadrate durchgeführt wird. Abbildung 7.3b zeigt solch eine Unterteilung für das Ruhrgebietsnetz mit einer Kantenlänge von 1,5km pro Quadrat[2]. Abbildung 7.6a illustriert die Foraging Zone der Area 1 bei einem Hop Limit von 2 Areas. Die Abbildung 7.6b zeigt den Net Layer, bestehend aus den Randknoten und den bereichsüberschreitenden Links des Routinggraphen. Abschließend zeigt die Abbildung 7.7 den Communication Layer, der das Kommunikationsnetz der Navigatoren darstellt. (Für das Shanghai-Netz sähen die Abbildungen jeweils ähnlich aus und werden daher nicht dargestellt).

Net und Communication Layer sind rein logische Schichten, die keinem Navigator in Gänze bekannt sind. Die Darstellungen hier dienen ausschließlich der Visualisierung, die Protokolle arbeiten rein auf lokalen Daten, d.h. nur dem Wissen über die direkten Nachbarn einzelner Knoten.

Die Tabelle 7.1 listet einige wichtige Kennzahlen der Graphen auf.

[2] Areas, die nicht stark zusammenhängen, werden weiter in ihre Komponenten zerlegt, hier nicht illustriert.

Tabelle 7.1.: Details der verwendeten Graphen

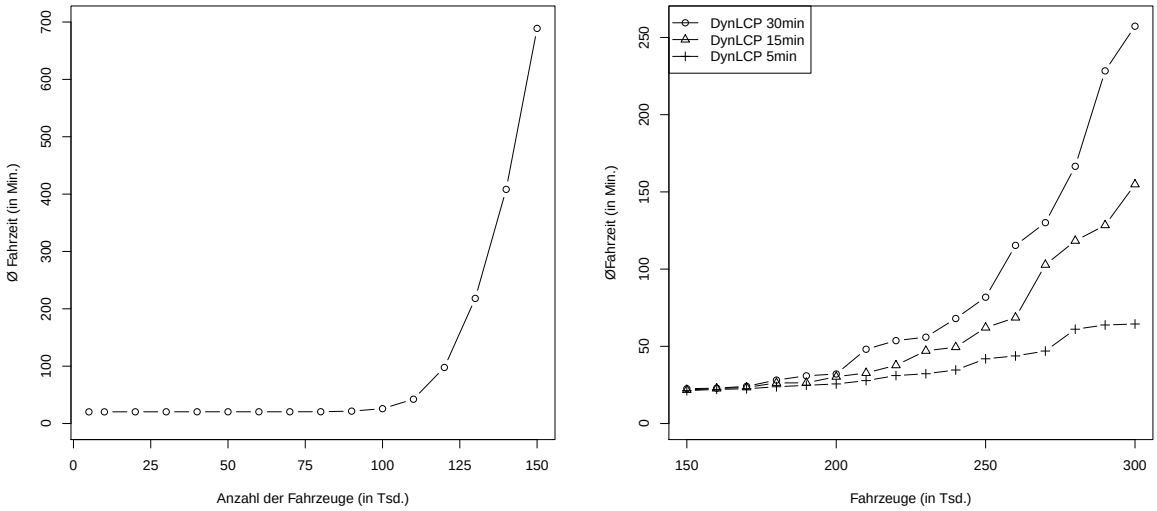
	Ruhrgebiet		Shanghai	
	Knoten	Links	Knoten	Links
Physikalischer Graph	5616	10185	3813	7570
Routing Graph	1783	4638	2112	5539
Net Layer Graph	1041	1171	1519	150
Communication Layer Graph	151	344	164	409

Wie in Abschnitt 3.1 bereits ausgeführt, ist es nicht möglich, auf realen Daten basierende Verkehrspläne zu verwenden. Stattdessen wird für jedes Fahrzeug zufällig ein Start- und Zielknoten bestimmt, sowie eine Startzeit $t \in [t_0, t_{end}]$, wobei t_0 den Simulationsbeginn und t_{end} das Ende der Fahrzeugerzeugung, nicht zwangsläufig aber das Simulationsende, darstellt. Für die Simulationen in dieser Arbeit werden Pläne erstellt, dessen Startzeiten in den ersten fünf Stunden der Simulation liegen. Die Simulation beginnt zum Zeitpunkt t_0 und läuft bis alle Fahrzeuge angekommen sind, d.h. im Regelfall über die fünf Stunden hinweg. Das hat zwei Ursachen. Zum Einen natürlich, dass Fahrzeuge, die gegen Ende starten, erst nach Ende der fünften Stunde ihr Ziel erreichen werden. Der wesentlichere Grund ist aber der Umstand, dass der eingesetzte Verkehrssimulator Fahrzeuge erst starten lässt, wenn die Startlinks über genügend freien Raum verfügen. Auf einem vollständig gestauten Link ist keinerlei freier Raum, wodurch dort startende Fahrzeuge vom Simulator verzögert werden, bis dort genügend Raum ist. In der Realität gibt es hingegen Einfädelungen, in dem ein startendes Fahrzeug, bspw. von einem Parkplatz, von den übrigen Fahrzeuge auf die Straße gelassen wird. So kann es aufgrund dieser Verzögerungen vorkommen, dass Fahrzeuge mitunter erst nach Ende der fünf Stunden auf ihren Startlink kommen.

Insgesamt befinden sich zu Beginn der Simulation demnach erst wenige Fahrzeuge, dann im Verlauf eine, bis zu einem Maximum ansteigende Anzahl und schließlich eine fallende Anzahl von Fahrzeugen im Straßennetz, bis alle Fahrzeuge ihr Ziel erreicht haben.

Die Abbildung 7.8a zeigt die durchschnittlichen Fahrzeiten unterschiedlicher Fahrzeuganzahlen unter Verwendung des LCP-Protokolls, demnach ein Routing gemäß der FFT-Metrik. Deutlich wird, dass bis zu ca. 100.000 Fahrzeugen nahezu keine Steigerung der durchschnittlichen Fahrzeiten zu beobachten ist. Sodann steigen diese jedoch von ca. 25 Minuten (bei 100.000 Fahrzeugen) auf über ca. 680 Minuten (bei 150.000 Fahrzeugen). Es entsteht ca. eine Versiebenundzwanzigfachung der durchschnittlichen Fahrzeiten, bei gerade einmal der anderthalbfachen Verkehrsnachfrage. Das ist aus zweierlei Gründen kritisch zu betrachten:

1. Offensichtlich handelt es sich um unrealistisches Verhalten. Zum Einen ist das durch das Queue-basierte Verkehrsmodell des verwendeten Simulators zu erklären, welcher z.B. keine Kreuzungen und Überholvorgänge abbildet. Stattdessen bewegen sich die Fahrzeuge mit der Freiflussgeschwindigkeit, bis die Maximalkapazitäten der Links ausgeschöpft sind. Danach tritt durch jedes weitere Fahrzeug eine zusätzliche und deutliche Verzögerung auf. In diesem konkreten Fall tritt diese Sättigung ab ca. 100.000 Fahrzeugen auf. Ein weiterer Punkt ist das verwendete Protokoll. Jedes Fahrzeug berechnet zu Beginn den LCP gemäß der FFT und ändert den dadurch bestimmten Pfad während der Fahrt nicht.



(a) LCP (b) DynLCP

Abbildung 7.8.: Ergebnisse der LCP-Protokolle

Die Autobahnen bspw. sind daher stark überlastet, kleinere Straßen werden kaum benutzt. In der Realität würden menschliche Fahrer, die die Umgebung kennen und häufiger diese Strecke fahren, Ausweichrouten wählen. Gerade das ist das zugrundeliegende Konzept des UE. In diesen Simulationen richten sich die Fahrer aber rein nach den Angaben des Protokolls, in diesem Fall schlicht dem nach FFT-LCP.

2. Darüber hinaus ergibt sich ein Problem bei der Gestaltung der Simulationsstudien. Werden zu wenig Fahrzeuge verwendet, bleiben die Fahrzeiten sehr nahe an der FFT. Werden nur „etwas“ zu viele Fahrzeuge eingesetzt, steigen die Fahrzeiten sehr stark an, bis hin zu vollkommen unrealistischen Ergebnissen. Denn eine durchschnittliche Fahrzeit von über 11 Stunden auf diesem Ruhrgebietsausschnitt ist selbst bei erheblichem Stau in der Realität keineswegs beobachtbar.

Verschärft wird diese Situation, sobald unterschiedliche Protokolle verglichen werden sollen. Sind die Fahrzeiten, welche durch ein Protokoll entstehen, bspw. bereits unrealistisch hoch, können die Fahrzeiten eines anderen Protokolls noch nahe an der FFT sein.

Die Abbildung 7.8b zeigt die Ergebnisse des DynLCP-Protokolls und verdeutlicht dieses Problem. Bei 150.000 Fahrzeugen ist die durchschnittliche Fahrzeit noch gering, im Gegensatz zu den vorherigen Ergebnissen des LCP-Protokolls. Auch wird die besondere Bedeutung der Aktualisierungsfrequenz ersichtlich. Bei 300.000 Fahrzeugen besteht ein Unterschied in den durchschnittlichen Fahrzeiten etwa um den Faktor 5 zwischen einer Aktualisierungsfrequenz von 5 und 30 Minuten.

Die Simulationsstudien sind im Folgenden so gestaltet, dass sich jeweils möglichst vergleichbare Situationen ergeben, die diese Umstände berücksichtigen. Dabei wird das DynLCP 5min-Protokoll als direkter BeeJamA Konkurrent eingesetzt, da dies eine typische zeitliche Auflösung kommerzieller VRGS darstellt. DynLCP 30min hingegen wird für „Hintergrundver-

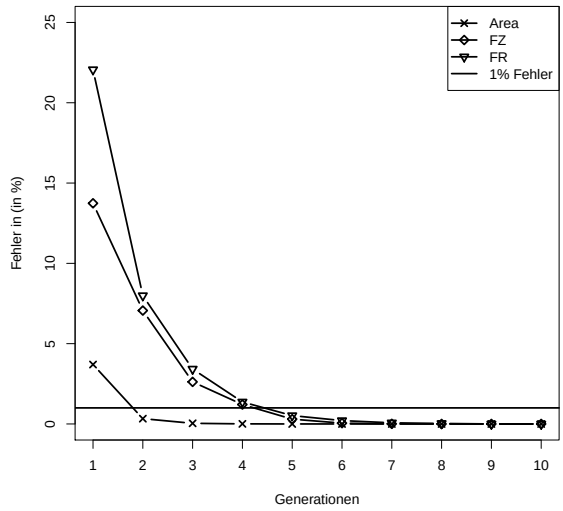


Abbildung 7.9.: LCP-Fehler

kehr“ verwendet, der einzig dazu eingesetzt wird, eine gewisse Grundauslastung der Links zu sichern. Ein FFT-LCP-Protokoll hingegen würde bei 300.000 Fahrzeugen nicht vertretbare Fahrzeiten und Link-Verstopfungen erzeugen.

7.2. BeeJamA-Basisprotokoll

Um das BeeJamA-Protokoll mit anderen Protokollen vergleichen zu können, wurden die folgenden Parameter verwendet. Die Kantenlänge einer Gridbox beträgt zunächst 1500 Meter und das Hop Limit der Foraging Zone-Scouts zwei Areas. Pro Area wird ein repräsentativer Knoten (zufällig) ausgewählt und keine weiteren Ebenen auf dem Net Layer verwendet. Durch die Grid-Aufteilung entstehen 151 Areas im Falle des Ruhrgebiet- und 164 Areas im Falle des Shanghai-Netzes. Das Aktualisierungsintervall beträgt eine 1 Sekunde, somit wird jede Sekunde eine neue Generation von Scouts gestartet. Diese hohe Frequenz ist realistisch, denn ob der linearen Nachrichtenkomplexität, kann das Protokoll kontinuierlich CCP-Kosten disseminieren ohne Netzwerkressourcen überzustrapazieren.

7.2.1. Propagierungsstrategien

Zunächst seien die beiden Propagierungsstrategien aus Abschnitt 6.4 untersucht. Ausgangspunkt sind uninitialisierte Tabellen zu Beginn einer Simulation, in welchen keine Einträge vorhanden sind. Das stellt eine Art Worst Case dar, da für keinen Zielknoten korrekte Kosten verzeichnet sind.

Abbildung 7.9 zeigt die Entwicklung des LCP-Fehlers über die Generationen bei Verwendung der Best-Strategie für den Ruhrgebietsgraphen. Der angegebene Fehler entspricht der durchschnittlichen Abweichung der in die Tabellen eingetragenen Kosten von denen des tatsächlichen LCP. Wie ersichtlich, ist der initiale Fehler umso kleiner, je geringer die Reichweite eines Scouts ist. In einer einzelnen Area existieren weniger Pfadalternativen, folglich gibt es weniger

Tabelle 7.2.: Fahrzeiten (in Minuten) / Ruhrgebiet

	1. Quartil	Median	Durchschnitt	SD	3. Quartil	Maximum
Free Flow Time	12,75	19,46	20,33	10,00	26,93	55,75
BeeJamA (100%)	13,73	21,42	22,41	11,76	30,03	61,12
DynLCP (5min/100%)	15,23	24,95	29,06	19,03	38,92	209,00
BeeJamA (30%)	15,40	25,38	30,41	21,26	39,62	174,10
DynLCP (5min/30%)	17,15	30,82	40,24	32,41	52,48	261,80

Fehlermöglichkeiten. Die Long Distance Scouts des Net Layers haben eine größere Reichweite als die Short Distance Scouts. Daher weisen die FR-Einträge größere Fehler auf, als die FZ-Einträge.

Nach spätestens fünf Generationen liegt der Fehler jeweils unter 1%, d.h. aus praktischer Sicht können die Tabellen zu diesem Zeitpunkt als konvergiert bezeichnet werden. Der Fehlerwert nach der ersten Generation entspricht dabei dem Ergebnis der First-Strategie, da bei der Best-Strategie initial ebenfalls der erste Scout weitergeleitet wird. Mehrfache Ausführung der First-Strategie führte zu sehr ähnlichen Ergebnissen, d.h. bspw., dass die FR-Einträge zu den repräsentativen Knoten während der gesamten Simulationslaufzeit über 20% von den tatsächlichen Werten abweichen können. Der Best-Strategie wurde folglich bei den kommenden Simulationen der Vorzug gegeben. Ändern sich demnach Transitkosten im Netz, stellt die Best-Strategie sicher, dass innerhalb weniger Generationen, sich diese Änderung im Netz verbreitet hat.

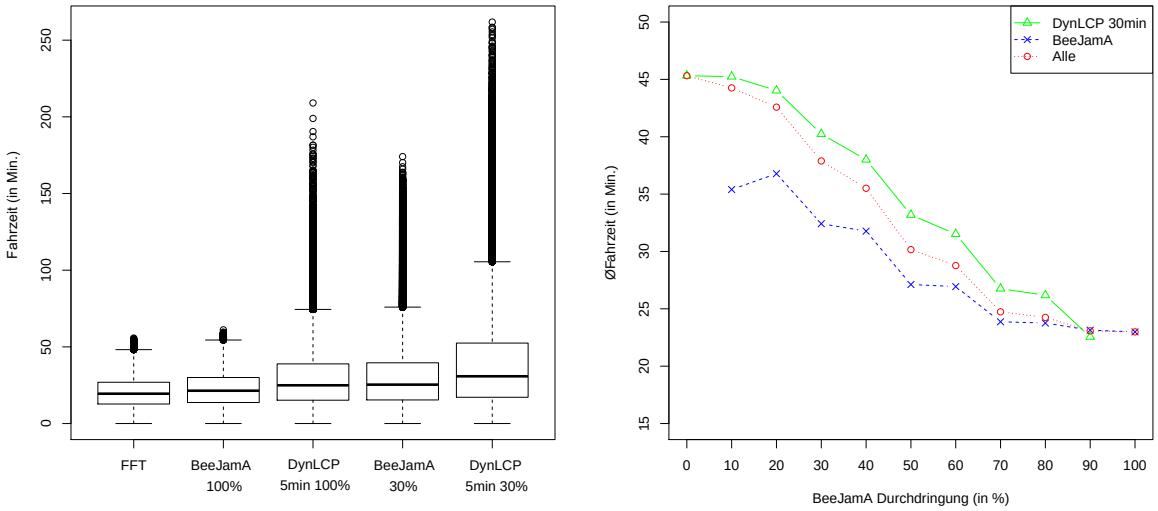
Die Best-Disseminationsstrategie ist auch ein gutes Beispiel von kooperativer Schwarmintelligenz. Über Generationen hinweg arbeiten unterschiedliche Scout-Typen über Schichten hinweg zusammen und verbessern eine globale Eigenschaft, die LCP-Kostenabweichung. Dabei fließen die Ergebnisse der Area-Scouts in die Kosten der Net Layer-Scouts mit ein und Scouts der gleichen Hierarchie ermitteln gemeinsam den LCP, der, wie hier gezeigt, bereits nach wenigen Generationen im Netz bekannt ist.

7.2.2. Kumulierte Ergebnisse

Für einen ersten BeeJamA-Vergleich werden die Ergebnisse einer Simulation mit 230.000 Fahrzeugen betrachtet. Die Tabelle 7.2 zeigt wesentliche Eigenschaften[3] der Fahrzeitverteilungen des BeeJamA- und des DynLCP 5min-Protokolls bei zwei Durchdringungen. Zusätzlich ist die FFT-Verteilung angegeben. Letztere wird ermittelt, in dem für jedes Fahrzeug die FFT bestimmt wird, einfach durch Summierung der FFT-Transitzeiten der Links des LCP. Abbildung 7.10a zeigt zugehörige Boxplots.

In einer realen Umgebung ist die FFT-Verteilung – selbst ohne zusätzlichen, störenden Verkehr – praktisch nicht zu erreichen. Denn Ampeln, Verzögerungen durch suboptimales Fahrverhalten (z.B. zu starkes Abbremsen, welches als Ursache für den „Stau aus dem Nichts“ identifiziert wurde) und Passagen, wie z.B. Kreuzungen, die in der Regel nicht mit Maximalgeschwindigkeit des Links befahrbar sind, verlängern üblicherweise die Fahrzeit. Hinzu

[3] SD steht dabei für die (empirische) Standardabweichung.



(a) Boxplots

(b) Unterschiedliche Durchdringungen

Abbildung 7.10.: Fahrzeiten im Ruhrgebiet

kommt der üblicherweise stark verzögernde, zusätzliche Verkehr. Die FFT-Verteilung stellt demnach ein praktisch nicht erreichbares Optimum dar.

Zumindest unter theoretischen Gesichtspunkten wären aber immerhin ein System Optimum bzw. ein User Equilibrium möglich. Da sich diese aber nicht von selbst einstellen, bedürfte es einer entsprechenden Flussoptimierung. Jedoch müssten dazu die stark einschränkenden Voraussetzungen erfüllt sein, wie sie in Kapitel 4 diskutiert wurden. Unter anderem müsste a priori die gesamte Verkehrsnachfrage, inkl. Abfahrtszeiten, bekannt sein und keinerlei unplanmäßigen Ereignisse, wie z.B. Unfälle, dürften auftreten. Offenkundig ist dies unrealistisch, weswegen in der Regel auch diese beiden Verteilungen nur eine theoretische Untergrenze darstellen. Betrachtet man das Ergebnis von BeeJamA bei einer Durchdringung von 100%, ist festzustellen, dass die Ergebnisse aber sogar sehr nahe an denen der FFT-Verteilung sind. So beträgt die Differenz bei der durchschnittlichen Fahrzeit in etwa 2 Minuten (20,33 min gegen 22,41 min). Die maximale Fahrzeit unterscheidet sich mit ca. 5,5 Minuten schon etwas deutlicher.

Das DynLCP Ergebnis, selbst bei 100% Durchdringung, unterscheidet sich deutlich. So ist allein der Durchschnitt fast 6,5 Minuten höher, ganz zu schweigen von Maximalfahrzeit von über 200 Minuten. Der Boxplot zeigt deutlich die große Anzahl von Ausreißern. Es sind nur zwei Gründe für diese erhebliche Erhöhung der Fahrzeiten möglich: 1) Die Fahrzeuge fahren ohne Not deutlich längere Strecken oder 2) die Transitzeiten sind höher. Der erste Fall ist unmittelbar aufgrund des LCP-Algorithmus auszuschließen. Stattdessen können nur verstopfte Links, vulgo Stau, ursächlich sein. Die späteren intra-simulativen Untersuchungen werden das noch genauer aufzeigen.

Ferner sind in der genannten Tabelle noch die Ergebnisse bei einer Durchdringung von jeweils 30% dargestellt. Die jeweils übrigen 70% sind mit vom DynLCP 30min-Protokoll geleiteten Fahrzeugen aufgefüllt. Diese 70% sollen Fahrer imitieren, die bspw. über Radiosendungen oder

Tabelle 7.3.: Fahrtstrecken (in Kilometer)

	1. Quartil	Median	Durchschnitt	SD	3. Quartil	Maximum
BeeJamA	11,52	18,87	20,13	11,05	27,08	47,53
DynLCP 5min	11,58	19,08	20,62	11,56	28.01	49,52

TMC informiert werden. Es handelt sich im Rahmen dieser Simulationsszenarien um Fahrzeuge die als „Hintergrundverkehr“ dienen: In die Auswertung fließen nur die Fahrzeiten der 30% ein. Betrachtet werden nur nur die 30%. Interessantes Ergebnis ist, dass BeeJamA 30% eine geringere Maximalfahrzeit erzielt als DynLCP 100%. Die höhere Aktualisierungsfrequenz hat somit erhebliche positive Auswirkungen.

Abschließend seien noch die Fahrtstrecken in Tabelle 7.3 verglichen. Mit Ausnahme von geringen Vorteilen zugunsten BeeJamAs sind dabei keine strukturellen Unterschiede ersichtlich gewesen. BeeJamA schafft es demnach Fahrzeuge ohne Umwege im Sinne der zurückgelegten Strecke so zu routen, dass die Fahrzeiten deutlich eher ans Ziel ankommen, da unterwegs weniger Stau entsteht. Da dieser Befund bei allen Simulationen zu finden war, wird im Folgenden auf den Aspekt der zurückgelegten Fahrtstrecke nicht mehr weiter eingegangen.

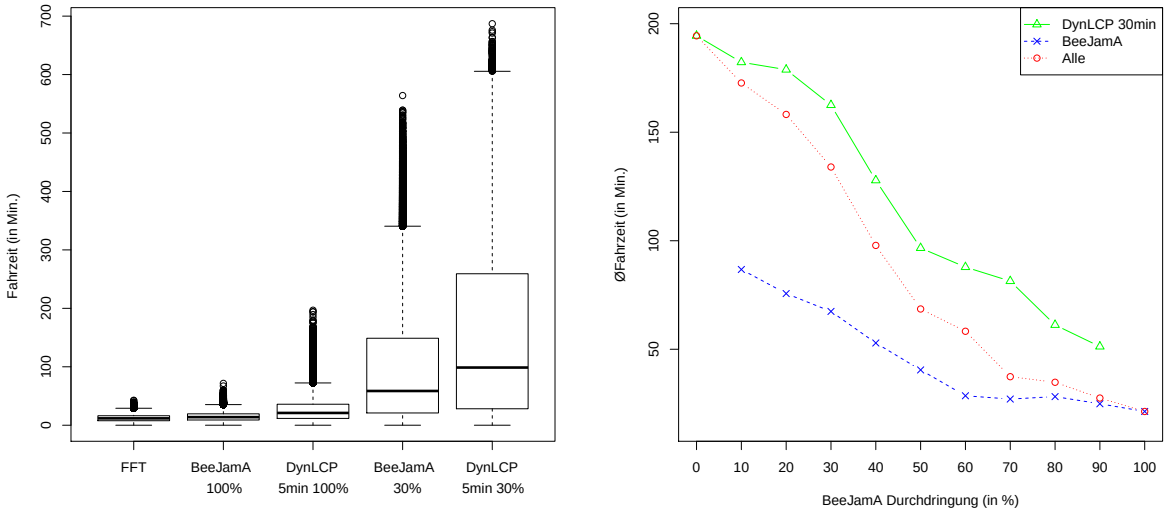
Auswirkungen partieller Durchdringungen Gerade eine unvollständige Durchdringung ist untersuchungswürdig, wird doch kein Protokoll – besonders zu Beginn einer Einführung – einen hohen, oder gar vollständigen Marktanteil vorweisen können. Abbildung 7.10b zeigt die durchschnittlichen Fahrzeiten bei einer ansteigenden BeeJamA-Durchdringung. Jede Simulation wurde mit insgesamt 230.000 Fahrzeugen durchgeführt. Drei Werte sind dabei abgebildet: Die durchschnittliche Fahrzeit der DynLCP 30min-Fahrzeuge, die BeeJamA-Fahrzeuge und die aller Fahrzeuge. Erkennbar ist die Tendenz, dass, je höher der BeeJamA-Anteil ist, desto geringer die durchschnittlichen Fahrzeiten sind – und das interessanterweise für alle Fahrer. Zunächst, bei 0% BeeJamA-Anteil (und folglich 100% DynLCP 30min-Anteil) liegt die durchschnittliche Fahrzeit bei ca. 45 Minuten. Bei einem BeeJamA-Anteil von 10% sinken die durchschnittlichen Fahrzeiten aller Fahrzeuge um etwa zwei Minuten, da die BeeJamA Fahrzeuge im Schnitt ca. 35 Minuten unterwegs sind. Eine Verringerung der Fahrzeiten der LCP-geleiteten Fahrzeuge tritt jedoch erst ab einem 20%-igen BeeJamA-Anteil ein, allerdings nur um eine Minute. Deutlicher wird der Abfall erst ab einem Anteil von 30%. Auffällig ist der leichte Anstieg der durchschnittlichen Fahrzeit der BeeJamA-Fahrzeuge bei 20% BeeJamA-Anteil, um ca. 1,5 Minuten. Dieser Effekt konnte in weiteren Simulationen ebenfalls beobachtet werden. Ursächlich hierfür ist ein höher Teil von „agilen“ Fahrzeugen, *womit die Fähigkeit bezeichnet sei, öfter zur Fahrtzeit auf veränderte Verkehrssituation reagieren zu können.* Denn BeeJamA-Fahrzeuge, mit der im Vergleich zum DynLCP 30min-Protokoll sehr hohen Aktualisierungsfrequenz von einer Sekunde, können damit an jedem Knoten anhand von aktuellen Verkehrsinformationen eine angepasste Next Hop-Routenempfehlung erhalten. Im Schnitt passiert ein Fahrzeug auf dem Ruhrgebietsgraphen etwa 48 Knoten und stellt dementsprechend viele Requests. Die DynLCP 30min-Fahrzeuge hingegen erhalten einmal zu Beginn und dann frühestens alle 30 Minuten während der Fahrt nochmals eine Routenempfehlung. Ab ca. 70% BeeJamA-Anteil fällt die durchschnittliche Fahrzeit der DynLCP 30min-Fahrzeuge unter 30 Minuten, von da ab werden die meisten dieser Fahrzeuge sogar nur noch die initiale Routenempfehlung bekommen.

Aufgrund dieser hohen *Agilität* schaffen es die BeeJamA-Fahrzeuge, bei einem 10%-Anteil, Routen abseits der stark überfüllten Links zu finden. Steigt nun der Anteil dieser agilen Fahrzeuge, kommt es zu dem Effekt, dass die Nebenrouten ebenfalls eine höhere Auslastung aufweisen. Zusammen mit dem Umstand, dass die Hauptrouten nicht vollständig vermieden werden können, ist ein leichter Anstieg der BeeJamA-Fahrzeiten bei 20% Durchdringung zu beobachten. Gleichzeitig sinkt aber die Überlast auf den Hauptrouten und die Fahrzeiten der DynLCP 30min-Fahrzeuge sinken ebenfalls (leicht). Ab 30% BeeJamA-Anteil dreht sich das Verhältnis. Die Last auf den Hauptrouten verringert sich, die DynLCP 30min-Fahrzeuge erreichen schneller ihr Ziel. Die agileren Fahrzeuge nutzen Nebenrouten, profitieren aber auch von der Lastreduzierung auf den Hauptrouten. Insgesamt ergibt sich eine durchschnittliche Fahrzeitreduzierung um etwa 7 Minuten für alle Fahrzeuge. Diese Reduzierung setzt sich fort bis 90% BeeJamA-Anteil. Dabei ist die durchschnittliche Fahrzeit der DynLCP 30min-Fahrzeuge sogar etwas unterhalb der BeeJamA-Fahrzeuge. Erklärlich ist dies durch den Umstand, dass die erstgenannten Fahrzeuge größtenteils nur noch die initiale Routenempfehlung bekommen und somit quasi einer statischen Route folgen. Die BeeJamA-Fahrzeuge hingegen sind deutlich in der Überzahl, und folgen, ob ihrer Agilität, stets den zum aktuellen Zeitpunkt in der Tabelle eingetragenen ELCP-Pfadknoten. Nur geschieht dies unkoordiniert, denn eine Routingentscheidung seitens eines Navigators basiert nur auf den CCP-Pfadkosten, welche durch die Scouts verbreitet wurden, jedoch nicht auf den intendierten Routen anderer Fahrzeuge. So kann bspw. ein Next Hop auf einer Route zu einem Entscheidungszeitpunkt zwar gewählt werden, diese Route in naher Zukunft aber überlastet sein, da zu viele Fahrzeuge auf diese Route ausweichen wollen. Anders ausgedrückt, die DynLCP 30min-Fahrzeuge bleiben auf ihrer ursprünglichen Route, die übrigen BeeJamA-Fahrzeuge weichen auf weniger belastete Nebenrouten aus, tun dies aber unkoordiniert und erzeugen dadurch erneut eine erhöhte Auslastung auf diesen Ausweichrouten. Die DynLCP 30min-Fahrzeuge hingegen fahren davon unberührt auf ihrem initialen LCP, unterlassen Ausweichversuche und erreichen schließlich ihr Ziel sogar etwas schneller. Allerdings konnte dieser Effekt nur bei einem 90% und 100% BeeJamA-Anteil auf dem Ruhrgebietsgraph beobachtet werden, bei dem nachfolgend noch vorgestellten Shanghai-Netz hingegen nicht. Bei einem 100% BeeJamA-Anteil liegt die durchschnittliche Fahrzeit unwesentlich über der im 90%-Fall, aufgrund der höheren Agilität der BeeJamA-Fahrzeuge. Das häufige Ändern der Routenintention ist folglich vielleicht nicht stets die beste Lösung, aber eine erhebliche Verbesserung ist allemal möglich (im Vergleich zum Ausgangspunkt von 0% Durchdringung). Zumal dieser Effekt gering ausgeprägt ist und nur bei sehr hohen Durchdringungen hier beobachtet wurde. Realistisch sind solche Durchdringungen wohl nicht, da ein solches Marktmonopol unwahrscheinlich erscheint. Zum Einen aufgrund konkurrierender (technischer) Systeme und zum Anderen aufgrund des Umstandes, dass stets ein gewisser Teil der Fahrzeuge gänzlich auf den Einsatz solcher Systeme verzichten wird, bspw. Pendler, die die Umgebung hinreichend gut kennen.

Auswirkungen partieller Aktualisierungen Die vorangegangenen Simulationen boten dem DynLCP-Protokoll den Vorteil, Aktualisierungen für alle Links berücksichtigen zu können. In der Realität, so eine Grundannahme dieser Arbeit, ist dies aufgrund der Zentralisierung des DynLCP-Protokolls für größere Straßennetze nicht umsetzbar. Stattdessen muss eine Limitierung in der abgedeckten Fläche oder in der Aktualisierungsfrequenz vorgenommen werden. Das kommerziell erhältliche Produkt TomTom HD Traffic deckt bspw. nur größere Straßen ab und bietet eine Aktualisierungsfrequenz von mehreren Minuten. Für das Shanghai-

Tabelle 7.4.: Fahrzeiten (in Minuten) / Shanghai

	1. Quartil	Median	Durchschnitt	3. Quartil	Max
Free Flow Time	7,70	11,73	12,25	16,23	42,15
BeeJamA (100%)	8,95	13,85	14,67	19,48	71,60
DynLCP (5min/100%)	11,65	21,00	26,43	35,93	196,80
BeeJamA (30%)	20,98	58,58	94,96	148,80	564,10
DynLCP (5min/30%)	28,15	98,68	150,60	259,20	687,00



(a) Boxplots der Fahrzeiten

(b) Unterschiedliche Durchdringungen

Abbildung 7.11.: Fahrzeiten im Shanghai-Netz

Netz wurden daher Simulationen durchgeführt, bei denen das DynLCP-Protokoll nur auf den grünen Links des Graphen in Abbildung 7.5 Aktualisierungen erhält.

Die Ergebnisse sind in Tabelle 7.4 und Abbildung 7.11a dargestellt. Prinzipiell sind die Resultate strukturell ähnlich. Die absoluten Unterschiede fallen dabei allerdings noch deutlicher auf. So ist bei 100% Durchdringung die durchschnittliche Fahrzeit bei DynLCP 5min schon fast doppelt so hoch wie bei BeeJamA und die Verschlechterung von DynLCP 5min bei 100% zu 30% Durchdringung ist beträchtlich.

Wie an den Boxplots gut abzulesen ist, sind auch insbesondere Varianz und Maximalfahrzeiten bei dem DynLCP-Protokoll erheblich höher. Die Links, die aus der Sicht des DynLCP-Protokolls nicht aktualisiert werden, sind aufgrund dauerhafter Kosten in Höhe der FFT günstig – zumindest im Vergleich zu Links, die eine gewisse Auslastung erfahren. Daher werden die Links mit höherer Wahrscheinlichkeit in berechnete LCPs aufgenommen, einhergehend mit entsprechend erhöhter Auslastung dieser Links.

Somit ist auch erklärbar warum die durchschnittlichen Fahrzeiten bei größerer BeeJamA-

Tabelle 7.5.: Durchschnittliche Fahrzeiten (in Min.) in Abhängigkeit der Anzahl der Areas

Gridboxgröße	1500m	3000m	4500m	6000m	7500m
ø Fahrzeit	22,41	24,50	25,16	26,43	28,06
#Areas	151	96	47	27	20

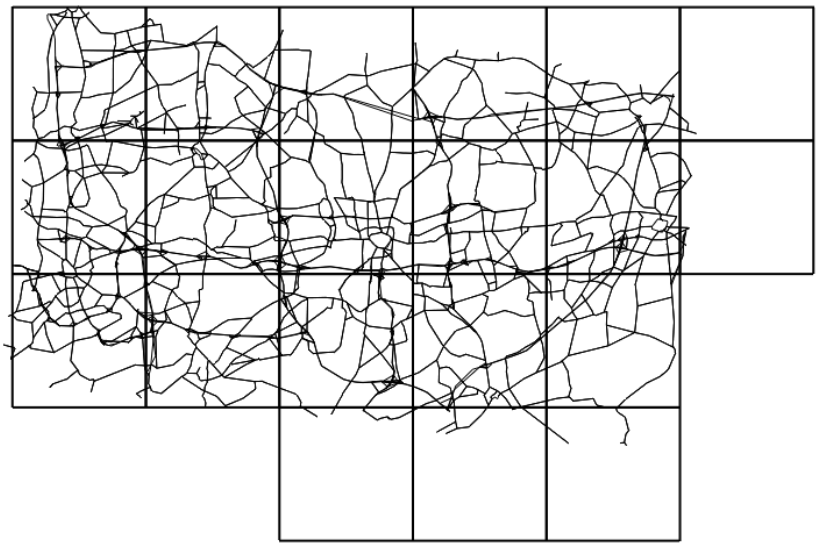


Abbildung 7.12.: Area-Aufteilung bei einer Gridboxgröße von 7500m

Durchdringung so stark fallen (siehe Abbildung 7.11b).

Die Einschränkung auf Links mit einer Maximalkapazität größer 1200 Fahrzeugen pro Stunde hat demnach erheblichen Einfluss auf die Fahrzeiten. Ein verteiltes VRGS wie BeeJamA bietet deutlich besseres Skalierungspotential, um so auch niederrangige Straßenabschnitte einbeziehen zu können.

Auswirkungen der Area-Größe Bisher wurde für die Simulationen des Basisprotokolls mit 230.000 Fahrzeugen eine Gridboxgröße von 1500m verwendet. Tabelle 7.5 verdeutlicht die Auswirkungen der Vergrößerung dieses Parameters. Abbildung 7.12 zeigt die Unterteilung in Areas bei einer Boxgröße von 7500m.

Ersichtlich ist, dass die Fahrtzeiten ansteigen, je größer die Areas werden. Zwei gegenläufige Effekte spielen dabei eine Rolle:

1. Aufgrund größer werdender Areas sind präzise Kosten direkt zum Zielknoten in einem größeren Umkreis um das Ziel verfügbar.

2. Da pro Area genau ein repräsentativer Knoten existiert, vergröbert sich die Zielauflösung, da eine größere Fläche auf jeden repräsentativen Knoten kommt.

Da die Fahrzeiten steigen, scheint der zweite Effekt ein stärkeres Gewicht zu haben in diesem Szenario. Dennoch ist selbst bei einer Größe von 7500m immer noch ein Vorteil von einer Minute im Vergleich zu DynLCP 5min zu verzeichnen (vgl. Tabelle 7.2).

Je größer eine Area wird, desto höher wird die benötigte Rechenkraft für einen einzelnen Navigator. Zeitgleich sinkt aber auch die Anzahl der versendeten Nachrichten wie der nächste Paragraph zeigt. Zur Reduzierung der Fahrzeiten jedenfalls, bietet sich nach diesen Ergebnissen eher eine hohe Dichte von repräsentativen Knoten an und somit eine kleine Größe der Areas. Prinzipiell lassen sich auch beide Aspekte verbinden: große Areas mit mehreren repräsentativen Knoten. Dazu durchgeführte Simulationen konnten aber keine positiveren Ergebnisse liefern.

Anzahl der versendeten Nachrichten Abschließend sei noch die Anzahl der versendeten Agenten betrachtet, siehe Tabelle 7.6. Jede Instanziierung eines (Upstream-) Scouts wird dabei als eine *Agenteninstanz* gezählt (für jeden Link den ein Agent traversiert, wird eine Instanz erzeugt). Davon muss aber nur ein Bruchteil wirklich über das Netz *versendet* werden, da viele Agenteninstanzen nur innerhalb einer Area erzeugt werden und somit nur im Speicher der Navigators existieren.

Für die Aufteilung in 151 Areas des Ruhrgebietszenarios mittels eines 1500m-Grids, werden ca. 27.400 Area-Agenteninstanzen pro Generation benötigt, ca. 16.700 FZ-Agenteninstanzen und ca. 904.200 FR-Agenteninstanzen (jeweils gerundet auf zwei Stellen). Insgesamt ergibt dies 948.300 Millionen Agenteninstanzen pro Generation. Über das Netz müssen davon allerdings nur 186.300 Agenten pro Generation geschickt werden. Im Schnitt bedeutet das, dass pro Generation jeder Navigator an jeden seiner (im Schnitt) vier Nachbarnavigatoren ca. 308 Agenten versenden muss. Wohlgemerkt können mit geringem Abstand zu versendende Agenten mit gleichem Ziel(-navigator) zunächst gepuffert werden und anschließend in einem einzelnen Paket versendet werden. Ein einzelner Scout besteht nur aus dem Scoutursprung und den aggregierten Pfadkosten, kann demnach sehr knapp kodiert werden (bspw. mit zwei 2-Byte-Feldern als Nutzdaten). So entstünden pro Generation ca. ($308 \cdot 4 \cdot 151 \cdot 4$ Bytes $\approx$) 0,73 MB Nutzdaten im gesamten Kommunikationsnetz.

Die Tabelle zeigt auch die entsprechenden Angaben bei einer Area-Aufteilung bei 3000m. Insgesamt werden ca. 104 Scouts pro Generation von jedem der 96 Navigatoren an im Schnitt 5 Nachbarnavigatoren gesendet. Insgesamt resultiert dies in 0,15 MB Nutzdaten pro Generation als Gesamtkommunikationsaufkommen. Eine Verdoppelung der Grid-Kantenlänge und damit eine Vervierfachung der pro Area abgedeckten Fläche, führt in diesem Fall zu einer Halbierung der benötigten Agenteninstanzen. Wie in Tabelle 7.5 festgehalten, geht diese Vergröberung mit ca. 2 Minuten Fahrzeitverlängerung pro Fahrzeug einher.

Im Vergleich dazu: Ein Bellman-Ford-Ansatz erzeugt im Schnitt pro Exploration ca. 7500 Agenteninstanzen. Angenommen, ein jeder Knoten würde – statt des BeeJamA-Protokolls – den AsyncBF-Algorithmus ausführen, dann entstünden ca. 13,4 Millionen Agenteninstanzen. Das entspricht ca. der Anzahl an Agenteninstanzen von vierzehn BeeJamA-Generationen (bei einer Area-Aufteilung von 1500m).

Bei Nachfrage- und AsyncBF-basierten Protokollen wie D-MAS oder dem WSN-basierten Protokoll (vgl. Abschnitt 6.13) werden bei einem, im folgenden Abschnitt geschilderten, Simulationsaufbau mit 300.000 Fahrzeugen ca. 22.500 Requests pro Minute gestellt. Daher wären insgesamt ca. 169 Millionen Agenteninstanzen pro Minute nötig.

Pro Agent muss mindestens das Ziel und die aggregierten Pfadkosten transportiert werden. Allein dadurch entstünden Nutzdaten von ca. 645 MB pro Minute, wenn jeder Agent über das Netz geschickt werden müsste. Legt man eine Infrastruktur wie im BeeJamA-Ansatz zugrunde, müssten nur ca. 20% der Agenten real versendet werden (bei einer Area-Aufteilung bei 1500m). Die meisten Daten entstehen aber dadurch, wenn der gesamte Pfad mittransportiert werden

Tabelle 7.6.: Durchschnittliche Agentenanzahl pro Generation (im Ruhrgebietsnetz)

Grid-Größe (#Areas)	Typ	Agenteninstanzen	Versendete Agenten
1500m (151)	Area	27.400	0
”	FZ	16.700	9.500
”	FR	904.200	176.800
3000m (96)	Area	87.000	0
”	FZ	20.500	9.500
”	FR	429.800	40.200

Tabelle 7.7.: Fahrzeiten (in Minuten) / Ruhrgebiet / 300.000 Fahrzeuge

	1. Quartil	Median	Durchschnitt	SD	3. Quartil	Maximum
Free Flow Time	12,75	19,46	20,33	10,00	26,93	55,75
BeeJamA	16,18	27,10	30,31	17,79	42,68	145,72
DynLCP 5min	23,42	48,37	64,45	52,20	94,30	347,40
DynLCP 15min	32,90	105,72	154,85	137,97	263,68	640,52
DynLCP 30min	41,05	188,18	257,30	229,40	461,63	894,35
LCP	684,03	2927,56	3253,96	2710,78	499,69	8793,05

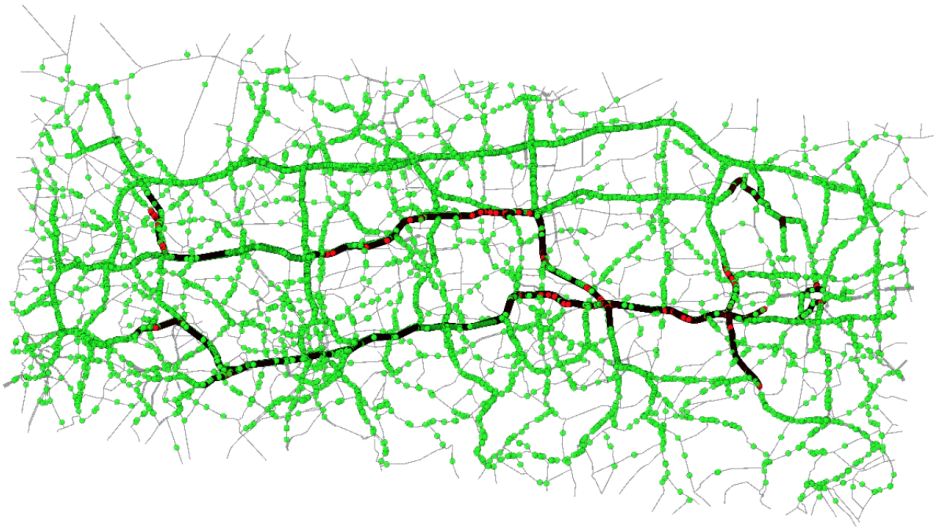
muss (Distanzpfad statt Distanzvektor), wie bei dem WSN- und D-MAS-Vefahren. Im Schnitt sind das bei dem Ruhrgebietsnetz pro Agenteninstanz 17 Knoten in der Distanzpfadknotenliste. Zusammen sind das etwa 6124 MB bzw. 1224 MB pro Minute, je nachdem, ob alle oder nur wieder 20% der Agenten real über das Kommunikationsnetz gesendet werden müssen.

Die Anzahl der Nachrichten des BeeJamA-Protokolls und die Menge der Nutzdaten ist nicht nur um Größenordnungen kleiner, sondern darüber hinaus auch unabhängig von der Anzahl der Routinganfragen.

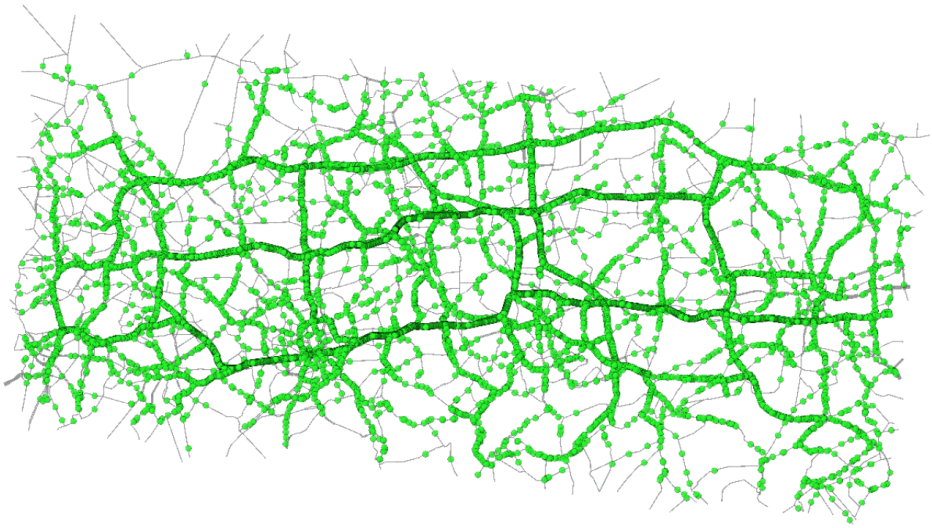
7.2.3. Intra-simulative Ergebnisse

Im vorherigen Abschnitt wurden gemittelte Kennzahlen der Protokolle vorgestellt, in diesem Abschnitt soll es um Kennzahlen über den Verlauf einer Simulation gehen. Dadurch werden die Auswirkungen der höheren Agilität des BeeJamA-Protokolls auf die Fahrzeuge ersichtlich.

Bisher wurden Simulationen mit 230.000 Fahrzeugen durchgeführt. Das BeeJamA-Protokoll ist aber in der Lage, auch bei 300.000 Fahrzeugen noch durchschnittliche Fahrzeiten zu ermöglichen, die als „realistisch“ zu betrachten sind. Im Gegensatz zu LCP- und DynLCP 30min-Protokolen, die hohe drei- oder gar vierstellige Ergebnisse liefern, siehe Tabelle 7.7. Die Unterschiede sind bei einer 100% Durchdringung schon erheblich, auf eine geringere Durchdringung wird daher bei dieser Fahrzeuganzahl verzichtet. Statt auf solchen aggregierten Ergebnissen, liegt der Fokus in diesem Abschnitt auf den, die Dynamik beeinflussenden, Faktoren im Verlauf einer Simulation. Dabei wird die höhere Agilität des BeeJamA-Protokolls



(a) LCP



(b) BeeJamA

Abbildung 7.13.: Auslastungsübersicht 3h nach Simulationsbeginn mit LCP 100%

im Vergleich zu DynLCP deutlich.

Eine visuell anschauliche Übersicht der Auslastung zu einem fixen Zeitpunkt in der Simulation, drei Stunden nach Simulationsbeginn, einmal mit dem LCP 100%-Protokoll und einmal mit dem BeeJamA 100%-Protokoll auf einem Ruhrgebietsnetz, ist in den Abbildungen 7.13a und 7.13b dargestellt[4]. Fahrzeuge, die schneller als 50% der FFT fahren, werden grün dargestellt, langsamere rot. Das ist eine recht grobe Darstellung, man stelle sich vor, dass erst ab 65km/h auf einer Autobahn (mit Richtgeschwindigkeit von 130km/h) von Stau die Sprache wäre. Dennoch ist deutlich zu erkennen, dass die Autobahnen im Falle des LCP-Protokolls

[4]Die Darstellungen wurden mit dem MATSim-Visualisierungstool erstellt. Auf die Parameter ab welcher Schwelle die Fahrzeuge wie eingefärbt werden, war kein Einfluss zu nehmen.

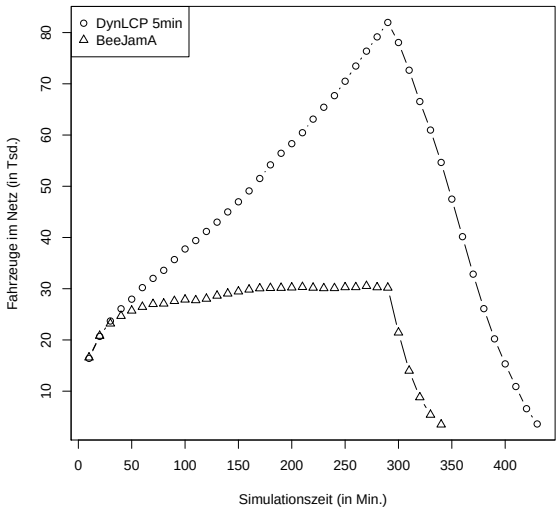


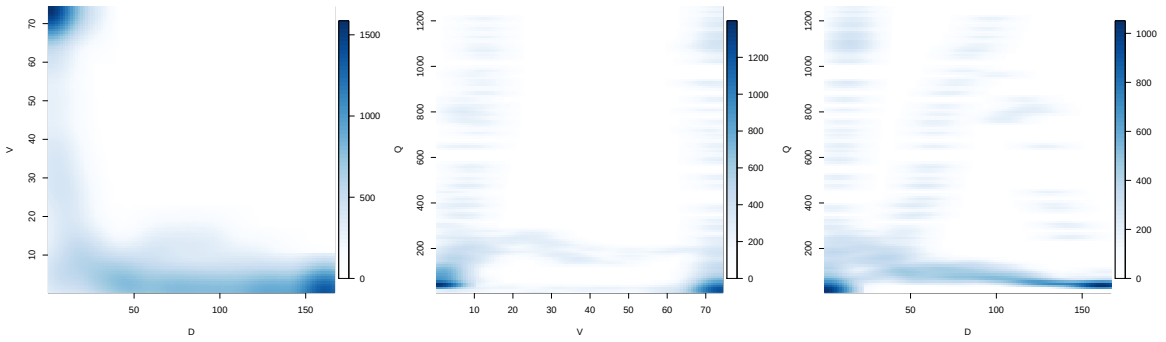
Abbildung 7.14.: Anzahl der Fahrzeuge im Netz

stärkeren „Stau“ aufweisen. In der Vergleichsabbildung des BeeJamA-Protokolls ist die „Staudichte“ geringer. Die „schwarzen Linien“ (insb. in der ersten Abbildung) entstehen durch die sehr hohe Dichte der eigentlich als rote Kreise dargestellten Fahrzeuge. Da diese Kreise aber schwarze Umrandungen besitzen, erscheint bei hoher Dichte eine schwarze Linie.

Einen weiteren Einblick gewährt die Abbildung 7.14, in welcher die Anzahl der zum jeweiligen Simulationszeitpunkt im Netz befindlichen Fahrzeuge dargestellt werden. Während im DynLCP 5min-Fall die Anzahl bis ca. 80.000 Fahrzeuge ansteigt, bleibt die Anzahl im BeeJamA-Fall bei ca. 30.000 Fahrzeuge konstant. Ab der fünften Simulationsstunde (300min) sinkt die Anzahl bei beiden Protokollen, da Fahrzeuge, wie bereits erwähnt, nur bis zu diesem Zeitpunkt erzeugt werden. Während die Links unter dem DynLCP-Protokoll verstopfen, kommen die Fahrzeuge unter BeeJamA in etwa in der gleichen Rate an ihr Ziel an, wie neue in die Simulation treten (sonst würde die Anzahl ebenfalls ansteigen).

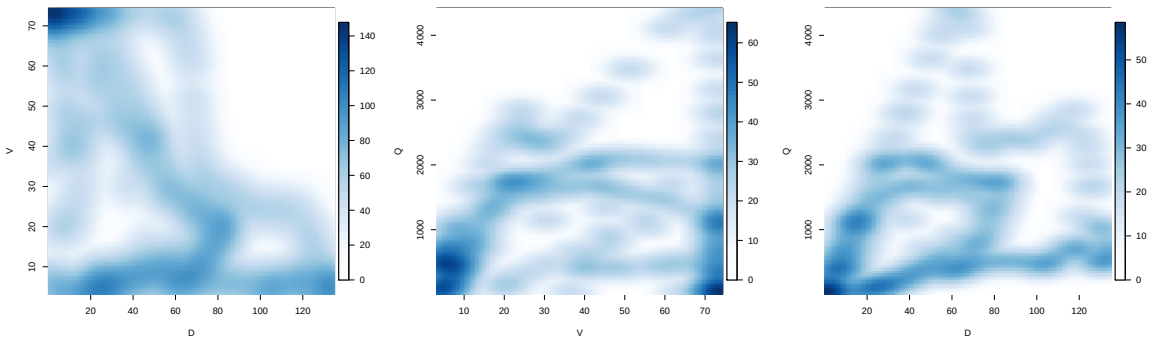
Die Abbildungen 7.15, 7.16 und 7.18 stellen empirische DV-, VQ-, DQ-Kurven (vgl. Abschnitt 3.3.1) des längsten Links (ca. 10km) des Routinggraphs einer Ruhrgebiets-Simulation mit vollständiger LCP-, DynLCP 30min- und 15min- bzw. BeeJamA-Durchdringung dar. Auf eine Darstellung des DynLCP 5min-Protokolls wurde verzichtet, da der visuelle Unterschied weniger ausgeprägt ist. Pro Minute wurde eine Messung pro Link gespeichert. Es handelt sich bei den Darstellungen um eine zweidimensionale Kerndichteschätzung der diskreten (gemittelten) Messwerte, die für den Link und pro Minute aufgezeichnet wurden. Die Farbelegende rechts gibt eine Zuordnung zu der Anzahl der Messungen.

Zunächst fällt auf, dass diese empirischen Diagramme sich im Aussehen deutlich von den theoretischen unterscheiden, unabhängig vom Protokoll (bzw. dessen Parametrisierung). Bei genauerem Hinsehen ist aber eine Transition vom LCP-Protokoll zu BeeJamA (über DynLCP 30min und 15min) zu erkennen. Während im DV-Diagramm des LCP-Protokolls nur bei geringer Dichte hohe Geschwindigkeiten auftreten, sind bei BeeJamA auch bei mittleren und hohen Dichten deutliche höhere Geschwindigkeiten aufgetreten. Besonders auffällig sind die Cluster bei der geringsten und höchsten Dichte in dem LCP-Diagramm: entweder es herrscht



(a) DV (b) VQ (c) DQ

Abbildung 7.15.: Empirische Fundamentaldiagramme bei vollständiger LCP-Durchdringung



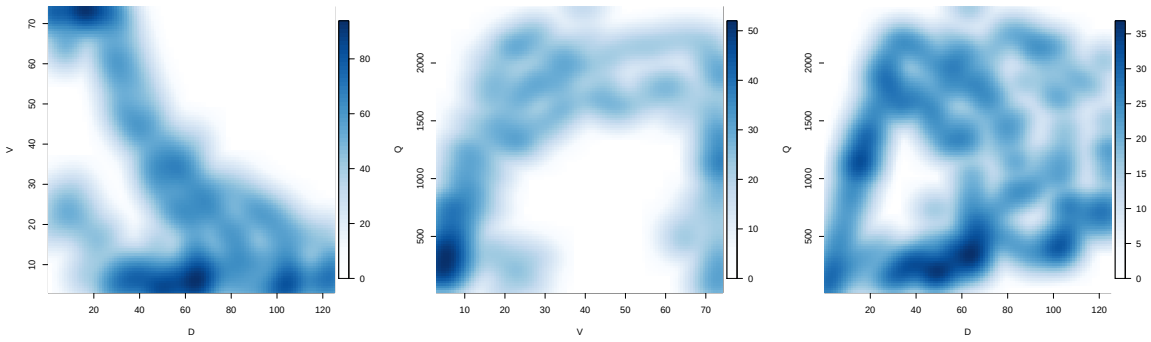
(a) DV (b) VQ (c) DQ

Abbildung 7.16.: Empirische Fundamentaldiagramme bei vollständiger DynLCP 30min-Durchdringung

sehr geringe Dichte, dann können die Fahrzeuge mit hoher Geschwindigkeit fahren; oder sehr viele Fahrzeuge stehen im Stau und die Geschwindigkeit liegt nur knapp über 0km/h. Bei BeeJamA ist diese Dichotomisierung nicht zu beobachten, viel mehr herrschen deutliche höhere Geschwindigkeiten. Bei mittleren Link-Auslastungen ist die maximale Geschwindigkeit nicht erreichbar und die kleineren, helleren Cluster entstehen. Auch bei den VQ-Diagrammen verhält es sich ähnlich. Im LCP-Fall gibt es einen Cluster (links) mit geringer Geschwindigkeit und einen Cluster mit hoher Geschwindigkeit. Im BeeJamA-Fall treten eher hohe Geschwindigkeiten und höhere Flüsse auf. Auch im DQ-Diagramm sind im LCP-Fall wieder zwei Cluster zu sehen: bei niedriger Dichte entsteht ein geringer Fluss, da der Link frei ist; bei hoher Dichte ist der Fluss gering, da der Link stark ausgelastet ist. Im BeeJamA-Diagramm ist der Effekt eines geringen Flusses bei hoher Dichte nicht zu erkennen.

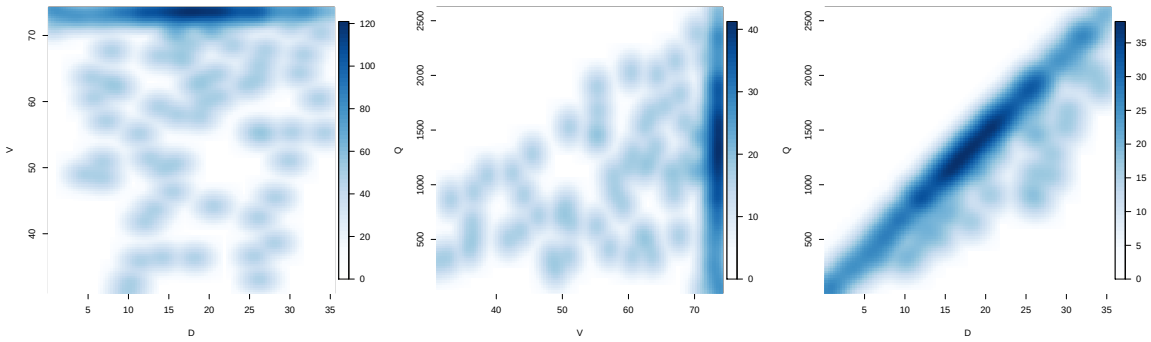
DynLCP 30min und 15min liegen jeweils zwischen diesen beiden Extrema.

Die Abbildung 7.19 verdeutlicht die unterschiedlichen Fahrzeiten bei Ankunft. In den Teilab-



(a) DV (b) VQ (c) DQ

Abbildung 7.17.: Empirische Fundamentaldiagramme bei vollständiger DynLCP 15min-Durchdringung



(a) DV (b) VQ (c) DQ

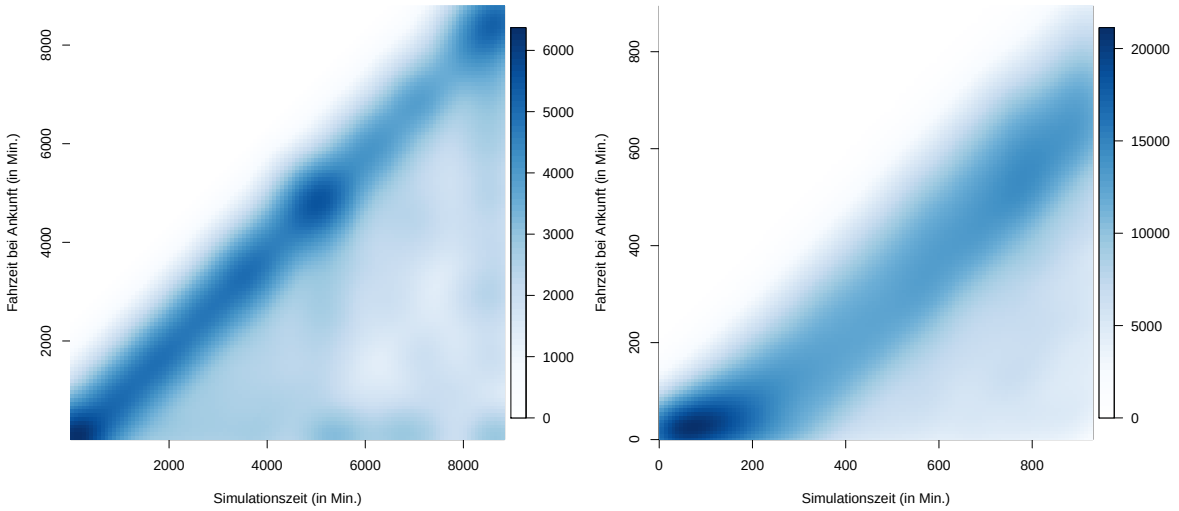
Abbildung 7.18.: Empirische Fundamentaldiagramme bei vollständiger BeeJamA-Durchdringung

bildungen (b)-(d) wurden die gleichen Skalierungen zur besseren Vergleichbarkeit verwendet. Offensichtlich kommen Fahrzeuge im BeeJamA-Fall deutlich früher und schneller ans Ziel.

Die Abbildungen 7.20a und 7.20b verdeutlichen die höhere Agilität des BeeJamA-Protokolls im Vergleich zum DynLCP 5min-Protokoll. Es ist zu erkennen, dass erheblich mehr Forwarding-Entscheidungen bei BeeJamA getroffen werden. Das lässt sich leicht dadurch erklären, dass an jeder Kreuzung, statt nur alle 5min, ein Request gestellt wird. Pro Minute entstehen ca. 22,5 Tausend Anfragen, wovon ca. 7000 pro Minute zu Forwarding-Entscheidungen führen, welche einer Pfadänderung des Fahrzeugs gleichkommen[5]. Unter DynLCP 5min führen nur ca. die Hälfte aller Anfragen zu Änderungen.

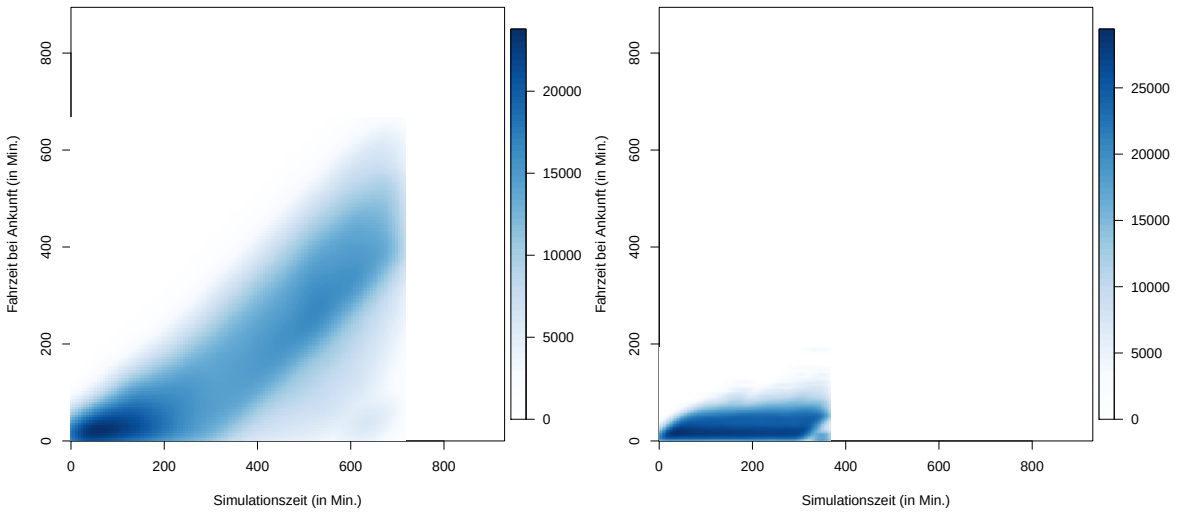
Um die unterschiedlichen Link-Auslastungen genauer vergleichen zu können, sind die Abbildungen 7.21a und 7.21b hilfreich. Dort ist über die Simulationszeit die Auslastung der

[5]Dazu wurde das BeeJamA-Protokoll so abgeändert, dass ganze Pfade (nach dem Reservation Forager-Prinzip) statt nur einem Next Hop bestimmt werden und anschließend der alte und neue Pfad verglichen.



(a) LCP

(b) DynLCP 30min

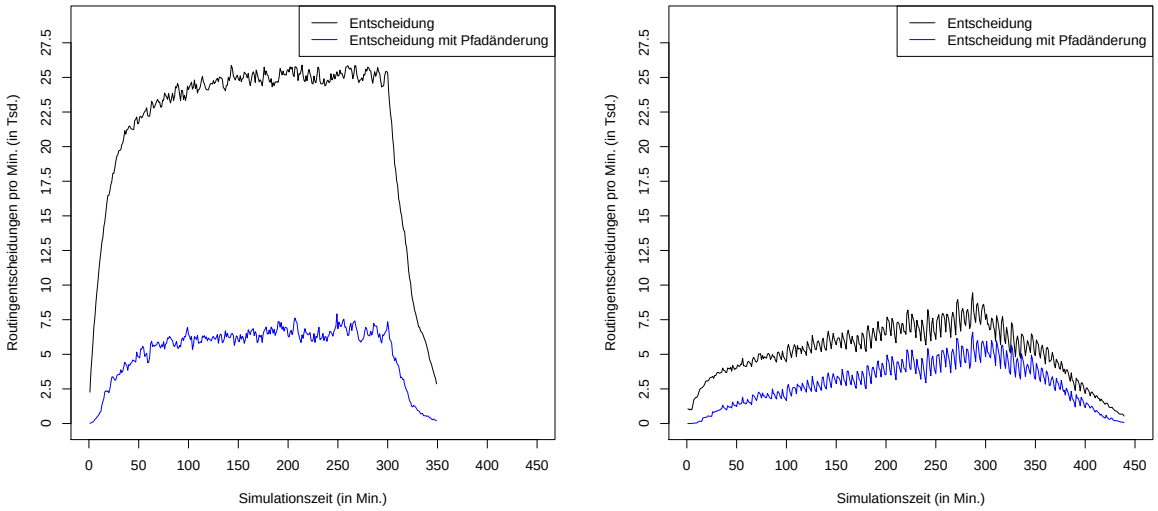


(c) DynLCP 15min

(d) BeeJamA

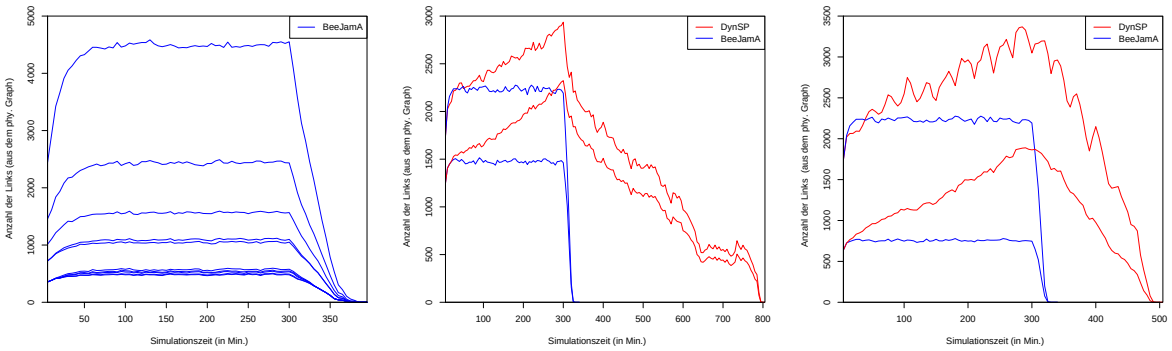
Abbildung 7.19.: Fahrzeiten bei Ankunft

Links einer Simulation mit 300.000 Fahrzeugen aufgetragen (die Protokolle haben jeweils eine vollständige Durchdringung). In erstgenannter Abbildung sind von oben nach unten die Auslastungen in 10%-Schritten einer Simulation dargetan. Bei der obersten Kurve handelt es sich demnach um die Anzahl der Links (des physikalischen Graphen), welche eine Auslastung von mindestens 10% aufweisen. Die darunter liegende Kurve gibt analog die Anzahl der Links mit einer Mindestauslastung von 20% usw. an. Erkennbar ist, dass erstens hohe Auslastungen seltener vorkommen und zweitens, dass die Auslastungen recht konstant sind. Zu Beginn der



(a) BeeJamA (b) DynLCP 5min

Abbildung 7.20.: Anzahl der Forwarding-Entscheidungen



(a) Ruhrgebiet, BeeJamA, 300k Fz (b) Shanghai, DynLCP 5min, 300k Fz (c) Shanghai, DynLCP 30min, 230k Fz

Abbildung 7.21.: Link-Auslastungen

Simulation steigt die Anzahl der Fahrzeuge und damit die Auslastung der Links. Während der Simulation bleibt die Anzahl der Links, die eine gewisse Auslastung aufweisen, nahezu konstant. Zum Ende hin sinkt die Auslastung, bis alle Fahrzeuge ihr Ziel erreicht haben und die Auslastung auf allen Links auf Null absinkt. Eine deutlich andere Situation ergibt sich aus der zweitgenannten Abbildung 7.21b, bei der die Auslastungskurven für 10% und 20% für das Shanghai-Netz dargestellt sind. Neben einer Simulation mit BeeJamA-Fahrzeugen, in blau und mit ähnlichem konstanten Auslastungsverhalten, ist in rot das Auslastungsverhalten einer DynLCP 5min-Simulation, welche nur auf den Links mit minimaler Maximalkapazität von 1200 Fz/h, eine Aktualisierung erhalten. Anstatt weitestgehend konstant zu bleiben, steigt die

Anzahl der ausgelasteten Kanten ungefähr bis 5 Stunden nach Simulationsbeginn (solange werden Fahrzeuge der Simulation hinzugefügt). Anschließend fallen die Auslastungskurven. Deutlich erkennbar ist der Unterschied in dem benötigten Simulationszeitraum. Während in der BeeJamA-Simulation alle Fahrzeuge nach ca. 320 Minuten ihr Ziel erreicht haben, erreichen die letzten Fahrzeuge in der DynLCP 5min-Simulation ihre Ziel erst nach ca. 800 Minuten. Abbildung 7.21c zeigt die Auslastungskurven für 10% und 50%, einer DynLCP 30min-Simulation, welche allerdings auf allen Links Aktualisierungen erhält. Deutlich wird dabei die 30 minütige Aktualisierungsfrequenz anhand der (roten) 10%-Kurve. Alle dreißig Minuten sinkt die Anzahl der mit mindestens 10% ausgelasteten Kurve um bis zu 500. Die 50%-Kurve zeichnet diesen Verlauf deutlich schwächer nach, dafür ist der starke Anstieg, insbesondere im Vergleich zu 50%-BeeJamA-Kurve, deutlich erkennbar. Zum Zeitpunkt 320 Minuten nach Simulationsbeginn weisen etwa 1000 Links mehr eine Mindestauslastung von 50% auf unter DynLCP 30min als unter BeeJamA. Auch ist der Abfall nach dem Höhepunkt erheblich flacher und die Simulation endet ca. 150 Minuten später als mit BeeJamA.

Das BeeJamA-Protokoll führt in den getesteten Szenarien demnach zu deutlich geringeren Auslastungen, wodurch die Staugefahr sinkt.

7.3. Stochastisches Routing

Ferner soll noch das Konzept des suboptimalen stochastischen Routings eruiert werden. Wie in Abschnitt 6.9 angedeutet, sind die möglichen Alternativpfade, wenn nur ein Forwarding zu Knoten mit geringeren Kosten möglich ist, sehr gering. In dem Ruhrgebiet-Netz weisen dann nur 7,09%, in dem Shanghai-Netz 7,21%, aller Knoten überhaupt noch mehr als einen Tabelleneintrag zu ein und demselben Ziel auf. Ein Fahrzeug im Ruhrgebietsnetz passiert im Schnitt 48 Knoten und folglich besteht im Schnitt nur an 3,4 Knoten die Möglichkeit, zwischen mindestens zwei Einträgen zu wählen. Unterschiede konnten simulativ nicht gefunden werden, außer, dass die Fahrzeiten minimal stiegen (an der zweiten Stelle hinter dem Komma). Schaut man sich die Situationen an, in denen alternative Einträge vorhanden sind, stößt man auf Fälle wie in Abbildung 7.22a. Dort abgebildet ist die Situation, in welcher Fahrzeuge von f zu a fahren möchten. Fände eine Weiterleitung stets über den besten Pfad statt, würden die Fahrzeuge zunächst nur über c weitergeleitet. Stiege die Dichte auf dem linken Pfad, würde ggf. die Route über d günstiger und die Fahrzeuge würden auch über d weitergeleitet. Bei stochastischer Weiterleitung hingegen würde direkt zu Beginn ein Teil der Fahrzeuge auch über d geleitet, obwohl ihre Fahrzeit über diesen Weg größer ist als über c. Der Idee nach soll solch eine Auffächerung den Fluss über c reduzieren und so einen Vorteil bringen. Das scheitert aber schon alleine an der Tatsache, dass beide Teilflüsse später wieder den gemeinsamen Link (b, a) nutzen müssen. Aber selbst wenn eine Situation wie in Abbildung 7.22b vorhanden ist, bei der eine Verzweigung mit mindestens zwei disjunkten Pfaden zum Ziel existiert. Dann muss nicht unbedingt eine Verbesserung bzgl. der durchschnittlichen Fahrzeiten eintreten. Denn im Regelfall ist ein Teilgraph nicht so isoliert wie dargestellt. Stattdessen ist auch noch weiterer Zufluss, in Abbildung 7.22c zumindest durch die beiden zusätzlichen Knoten h und i angedeutet, vorhanden. Neben dem Fluss, welcher von f in Richtung a ausgeht, kommt somit zusätzlich noch der Fluss von i ebenfalls in Richtung a. Dadurch steigt unweigerlich die Auslastung auf dem Pfad über d und die Attraktivität an e den Fluss von f über h und d weiterzuleiten, sinkt. Demnach steigt auch wieder die Auslastung über c.

Zusammengefasst wird bei schwacher Auslastung ein Teil des Flusses über suboptimale Pfade

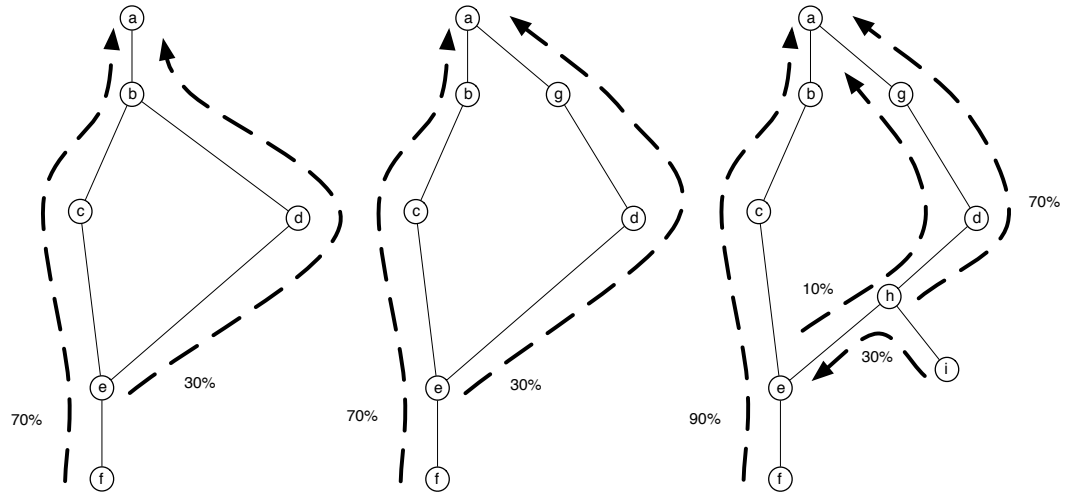


Abbildung 7.22.: Zum Versagen des suboptimalen Routings

weitergeleitet, wodurch diese Fahrzeuge eine höhere Fahrzeit erfahren als nötig. Bei hoher Auslastung, und das ist das getestete Szenario in dieser Arbeit, sinkt die Weiterleitung über alternative Pfade, gleichzeitig mischen sich die Flüsse (in diesem Fall von f und i ausgehend). Insgesamt konnte daher in den Simulationen kein Unterschied festgestellt werden. Schließlich muss noch die Fahrzeug-spezifische Posteriori-Vermeidung evaluiert werden. Der Vorteil gegenüber der A-Priori-Vermeidung ist, dass über jeden Nachfolgeknoten Kosteninformationen vorliegen (sofern Scouts eine Pfad darüber innerhalb ihrer Hop-Reichweite explorieren konnten). Insbesondere solange sich ein Fahrzeug im Routingfall 3 befindet, sind sehr lange Umwege möglich und ein probabilistisches Forwarding ist ein (gewichteter) Random Walk auf einem Großteil des (Routing-)Graphen. Es muss demnach eingeschränkt werden, welche Verschlechterung akzeptiert wird, um nicht beliebig lange Umwege zuzulassen. Es wurde mit unterschiedlichen Akzeptanzen erfolglos versucht, eine Verbesserung zu erreichen.

Werden Verschlechterungen bis zu 5% bei jeder Forwardingentscheidung akzeptiert, veränderten sich die Fahrzeiten nicht auffällig, da nicht viele Alternativen überhaupt betrachtet wurden. Bei höherer Akzeptanz begannen sich aber die Fahrzeiten teils drastisch zu verschlechtern, so dass keinerlei Aussicht besteht, mittels dieses Ansatzes Vorteile zu erzielen. Ursächlich scheint die gedächtnislose Forwarding-Entscheidung zu sein, in Verbindung mit dem Umstand, dass Auslastungen keinerlei Rolle spielen. Gedächtnislos beschreibt, dass bei jeder Forwardingentscheidung aufs Neue Fahrzeuge rein aufgrund der Kosten der alternativen Next Hops weitergeleitet wurden. Fahrzeuge entfernten sich deshalb teilweise deutlich vom Ziel (im Sinne der räumlichen Distanz als auch der Fahrzeit), da die Auswahlwahrscheinlichkeit eines Next Hops, der nur minimal weiter entfernt ist als der „bessere“ Next Hop, nur geringfügig geringer war. Aufgrund der generellen hohen Auslastung des Netzes, stellten sich Fahrzeuge wieder in Staus, welche aber auch noch weiter vom Ziel entfernt waren. Bei einer 50%-Akzeptanz stiegen die durchschnittlichen Fahrzeiten aber schon um 8 Minuten, bei einer akzeptierten Verdoppelung (100%) um ca. 13 Minuten. Die zurückgelegten Distanzen stiegen im ersten Fall um ca. 0.9km, im Zweiten um ca. 1,9km.

Mittels stochastischem Routing ließ sich demnach keinerlei Verbesserung erzielen.

7.4. Reservierungen und Randkostenbepreisung

In Abschnitt 6.11 wurde das Konzept der Reservierung vorgestellt. Abbildung 7.25a zeigt die durchschnittlichen Fahrzeiten des BeeJamA-Basisprotokolls und dem als *ResBeeJamA-N* bezeichneten modifizierten BeeJamA-Protokoll mit naiver Reservierung (d.h. allein basierend auf den Eintragungen im Reservierungslog). Es ist deutlich zu erkennen, dass dieses Konzept keinen Vorteil bringt – mit der einzigen Ausnahme einer vollständigen, 100-prozentigen Durchdringung. Nur in diesem Fall liegt der Durchschnitt niedriger, er fällt von 29,17min auf 27,68min (siehe Tabelle 7.9). Jedoch handelt es sich um eine Durchdringung, die in der Realität kaum auftreten wird. Aber selbst bei einer Durchdringung von 90% ist der Unterschied schon sehr groß. Allein diese zehn Prozent, welche keine Reservierungen vornehmen, führen zu einer Ungenauigkeit, der durch die LPF prognostizierten Transitzeit. Dies führt schlussendlich dazu, dass Fahrzeuge einen Link verwenden, der teurer ist als vorhergesagt. Anders ausgedrückt, die Fahrzeuge verwenden stärker ausgelastete Links als geplant, wodurch sich die Fahrzeiten verlängern.

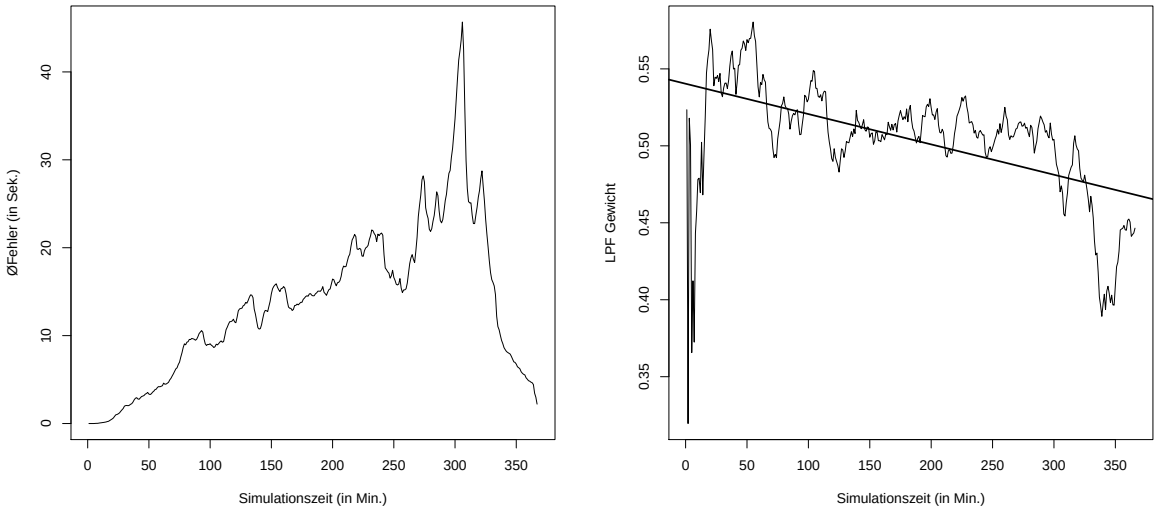
Abbildung 7.26a zeigt die kumulierte Anzahl angekommener Fahrzeuge über den Verlauf der Simulationszeit. Es ist zu erkennen, dass BeeJamA und das mit *ResBeeJamA-DH* bezeichnete BeeJamA-Protokoll mit Hybrid-Reservierung (basierend auf CCP- und PCP-Transitzeiten) alle Fahrzeuge rund 300 Minuten früher zum Ziel bringen als naive Reservierung und das mit *ResBeeJamA-S* bezeichnete Protokoll. Dabei handelt es sich um ein simples statisches Modell zur Prognose der auf einem Link angemeldeten Fahrzeuge. Ist die Reservierungsdurchdringung kleiner als 100%, wird die Anzahl der Fahrzeuge falsch geschätzt. Dieses Modell schätzt die Anzahl der Fahrzeuge einfach mittels linearer Extrapolation. Reserviert z.B. ein Drittel aller Fahrzeuge, wird demnach geschätzt, dass drei mal so viele Fahrzeuge auf dem Link sind als registrierte. Diese einfache Heuristik ist scheinbar deutlich zu ungenau, denn das Ergebnis liegt in der dargestellten, und einigen anderen stichprobenartig durchgeführten, Simulationen jeweils maximal in der Größenordnung des naiven Ansatzes. Auf eine weitere Analyse wurde daher verzichtet.

Wie Tabelle 7.9 zeigt, ist der hybride Ansatz bzgl. der durchschnittlichen Fahrzeiten ab einer Durchdringung von 40% besser als der unmodifizierte Ansatz. Im Falle einer 40 prozentigen Durchdringung mit ca. 4min Unterschied sogar recht deutlich, im Falle einer vollständigen Durchdringung nicht ganz 2,5min. Davor scheint die Schätzung mittels LPF zu ungenau zu sein. Das unmodifizierte Protokoll schneidet dabei besser ab.

In den Abbildungen 7.23a, 7.23b und 7.24 ist der Fehler (Diskrepanz LPF-Prognose und tatsächliche Transitzeit), das LPF-Gewicht und die Anzahl der Fahrzeuge im Netz dargestellt.

Die Abbildungen 7.26b und 7.26c stellen die kumulierte Anzahl von Fahrzeugen bei 30% bzw. 90% Durchdringung über den Simulationsverlauf getrennt nach Protokollen dar. Pro Abbildung sind demnach die Ergebnisse zweier Simulationen im Vergleich dargestellt: Jeweils eine Simulation mit BeeJamA und ResBeeJamA-DH, wobei die Anzahl der angekommen Fahrzeuge aufgesplittet ist nach (Res)BeeJamA(-DH) und DynLCP 30min. Zum Einen ist ersichtlich, dass alle Fahrzeuge im 90%-Fall rund 200 Minuten eher angekommen. Darüber hinaus sind die Unterschiede visuell weniger gut erkennbar. Allerdings erkennt man, dass im 30% Fall in der BeeJamA-Simulation die Fahrzeuge jeweils eher ankommen, als in der ResBeeJamA-DH-Simulation. Im 90%-Fall ist dies umgekehrt, wenn auch weniger deutlich. Dies stimmt mit der Angabe der durchschnittlichen Fahrzeit aus Tabelle 7.9 überein, denn auch dort hat im 30%-Fall BeeJamA einen Vorteil (und umgekehrt im 90%-Fall).

Reservierungen konnten in den getesteten Szenarien zu einer Verbesserung in der Größenordnung



(a) LPF-Prognosefehler (b) LPF-Gewicht

Abbildung 7.23.: LPF-Eigenschaftgen

Tabelle 7.8.: DynSP travel times (in minutes)

	1st Q.	Median	Mean	3rd Q.	Max
DynSP 10min	27.28	67.50	100.66	161.78	497.68
DynResSP 10min	15.18	25.08	42.48	39.93	438.43

von ca. 10% beitragen (im Vergleich zum Basisprotokoll).

Auswirkungen des Marginal Cost Pricing Zusätzlich sei noch das Konzept des Marginal Cost Princing betrachtet. Ergebnisse für das ResBeeJamA-DH mit MCP findet sich in der Tabelle 7.10. Ein positiver Effekt ließ sich nicht feststellen. Auch für das unmodifizierte BeeJamA mit MCP, siehe Tabelle 7.11, ließ sich dieser nicht feststellen. Allerdings ist, je höher die Durchdringung wird, der Abstand zu den durchschnittlichen Fahrzeiten des ResBeeJamA-DH immer kleiner. Ist aber bspw. die Nachrichtenkomplexität ein relevanter Faktor, dann kann BeeJamA mit MCP eventuell eine Alternative sein, da die Downstream-Scouts und Forager eingespart würden (im Vergleich zum Basisprotokoll).

Insgesamt betrachtet, kann das Konzept der Reservierung demnach durchaus Vorteile bringen, allerdings erst, wenn die Durchdringung hoch genug ist. Allerdings scheint naive Reservierung dem hybriden Konzept unterlegen zu sein, genau wie MCP, das zumindest bezüglich der Fahrzeiten keinen Vorteil in den Simulationen brachte.

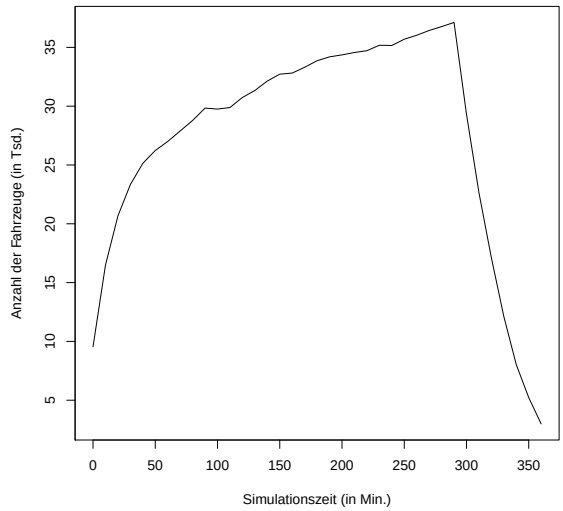


Abbildung 7.24.: BeeJamA vs. ResBeeJamA-N

7.5. BeeJamA-Protokoll mit mehreren Kriterien

Dieser Abschnitt evaluiert die Erweiterung zur Weiterleitung nach mehreren Kriterien. Als weiteres Kriterium wird in dieser Arbeit nur die Auslastung der Links betrachtet. Zwar spielen im Straßenverkehr auch Distanz und Treibstoffverbrauch eine Rolle, jedoch liegt der Fokus in dieser Arbeit auf der Fahrzeit. Distanz an sich ist eher wenig geeignet, diese zu reduzieren. Treibstoffverbrauch ist zwar ein wichtiges Thema, aber allein aufgrund des Simulators nicht gut analysierbar, da bspw. keine Beschleunigungen, sondern nur durchschnittliche Geschwindigkeit, simuliert werden. Zudem korreliert Treibstoffverbrauch auch stark mit der Fahrzeit.

Zunächst wurde der simple Ansatz der gewichteten Kostensumme getestet. Die Kosten eines Links entsprechen nicht mehr nur der Transitzeit, sondern werden wie folgt bestimmt: $c_{ij} = w_1\tau_{ij} + w_2\tau_{ij}L_{ij}$, wobei τ_{ij} der Transitzeit des Links (i, j) und L_{ij} dessen Auslastung bezeichnet, mit $w_1 + w_2 = 1$. Für $w_1 = 1$ (und somit komplementär $w_2 = 0$) entsprechen die Kosten der Transitzeit. Sinkt w_1, gewinnt die Auslastung an Bedeutung. Ist ein Link mit einer hoher Transitzeit stark ausgelastet, sind die resultierenden Kosten höher als auf einem ähnlich ausgelastetem Link mit geringerer Transitzeit. Die Ergebnisse bzgl. der Fahrzeit sind in Abbildung 7.27 dargestellt. Die angegebenen Gewichte entsprechen w_1, folglich zeigt der linke Boxplot die Verteilung der Fahrzeiten einer Simulation, bei der die Kosten ausschließlich den Transitkosten entsprechen. Alle getesteten Gewichtungen führten zu schlechteren Ergebnissen, beispielhaft sind Boxplots für $w_2 = 0,25$ und $w_2 = 0.5$ abgegeben. Je höher die Bedeutung der Auslastung wurde, desto schlechter wurden die Ergebnisse. Ferner wurden auch Simulationen durchgeführt, in der die Länge und die Differenz zwischen FFT und Transitzeit (jeweils entweder zusätzlich zur Auslastung oder auch exklusiv) als zusätzlicher Term in die obige Summe mit einging. In keiner Kombination konnten positive Effekte festgestellt werden. Nicht-individuelle Gewichtungen führen an dieser Stelle offensichtlich zu unkoordinierten, kontraproduktiven Flussumleitungen.

Zusammen mit der Auslastung soll aber auch folgendes Szenario geprüft werden: Kann die

Tabelle 7.9.: Auswirkungen von Reservierungen (Fahrzeiten in Minuten)

		ResBeeJamA	
Durchdringung	BeeJamA	N	DH
0%	257,74	257,74	257,74
10%	155,32	211,91	160,46
20%	91,02	159,10	100,49
30%	58,44	134,29	69,03
40%	45,00	123,15	40,91
50%	32,78	94,44	31,65
60%	30,00	81,77	29,04
70%	28,02	78,78	26,91
80%	28,96	76,95	26,12
90%	29,18	72,23	25,92
100%	29,17	27,68	26,53

Tabelle 7.10.: Auswirkungen von MCP (Fahrzeiten in Minuten)

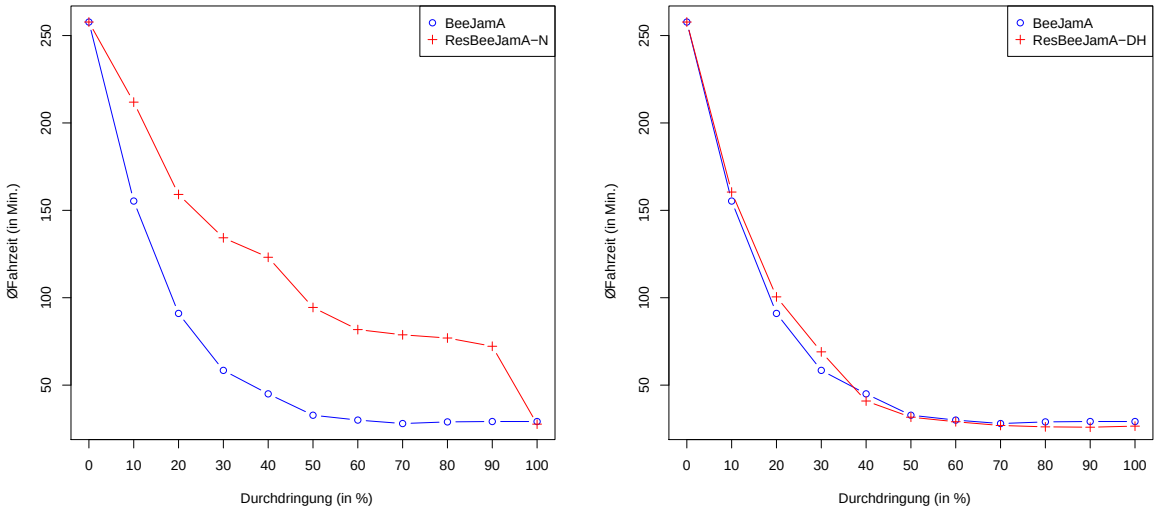
Durchdringung	Protokoll	1. Quartil	Median	Durchschnitt	3. Quartil	Max
30%	Res	22,37	52,98	69,03	104,71	219,02
30%	ResMCP	23,07	54,91	72,76	106,60	221,09
100%	Res	15,27	24,37	26,53	36,12	104,58
100%	ResMCP	15,55	25,07	27.38	37,75	105,05

Mehrheit Vorteile daraus ziehen, wenn eine Minderheit freiwillig Umwege (bzgl. der Fahrzeit) in Kauf nimmt. Dazu wird eine Teilmenge der Fahrer sich wie folgt verhalten: Wähle den günstigsten Pfadknoten zum Ziel, dessen gemittelte Auslastung nicht über einer gewissen Schwelle liegt. Ist solche eine Alternative nicht vorhanden, wähle den ELCP-Next Hop.

Diese Next Hop-Selektionsstrategie ist nur durch die Verwendung eines zusätzlichen Scout-typs möglich, der gemäß des Disseminationskriteriums der gewichteten Auslastung propagiert wird, gleichzeitig aber auch die Transitzeit aggregiert.

Bezweckt werden soll damit, dass einige Fahrer Umwege akzeptieren, die noch nicht zu stark ausgelastet sind. Zwar benötigen diese Fahrzeuge dann unter Umständen länger bis zu ihrem Ziel, dafür sinkt die Auslastung auf den stark frequentierten Hauptrouten. Insgesamt könnte dadurch insbesondere für die übrigen Fahrer ein Fahrzeitvorteil entstehen.

Das Ergebnis der zugehörigen Simulationen ist in den Abbildungen 7.28a und 7.28b zu sehen. In der erstgenannten Abbildung sind die Fahrzeiten der Fahrzeuge zu sehen, welche gemäß des unmodifizierten BeeJamA-Protokolls weiterhin den LCP verwenden. In der zweiten Abbildung sind die durchschnittlichen Fahrzeiten der Fahrzeuge zu sehen, welche gemäß obiger



(a) BeeJamA vs. ResBeeJamA-N

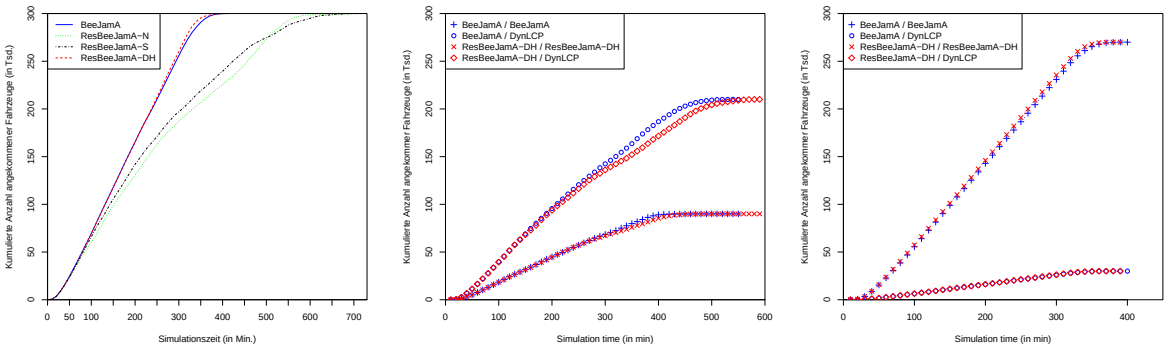
(b) BeeJamA vs. ResBeeJamA-DH

Abbildung 7.25.: BeeJamA vs. ResBeeJamA-Varianten

Vereinbarung Umwege akzeptieren. In der Tat ist eine Verbesserung für die Mehrheit (DynLCP) festzustellen, sobald eine Minderheit (BeeJamA) Umwege akzeptiert. Besonders auffällig ist der Effekt bei einer geringen Durchdringung (10-20%). Dabei wählt die Mehrheit den LCP (gemäß DynLCP-Protokoll) und eine Minderheit weicht auf unausgelastete Strecken aus.

Es sind zwei Effekte beobachtbar. Zum Einen erzielt eine Akzeptanz von 20% die größte Veränderung. Zum Anderen, je größer die Durchdrinung der BeeJamA-Fahrzeuge wird (d.h. je größer die Umwege-akzeptierende „Minderheit“ wird), desto geringer fällt zwar diese Verändeurng aus, aber dennoch fallen auch die Fahrzeiten. Bei einer Akzeptanz von 0% Auslastung werden nur alternative Next Hops akzeptiert, welche auf einem vollständig freien Pfad liegen. Da diese, außer zu Simulationsbeginn, quasi nicht vorhanden sind, greift die Regelung, dass ansonsten der ELCP-Knoten gemäß BeeJamA-Protokoll gewählt wird. Folglich ist die durchschnittliche Fahrzeit sehr ähnlich zu dem Fall der Akzeptanz von 100 prozentig ausgelasteter Pfade (bzw. von Next Hops auf diesen Pfaden). Akzeptiert die Minderheit jedoch auch Pfade mit 20 prozentiger Auslastung, so reduzieren sich die Fahrzeiten wie angegeben, da die ansonsten stärker ausgelasteten Strecken entlastet werden. Steigt die Bereitschaft der Minderheit, stärker ausgelastete Strecken zu nutzen, so nähert sich das dem „normalen“ Verhalten an, stets die LCP zu wählen. Bei 100% Auslastungsakzeptanz schließlich ist die Auslastung irrelevant und das so modifizierte BeeJamA-Protokoll verhält sich wie das normale BeeJamA-Protokoll. Wie bspw. aus Abbildung 7.21a ersichtlich, weisen im Laufe einer Simulation nur wenige Links eine deutlich höhere Auslastung als 20-30% auf, was mit diesem Ergebnis korrespondiert. Denn akzeptiert die Minderheit größere Auslastungen, liegen die „direkteren“, aber höher ausgelasteten, Strecken im akzeptablen Bereich.

Steigert man die Durchdringung des modifizierten BeeJamA-Protokolls, nimmt der Effekt allerdings ab. Immer mehr Fahrzeuge sind dazu bereit, Umwege zu akzeptieren und sind dabei auch gleichzeitig agiler, um so die eingeschlagene Route noch ändern zu können. Durch



(a) 50% Durchdringung

(b) 30% Durchdringung: BeeJamA vs. ResBeeJamA-DH

(c) 90% Durchdringung: BeeJamA vs. ResBeeJamA-DH

Abbildung 7.26.: Auswirkungen unterschiedlicher Durchdringungen

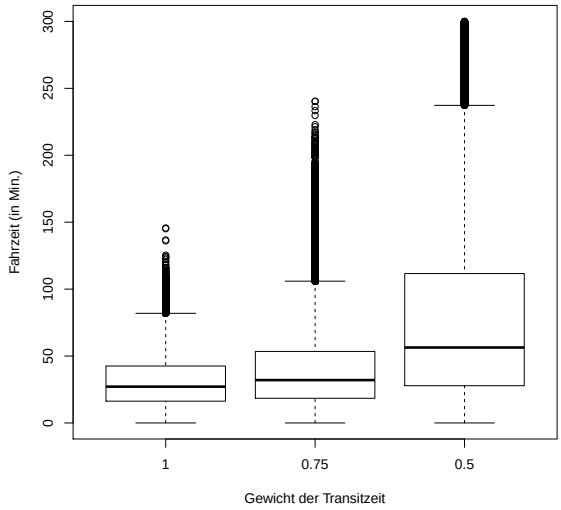


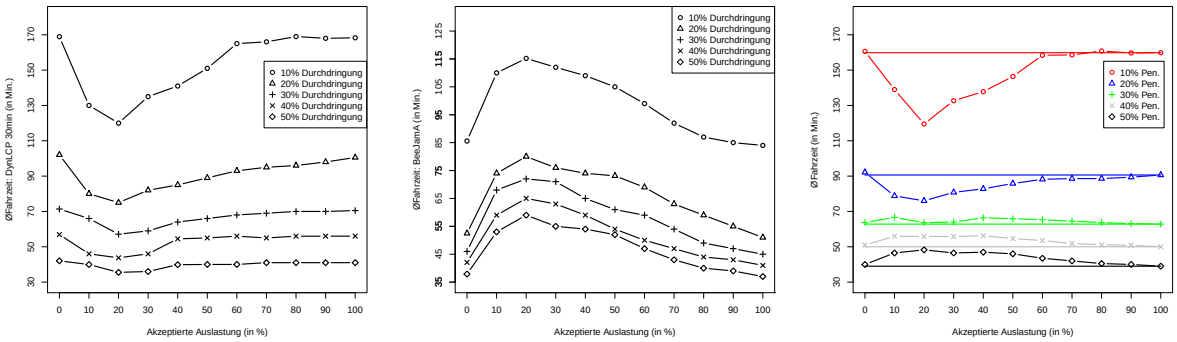
Abbildung 7.27.: Boxplots zur gewichteten Kostensumme

die erhöhte Agilität verteilen sich die BeeJamA-Fahrzeuge früher und gleichmäßiger über das Netz, wodurch eine gleichmäßigere, aber auch höhere mittlere Auslastung auf sonst eher weniger ausgelasteten Links vorherrscht. Eine höhere Auslastungsakzeptanz wäre somit notwendig, damit Fahrzeuge alternative Routen nutzen. Das wiederum erhöht allerdings auch die Wahrscheinlichkeit, dass der ELCP unterhalb der Akzeptanzschwelle liegt. Der Effekt des „freiwilligen“ Umwegs verflacht bei höherer BeeJamA-Durchdringung zusehends. Wie zu erkennen, sind bei einer Durchdringung von 50% die durchschnittlichen Fahrzeiten geringer als bei niedrigeren Durchdringungen. Der „Umwege“-Effekt hingegen verschwindet fast vollständig.

Abbildung 7.28b zeigt hingegen die durchschnittlichen Fahrzeiten der BeeJamA-Fahrzeuge. Wenig überraschend sind bei 20%-Auslastungsakzeptanz die Fahrzeiten maximal. Auffällig hingegen ist, dass selbst bei 50-prozentiger BeeJamA-Durchdringung und bei einer Auslastungs-

Tabelle 7.11.: Auswirkungen von MCP (ØFahrzeiten in Minuten)

Durchdringung	BeeJamA	+MCP	+Res
0%	257,74	257,74	257,74
10%	155,32	168,21	160,46
20%	91,02	104,73	100,48
30%	58,44	72,04	69,03
40%	45,00	41,17	40,91
50%	32,78	32,01	31,65
60%	30,00	29,47	29,04
70%	28,02	27,30	26,91
80%	28,96	26,58	26,12
90%	29,18	26,31	25,92
100%	29,17	26,92	26,53



(a) DynLCP 30min-Fahrzeuge (b) BeeJamA-Fahrzeuge (c) Alle Fahrzeuge

Abbildung 7.28.: Durchschnittliche Fahrzeiten des bikriteriellen Routings

akzeptanz von 20%, die BeeJamA-Fahrzeiten noch deutlich erhöht sind. Diese Phänomen muss in Folgearbeiten noch weiter untersucht werden, aber vermutlich sorgt die gleichmäßigere Ausbreitung und schnellere Ankunft der BeeJamA-Fahrzeuge dafür, dass die mittlere Auslastung genügend vieler Pfade im Akzeptanzbereich liegt. In Summe fahren viele BeeJamA-Fahrzeuge entsprechend Umwege und die DynLCP-Fahrzeuge profitieren von freieren Hauptrouten. Dadurch steigt die Fahrzeit der BeeJamA-Fahrzeuge und die DynLCP-Fahrzeuge können kaum noch schneller ans Ziel kommen.

Zusammenfassend erscheint es als angebracht, bei kleineren BeeJamA-Durchdringungen eine gewisse Auslastungsakzeptanz zuzulassen und diese bei höheren Durchdringungen vollständig zu reduzieren. In Abbildung 7.28c sind die durchschnittlichen Gesamtfahrzeiten (der BeeJamA- und DynLCP 30min-Fahrzeuge) dargestellt. Gleichsam kennzeichnen die horizontalen Geraden die Fahrzeit, welche das Basisprotokoll erzielen würden (entsprechend der Fahrzeit bei 100%

Auslastungsakzeptanz). Im Falle einer 10 und 20 prozentigen Durchdringung des BeeJamA-Protokolls mit Auslastungsakzeptanz ist sogar ein globaler Vorteil vorhanden. Bei höheren Auslastungen stellt sich hingegen ein kontraproduktiver Effekt ein.

Ist eine Minderheit bereit Umwege zu akzeptieren, ist somit eine globale Verbesserung zu beobachten.

7.6. Zusammenfassung und Diskussion

Mit der Evaluation dieses Kapitels ist die Beschreibung des BeeJamA-Protokolls abgeschlossen. Es wurde empirisch gezeigt:

- Das BeeJamA-Basisprotokoll ist kompetitiv und erzeugt deutlich bessere Fahrzeiten als DynLCP 5min.
- Die Anzahl der benötigten Nachrichten (und teilweise die Nachrichtenlänge) ist um Größenordnungen geringer als in Bellman-Ford-basierten Ansätzen. Auch hat die Nachfrage keinen Einfluss auf die Anzahl der Nachrichten.
- Die Hierarchisierung reduziert die Nachrichtenanzahl und bietet ein vertretbares Maß an Fahrzeitenverschlechterung.
- Zunehmende Fahrtstrecken unter BeeJamA konnten nicht beobachtet werden.
- Die deutlich höhere Agilität führt zu besseren durchschnittlichen und maximalen Fahrzeiten und die Link-Auslastungen sind geringer als die getesteten Konkurrenz-Protokolle, anders ausgedrückt: es entsteht weniger Stau.
- Stochastisches Forwarding scheint nicht von Vorteil zu sein.
- Reservierungen können die globalen Fahrzeiten weiter reduzieren.
- Marginal Cost Pricing kann in dynamischen Situationen keine Vorteile im Vergleich zu der hybriden Reservierung erzielen. Jedoch kann MCP als Zusatz zu dem Basisprotokoll bei hohen Durchdringeungen fast die hybride Reservierung ersetzen, ohne jedoch zusätzliche Agententypen zu benötigen.
- Das spezielle bikriterielle Routing kann dazu genutzt werden, globale Fahrzeitverbesserungen zu erzielen, wenn eine gewissen Minderheit bereit ist, Umwege zu akzeptieren.

Insgesamt ist festzuhalten, dass in den durchgeführten Simulationen die BeeJamA-Varianten bzgl. der Fahrzeiten besser abschnitten und weniger Nachrichten erzeugten als andere verteilte Ansätze. Daraus folgt eine höhere Skalierbarkeit ohne Verschlechterung für die Teilnehmer. Verbesserungen für die Allgemeinehit stellen sich auch schon ein, wenn nur ein kleiner Teil der Fahrzeuge mit BeeJamA ausgestattet ist. Ab ca. 30-40% sind dann erhebliche Unterschiede in den durchgeführten Simulationen festzustellen.

Letztendlich bleiben zwei Fragen durch die durchgeführten Simulationen unbeantwortet:

1. *Für welche Netze und ab welchen Auslastungen ist höhere Agilität kontraproduktiv?* Wie in Abschnitt 3.5 dargestellt, existieren Eingabeinstanzen, in denen BeeJamA schlechter abschneidet als weniger agile Protokolle. Neben dem Ruhrgebiet und Shanghai wurde

andere reale Netze stichprobenartig auf dieses Phänomen hin überprüft, positiverweise erfolglos. In *realistischen* Netzen scheint das Problem daher augenscheinlich nicht auftreten, in synthetischen Netzen, wie einem Grid schon. Es ist derzeit unklar, ab wann genau die Verkehrsnachfrage so groß wird, dass ein bestimmtes Netz zu wenige unausgelastete Alternativpfade aufweist, um Protokollen mit höherer Agilität einen Vorteil zu bieten.

2. *Was lassen diese Ergebnisse für eine reale Umsetzung erwarten?* Zwar existieren Studien, die Ergebnisse eines Simulators mit realen Daten vergleichen, jedoch handelt es sich dabei häufig um kleine Netze und der Fokus liegt eher auf der Synchronität weniger Fahrzeuge in Simulation und Realität. Für große Netze unter Berücksichtigung der Auswirkungen eines VRGS, existieren solche Studien bisher nicht. So ist nicht unmittelbar abschätzbar, in wie weit sich diese positiven Simulationsergebnisse vollends in der Realität bestätigen lassen. An den teils sehr unrealistischen Fahrzeiten ist aber bereits zu erkennen, dass sich in der Realität deutliche Unterschiede ergeben werden. Dennoch ist insgesamt von einem Gewinn auszugehen, da BeeJamA auch bessere Ergebnisse liefert als zentrale LCP-Protokolle mit hoher Agilität.

Die Simulationsergebnisse zusammenfassend ist schlussendlich ein gutes Abschneiden des BeeJamA-Protokolls festzuhalten. Dies lässt für reale Umgebungen hoffen, ebenfalls zu einer Reduzierung der Fahrzeiten beitragen zu können. Das nächste und gleichzeitig letzte Kapitel fasst die Arbeit zusammen und gibt einen Ausblick.

Kapitel 8

Fazit

In diesem letzten Kapitel soll im folgenden Abschnitt 8.1 zunächst ein Ausblick in Hinsicht potentieller Erweiterungen des BeeJamA-Protokolls gegeben werden, aber auch zwei angrenzende Anwendungsgebiete mit Verknüpfungspunkten diskutiert werden. Anschließend wird in Abschnitt 8.2 eine Zusammenfassung und abschließend in Abschnitt 8.3 ein Zielabgleich sowie ein Fazit formuliert.

8.1. Ausblick

Im Folgenden seien einige mögliche Erweiterungen für das BeeJamA-Protokoll dargelegt:

Agentenreduktion: Je größer das abgedeckte Gebiet, desto mehr Agenten müssen eingesetzt werden. Bei dem bisherigen Modell startet jeder Knoten in jeder Generation Agenten, obwohl dies eventuell gar nicht notwendig ist. Agenten breiten sich vom Ursprung o radial, d.h. in alle Richtungen gleichsam, aus. Vielleicht kommen aber gegenwärtig nur Anfragen aus dem Süden, so dass in nördlicher Richtung eigentlich kein Bedarf für aktualisierte Routeninformationen besteht. Mit hoher Wahrscheinlichkeit werden die anfragenden Fahrzeuge aus dem Süden gar nicht in die Region nördlich von o geleitet, weil die günstigen Routen dort nicht herführen (aus südlicher Sicht betrachtet). Die Scouts explorieren dennoch überflüssigerweise die Möglichkeit günstiger Routen durch die nördliche Region. Diese „Einsparung“ ist hingegen nicht unmittelbar vorzunehmen, da a priori nicht bekannt ist, woher die günstigen Routen führen. Selbst wenn aus dem Norden gegenwärtig keine Anfragen kommen, ist es eventuell für die Fahrzeuge aus dem Süden sinnvoll, erst etwas abseits am Ziel vorbeizufahren („zu weit fahren“). Völlig aufgegeben werden darf die Exploration dieser Gegend folglich nicht. Eine sinnvolle Abwägung könnte sein, dass die Scouts sich an den Erfahrungen der vorherigen Generationen orientieren. Dies könnte mittels der (Upstream-)Forager des Reservierungprotokolls umgesetzt werden. Fahrzeuge aus dem Norden senden in diesem Fall ständig Forager zu Reservierung von Pfaden. Anhand dieser Reservierungen ist für die Scouts erkennbar, wo der gegenwärtig günstigste Pfad verläuft. So können Areas, durch die Reservierungen zu einem Zielknoten verlaufen (sowie angrenzende Areas) (FZ- und FR-)Scouts des Ziels immer weiterleiten, andere hingegen nur mit geringerer Frequenz. So entstünde

ein selbstverstärkendes System, in dem besonders gefragte und lohnenswerte Gebiete gründlicher exploriert würden.

Selbstorganisierende Parametrisierung: Das BeeJamA-Protokoll kann als eine Art Baukasten betrachtet werden. Es gibt eine Vielzahl von qualitativen und quantitativen Parametern, welcher in dieser Arbeit festgesetzt wurden, z.B. die Anzahl (und die Form) der Areas auf der Bereichsschicht. Zu Evaluationszwecken wurde in Kapitel 7 eine sehr große Anzahl von Areas gewählt, um die Flexibilität zu verdeutlichen. Je größer die Bereiche dabei werden (und desto weniger Areas deshalb entstehen), desto weniger Agenten müssen (über das Kommunikationsnetz zwischen den Bereichen) versendet werden. Doch welche Area-Aufteilung gewählt werden soll und wie weit die Level-Reichweiten auf der Netzschicht sein sollen, ist bisher manuell festzusetzen. Für einen praktischen Einsatz könnte vorab die Netzbelastung simuliert werden und die Parameter entsprechend gewählt werden. Selbstorganisierend würde in diesem Fall bedeuten, dass die Parameter zur Laufzeit selbstständig eruiert und justiert werden. Nächtens mögen vielleicht weniger Rechnerkapazitäten notwendig sein, da weniger Routinganfragen gestellt werden. So können in dieser Situation weniger Navigator-Cloudprozesse ausreichen, um das gesamte Gebiet abzudecken. All diese Parameter sind in der gegenwärtigen Umsetzung noch fix und werden vor Simulationsbeginn festgelegt. Eine dynamische Anpassung an die realistische Nachfrage unter den gegebenen Rahmenbedingungen wäre sinnvoll. Auch die diskutierten Routingoperationen stellen (qualitative) Parameter dar, die zur Laufzeit passend ausgetauscht und/oder angepasst werden könnten.

Berücksichtigung der Fahrgeschwindigkeit: Zur Vermeidung von Überlast kann auch eine gezielte Verlangsamung des Verkehrs beitragen. Wird von einem (dräuenden) Stau ausgehend in Upstream-Richtung die makroskopische Geschwindigkeit gesenkt, kann zusätzlicher Stau ggf. reduziert werden. Es existieren heute bereits Wechselverkehrsschilder auf Autobahnen, die, dynamisch an die Verkehrssituation angepasst, eine Geschwindigkeit empfehlen bzw. vorschreiben. Diese bisher zentral berechneten Empfehlungen, könnten in dezentraler Weise in das BeeJamA-Protokoll integriert werden.

Pfadverhandlungen: BeeJamA sieht bisher im Wesentlichen ein Selfish-Verhalten der Fahrzeuge vor, welche jeweils mit gleicher Priorität Links nutzen dürfen. In diesem Sinne wird eine First Come First Served-Strategie umgesetzt. Es wäre aber auch denkbar, dass Strecken in rundenbasierten Verhandlungen versteigert werden. Besonders beliebte (weil schnelle) Pfade könnten so nur von den höchstbietenden befahren oder reserviert werden. Aufgrund der Schnelligkeit dieser dynamischen Verhandlungen können die Verhandlungen nicht von den Fahrern selbst durchgeführt werden. Stattdessen werden Verhandlungsagenten benötigt, welche an Stelle der Fahrer mit individuellen Strategien Pfade aushandeln. Solch ein Rundenkonzept griffe die Idee des Frank-Wolfe-Algorithmus auf, welcher in der ersten Iteration für jedes Fahrzeug den (E)LCP wählt (wie das Basisprotokoll). In den folgenden Iterationen wird dann der Fluss in Richtung UE oder SO zwischen den Links verschoben. Entscheidend, ob und falls ja, wohin, Fluss verschoben wird, ist der potentielle Vorteil in Bezug auf die Fahrzeit. Die notwendigen Informationen über Auslastungen können mittels des geschilderten Scout/Forager-Verhaltens zur Reservierung verbreitet werden.

Je dringlicher der Wunsch, eine möglichst gute Strecke reservieren zu können, desto mehr finanzielle Mittel würde ein Fahrer aufbringen, um eine Verhandlung zu „gewinnen“. Je

beliebter eine Strecke gegenwärtig ist (und damit Überlast droht), desto höher stiegen die Preise, wodurch die Attraktivität (und damit die Staugefahr) weiter sänke. Es besteht natürlich die Gefahr, dass finanzstarke Fahrer stets sehr gute Strecken ersteigern können. Für andere Fahrer blieben dann nur weniger gute Strecken. Das Konzept der Gleichberechtigung wäre dadurch aufgehoben, es ist zu eruieren, in wie fern so etwas von Verkehrsteilnehmern im Allgemeinen gewünscht und politisch gewollt ist. Praktisch ist es aber nur eine dynamische Variante bereits eingesetzter dichotomer Mautkonzepte. Besonders „gute“ Strecken (in der Regel Autobahnen), erfordern einen finanziellen Einsatz, „schlechtere“ Strecken (z.B. Landstraßen) hingegen nicht.

Extrapolation historischer Daten: Bisher werden nur bereits eingetretene Ereignisse (Linkpassagen, getätigte Reservierungen) bei der Bewertung von Linkkosten berücksichtigt. Wenn aber bspw. regelmäßig an einem bestimmten Ort Großveranstaltungen stattfinden und dies bekanntermaßen zu einer gewissen Auslastung um den Veranstaltungsort führt (z.B. bei Fussballenspielen um das Stadion), kann mit entsprechend hoher Wahrscheinlichkeit abgeleitet werden, dass bei zukünftigen Ereignissen dieser Art ebenfalls entsprechende Auslastungen auftreten werden. Aus Satellitenstädten fahren werktags morgens auch üblicherweise viele Fahrer in das Zentrum. Ein Wissen, dass genauso in die Bepreisung miteinbezogen werden könnte, wie vorhandene (anonyme) Fahrtprofile.

Reale Umsetzung: So sehr Simulationen zu Beginn einer Entwicklung hilfreich sind, eine reale Umsetzung können sie nicht ersetzen. Eine Realisierung des BeeJamA-Protokolls erfordert prinzipiell keinen übermäßigen Aufwand. Mittels Smartphone lässt sich leicht ein BeeJamA-fähiger Personal Navigation Assistent (PNA) entwickeln und die Navigatoren lassen sich kostengünstig als Cloudprozess umsetzen. Am schwierigsten ist gegenwärtig die Beschaffung von Linkkosten. In der ersten Stufe können Floating Car Data-Informationen der BeeJamA-fähigen PNAs CCP-Kosten liefern. Gleichzeitig, und dies ließe sich nur noch mit Kooperationspartnern bewerkstelligen, sollten auch Informationen aus Sensoren (Induktionsschleifen, Videoüberwachung) und Phone Velocity Data (PVD) einfließen. Am sinnvollsten wäre die sofortige dezentrale Nutzung der Daten, statt der vorherigen zentralen – und somit Flaschenhals-gefährdeten – Sammlung. So fallen PVD dezentral im GSM-Netz an, die direkt vor Ort an den jeweiligen Navigator übergeben werden könnten. Analog dazu können die vorhandenen kommunalen Sensoren ihre Daten unmittelbar an die lokalen Navigatoren übermitteln, statt sie erst in einem Traffic Information Center zentral zu akkumulieren. Das greift dem zukünftig zu erwartenden Aufkommen von Vehicle-2-Infrastructure-Architekturen voraus.

Neben reinen Erweiterungen des BeeJamA-Protokolls, seien noch zwei Anwendungsgebiete genannt, die im weitesten Sinne mit der Thematik in Verbindung gebracht werden können:

Vehicle Routing Problem: Bei dem Vehicle Routing Problem (VRP) [143] handelt es sich, trotz des Namens, um ein kombinatorisches Optimierungsproblem, nicht um eine VRGS-Problemstellung. Statt einer Lösung für ein Flussproblem wird eine (Link-)kostenminimale Depot-Permutation gesucht, in welcher (Lastkraft-)Fahrzeuge der Reihe nach eine bestimmte Teilmenge von Knoten (die Depots) besuchen müssen. Es handelt sich um ein typisches Problem der Transportlogistik. In der Regel muss das Problem aber nicht verteilt gelöst werden, denn Disponenten-Entscheidungen müssen nicht so häufig getroffen werden wie bspw. Routingentscheidungen. Zudem gilt es, eine erhebliche

Anzahl von Nebenbedingungen zu berücksichtigen (gesetzliche Ruhepausen, Art des Fahrzeuges, rechtzeitige Beendigung der Tour uvm.). Für typische Speditionen wird das Sammeln der dynamischen Informationen aller Fahrzeuge vermutlich zu keinem zentralen Flaschenhals führen, da nur die Positionen und die Fahrzeiten zu den Depots relevant sind (wobei letztere Informationen durchaus aus einem separaten dezentralen VRGS stammen können). Sind diese Informationen gesammelt, können spezialisierte Algorithmen (oder Heuristiken) eine Lösung bestimmen und den Fahrern entsprechende Anweisungen geben.

Sollte dennoch eine verteilte Lösung angestrebt sein, ist BeeJamA in der gegenwärtigen Form nicht unmittelbar einsetzbar. Denn aus ELCPs lässt sich keine Depot-Permutation ableiten, stattdessen muss dies kooperativ zwischen den Fahrzeugen geschehen. Die zuvor erwähnte BeeJamA-Erweiterung zur iterativen Pfadverhandlung hingegen wäre ein sinnvoller Ausgangspunkt für eine kooperative Verhandlung zwischen den Fahrzeugen zur Aufteilung der abzudeckenden Routen.

Stromnetzstabilität: Ein Problemfeld, in dem verteilte Ansätze äußerst sinnvoll eingesetzt werden können, ist die Stabilität von Stromnetzen [58]. Fluktuationen bei der Erzeugung und dem Verbrauch von Strom, insb. in Zeiten von regenerativen Energien, führen bisweilen zur Ausreizung der Netzkapazitäten. Stromführende Kabel, besonders Hochspannungs-Überlandleitungen, sind nur für bestimmte Lastflüsse ausgelegt, bei deren Überschreitung es durch Schäden an der Infrastruktur zu Kurzschlüssen kommen kann. Mittels sogenannter Power Flow Controller (PFC) können bei drohender Überlast bestimmte Leitungen in einem Netz entlastet werden, indem Fluss auf parallele Leitungen umgeleitet wird.

Die Einstellungen der PFC-Transformatoren wird heutzutage mitunter noch manuell durchgeführt, nachdem optimale Netzzustände (Optimal Power Flow) zentral durch ein konvexes Optimierungsprogramm berechnet wurden. Diese Berechnungen benötigen für das deutsche Stromnetz mehrere Minuten. Da Netzzustände sich schnell ändern können, die zentralen Berechnungen und manuelle Vorgehensweise jedoch langwierig sind, bietet es sich an, online und verteilt Einstellungen zu finden, so dass der Stromfluss die Leitungen nicht außerhalb ihrer Spezifikationen belastet.

Das Problem erinnert an die VRGS-Problematik und die Ansätze aus BeeJamA stellen eine Möglichkeit dar, Lösungsstrategien zu erarbeiten. Der wesentliche Unterschied zum Straßenverkehr ist, das ein Fahrzeug nur auf einem einzigen Link eine Belastung herbeiführt. Strom hingegen breitet sich ungefähr mit halber Lichtgeschwindigkeit auf allen Leitungen entsprechend der jeweiligen Widerstände aus. Dieser Einflussbereich ist i.d.R. deutlich größer als nur ein Link. Ziel muss es daher sein, mittels Agenten den gesamten Einflussbereich einer potentiellen PFC-Regelung in die Entscheidung mit einzubeziehen[1].

Der nächste Abschnitt bietet eine Zusammenfassung der Arbeit sowie ein Fazit.

[1] Der Autor war beschäftigt am Lehrstuhl III der Fakultät Informatik der TU Dortmund, der Teil der DFG-Forschergruppe FOR 1511 war, die u.a. dieses Thema behandelte.

8.2. Zusammenfassung

Die vorliegende Arbeit präsentiert und diskutiert das BeeJamA-Protokoll, einschließlich verschiedener Varianten. Es handelt sich dabei um ein verteiltes *Vehicle Routing Guidance System* (VRGS), zum dynamische Routing von Fahrzeugen im Straßenverkehr, inspiriert durch das Schwarmverhalten der Honigbienen beim Sammeln von Futter. Vorrangiges Ziel ist die Verringerung der individuellen Fahrzeiten der Verkehrsteilnehmer. Straßenverkehr ist ein hoch dynamisches, offenes System lauter Unvorhersagbarkeiten. Zentralisierte Ansätze sind zwangsläufig einer Limitierung im abgedecktem Verkehrsraum und/oder den Aktualisierungszeiten unterworfen. Ein verteilter Ansatz wie der vorgestellte bietet die Möglichkeit, bessere Skalierbarkeit in Raum und Zeit zu erzielen.

Das erste Kapitel dieser Arbeit führt in das Thema ein. Dabei wird die Themenstellung motiviert, die Ziele und Vorgehensweise erläutert und verwandte Arbeiten diskutiert. Ferner werden relevante Notationen vereinbart.

Das Kapitel 3 führt ein *Generisches Routing Framework* (GRF) ein, dessen Zweck in der Bereitstellung von Middelware-Funktionalität zwischen Verkehrssimulatoren und Routingprotokollen liegt. Ursprüngliches Ziel war die einfache Anwendbarkeit diverser Verkehrssimulatoren für individuelle Routingprotokolle. Es ist der erste bekannte Versuch, solch eine Middleware zu entwickeln. Es zeigte sich, dass zwei der drei getesteten Simulatoren gegenwärtig nicht für größere Simulationsszenarien auf beliebigen Netzen geeignet erscheinen. Die höhere Detailtreue, im Vergleich zu einem Queue-basierten Ansatz, insbesondere das Verhalten an Kreuzungen sowie unrealistisches Spurwechselverhalten, führte dabei zu erheblichen Problemen in Form von langen Rückstaus. Prinzipiell inkludieren die von den Simulatoren umgesetzten Verkehrsmodelle spezielle Maßnahmen, um Auffahrtssituationen korrekt abzubilden. Dafür müssen die Straßennetze jedoch entsprechend annotiert sein, was aufgrund der Menge der Daten manuell nicht möglich ist. Die vorhandenen automatischen, heuristischen Methoden zur Erkennung von (Autobahn-)Auffahren erwiesen sich als nicht treffsicher genug. So hilfreich das GRF bei der Entwicklung der Routingprotokolle, insbesondere BeeJamA, auch war, von einer einfachen Austauschbarkeit der Simulatoren kann gegenwärtig noch keine Rede sein. In Folgearbeiten muss sichergestellt werden, dass große Straßennetze ohne weitere (zeitraubende) manuelle Eingriffe simuliert werden können. Bis dahin ist mit MATSim ein weitverbreiteter und lauffähiger Simulator an das GRF angebunden, welcher bei der Evaluation des BeeJamA-Protokolls wertvolle Dienste leistete. Schon in der einführenden Vergleichsstudie in Abschnitt 3.5, bei der Aimsun und MATSim mit demselben Grid-Netz und denselben Plänen arbeiteten, zeigten sich jedoch erhebliche Unterschiede. Wie zu erwarten, ergaben sich deutliche höhere Fahrzeiten (bis zum Faktor vier) bei der Verwendung des Aimsun-Simulators.

Das anschließende Kapitel 4 führte traditionelle Konzepte der Verkehrsplanung ein. Die klassische Sichtweise auf Transportprobleme in der Informatik (und Mathematik) ist die Formulierung als, i.d.R. lineares oder konvexes, Optimierungsproblem. Diese Methodiken sind allerdings nur unter starken Einschränkungen erfolgreich. So wird bspw. die vollständige Kenntnis von Verkehrsplänen und Transitzeiten a priori vorausgesetzt. Offensichtlich sind diese Informationen höchstens in Offlinevarianten des Problems gegeben – in realistischen VRGS-Umgebungen hingegen nicht. Denn der Straßenverkehr ist ein Paradebeispiel eines offenen Systems, in dem nur schwerlich Vorhersagen getroffen werden können. Darüber hinaus sind dynamische Varianten des Optimierungsproblems nicht in sinnvollen zeitlichen Größenordnungen zu lösen. Dies erhöht die Anwendbarkeit von verteilten Systemen.

Kapitel 5 führt die für die Arbeit notwendigen Konzepte verteilter Systeme ein. Neben

den Grundlagen sind das zum Einen ein asynchroner Leader-Election-Algorithmus und zum Anderen der asynchrone Bellman-Ford-Algorithmus zur Bestimmung eines LCP. In BeeJamA dient eine Leader Election der Auswahl der repräsentativen Knoten auf höheren Netzschichten. Der Bellman-Ford-Algorithmus ist ein zentraler Bestandteil vieler Routingprotokolle zur verteilten Ermittelung günstigster Pfade. Die Algorithmen werden in der Notation der Timed I/O-Automata vorgestellt und jeweils mit einer Analyse der Nachrichtenkomplexität dargestellt. Dabei wird deutlich, dass eine Leader Election effizient möglich ist, die LCP-Bestimmung mittels asynchronen Bellman-Ford-Algorithmus allerdings im Worst Case faktoriell viele Nachrichten benötigt. Dieses Feststellung hatte direkte Auswirkungen auf die Gestaltung des BeeJamA-Protokolls, da dort eine lineare Nachrichtenkomplexität im Sinne einer hohen Skalierbarkeit besondere Relevanz besitzt.

Ferner wird in diesem Kapitel das Thema Routing, sowie gängige Routingverfahren aus dem Kontext der Kommunikationsnetze eingeführt und vorgestellt. Besonders relevant ist dabei das Distanzvektorprotokoll, welches den grundlegenden technischen Rahmen des BeeJamA- (und BeeHive-)Protokolls darstellt. Vorteilig ist, dass nur anhand von lokalen Daten Forwarding-Entscheidungen getroffen werden können.

Das Kapitel schließt ab, indem Multi Agenten-Systeme und kursorisch das BeeHive-Protokoll eingeführt werden.

Das Kapitel 6 führt das BeeJamA-Protokoll inkl. einiger Varianten in voller Gänze ein. Zunächst wird eine Vehicle-2-Infrastructure (V2I) Architektur vorgestellt, welche die technische Umgebung des Protokolls darstellt. Die Aufteilung des gesamten Straßennetzes in kleinere Bereiche, genannt *Areas*, folgt dabei einem *Divide-and-Conquer*-Ansatz. Dadurch entstehen kleine autonome Teilnetze, in welchen eine möglichst hohe Genauigkeit beim Routing erreicht werden kann, ohne das ein Skalierungsengpass eintritt. Dabei wird angenommen, dass die aktuellen Transitzeiten in einer Area bekannt sind (z.B. mittels statischer Systeme wie Induktionsschleifen und Videoüberprüfung oder Systemen wie Floating Car Data und Phone Velocity Data). Als zweites wird in diesem Kapitel das natürliche Schwarmverhalten der Honigbienen, insbesondere bei der Futtersuche, beschrieben und Verknüpfungen zu dem technischen Kontext aufgezeigt. Das natürliche Vorbild weist hohe Komplexität auf und ist perfekt an die Herausforderungen der Futtersuche in dynamischen Umgebungen angepasst. Technische Adaptionen setzen nur einen Teil des Vorbilds um und sind somit auch von deutlich geringerer Komplexität. Der dritte Teil des Kapitels führt das Multi-Agenten System (MAS) des BeeJamA-Protokolls ein. Dabei wird zunächst ein grober Abriss erstellt und die einzelnen Handlungsoptionen an bestimmten Schlüsselstellen aufgezeigt.

Daraus wird ein Basisprotokoll zur Verbreitung von LCP-Informationen abgeleitet. Diskutiert werden dabei u.a. Konzepte für weitere Hierarchieschichten, Erweiterungen für stochastisches Forwarding, für Reservierungen und für mehrkriterielles Routing. Das umgesetzte Multi-Agenten-System zeichnet sich dadurch aus, dass Wert auf einen geschichteten Aufbau gelegt wurde, wodurch große Gebiete abgedeckt werden können. Andere verteilte Verfahren arbeiten z.T. mit flacher Hierarchie, weswegen Agenten für eine Pfadexploration das gesamte Netz durchlaufen müssen. Zudem wird lineare Nachrichtenkomplexität und Konvergenzgeschwindigkeit (zu LCP-Tabelleneinträgen) durch das Protokoll garantiert. Dadurch ist es möglich, dass kontinuierlich aktuelle Kosteninformationen disseminiert werden können.

Das Kapitel 7 untersucht das BeeJamA-Protokoll im Vergleich zu zentralen Protokollen auf Straßennetzen bestehend aus mehreren Tausend Knoten. Es zeigt sich, dass das BeeJamA-Protokoll zu besseren Fahrzeiten führt als zentrale, dynamische LCP-Protokolle mit fünf minütiger Aktualisierungsfrequenz. Im Gegensatz zur Realität wird dem LCP-Protokoll dabei

sogar eine Aktualisierung auf allen Links gestattet. Insgesamt zeigt sich, dass höhere Aktualisierungsfrequenzen deutlich positive Auswirkungen auf die Fahrzeiten haben. Es zeigt sich auch, dass sich schon bei geringen BeeJamA-Durchdringungen schnell Verbesserungen für alle Fahrzeuge einstellen. Dadurch, dass etwa ein Drittel der Fahrzeuge mit BeeJamA geleitet wurden, sanken in den Simulationen die durchschnittlichen (und maximalen) Fahrzeiten aller Fahrzeuge. Somit profitieren nicht nur die BeeJamA-geführten Fahrzeuge von einer höheren BeeJamA-Durchdringung.

Das Konzept der zusätzlichen Reservierung brachte weitere Fahrzeitvorteile, wenn auch nur eingeschränkt. Naives Reservieren konnte sich nur im Falle von vollständiger Durchdringung positiv auswirken. Die hybride Variante, welche neben dem Reservierungslog, noch die aktuelle Fahrzeit als Indikator mit einbindet, konnte in dem getesteten Szenario schon ab einer Durchdringung von 40% Vorteile einbringen. Ferner konnte festgestellt werden, dass die Akzeptanz von schlechteren Pfaden eines Teils der Fahrer zu deutlich besseren Fahrzeiten bei den übrigen Fahrern führte.

Die Arbeit schließt mit diesem Kapitel, bestehend aus einem Ausblick, dieser Zusammenfassung und dem folgenden Zielabgleich.

8.3. Zielabgleich und Fazit

Die vier in Abschnitt 1.2 genannten Ziele wurden diskutiert und erreicht:

1. Es wurde mit den BeeJamA-Protokollvarianten vollständig verteilte und dezentrale VRGS-Verfahren vorgestellt. Auf der Grundlage der Analyse klassischer verteilter Ansätze wurden Schwarmintelligenzansätze diskutiert und daraus die genannten Varianten abgeleitet unter besonderer Berücksichtigung einer handhabbaren Nachrichtenkomplexität. Die erzielte, linear begrenzte Nachrichtenanzahl ermöglicht die geforderte, *hohe Skalierbarkeit in Raum und Zeit*.

2. Das BeeJamA-MAS ist nicht nachfragebasiert, stattdessen werden kontinuierlich günstige Routen exploriert und bekannt gemacht. Dadurch kann zu jedem Zeitpunkt des Eintreffens einer Next Hop-Anfrage, eine aktuelle ELCP-Next Hop-Antwort durch den Navigator geliefert werden. Durch die hohe Skalierbarkeit ist diese Routingempfehlung stets aktuell (CCP-Kosten) und da der Navigator den Next Hop durch (Hash-)Tabellen-Lookups effizient bestimmen kann, ist gesichert, dass jedes Fahrzeug *rechtzeitig vor jeder Kreuzung* eine aktuelle Empfehlung erhält.

3. Aus dem Erreichen des erstgenannten Ziels und dem Umstand, dass ein Best-Effort-Ansatz pro Generation verfolgt wird (statt eines Stop-And-Wait-Ansatzes), folgt, dass eine *Aktualisierungsfrequenz* für große Systeme im Sekundenbereich möglich ist. Dies beruht auf der Annahme, dass innerhalb des Zeitraums einer Sekunde, sich das System des Straßenverkehrs nicht derartig ändert, dass signifikante Änderungen eintreten. Dadurch kann sich das MAS Fehler bei der Exploration erlauben, die in nachfolgenden Generationen behoben werden.

4. Die Simulationen haben gezeigt, dass Fahrzeitverbesserungen im Vergleich zu etablierten, zentralen Systemen möglich sind. Somit ist auch das Ziel der *Kompetitivität* erfüllt. Es wurde gezeigt, dass eine schnelle Aktualisierung der Pfadkosten immer, und Reservierungen ab einer Durchdringung von 30%, deutliche Vorteile bringen.

Damit wurde im Zuge dieser Arbeit erstmalig ein verteiltes VRGS auf großen Netzen mit diesem Funktionsumfang und unter partieller Durchdringung evaluiert. Die genannten, übergeordneten Ziele, *Kompetitivität* und *Skalierbarkeit*, sind erfüllt.

Anhang A

Timed I/O Automata

Timed I/O Automata (TIOA) sind Zustandsautomaten zur Modellierung verteilter Algorithmen. Die zugehörige Spezifierungssprache stellt eine Mischung aus deklarativen und imperativen Pseudocode dar. Mit TEMPO [142] existiert auch eine kompilierbare Variante (inkl. Ausführungsplattform), die die TIOA-Paradigma in exakt definierter Form umsetzt. Zugunsten der besseren Lesbarkeit wird jedoch die ursprüngliche Pseudocode-Variante verwendet. Um die Länge des benötigten Pseudocodes zu verkürzen, wird zusätzlich ein einfaches Vererbungssystem eingeführt.

TIOAs stellen ein umfangreiches formales Framework dar, welches hier nur soweit eingeführt wird, wie zum Verständnis der dargestellen Automaten notwendig. Einer intuitiven Beschreibung wird dabei der Vorzug gegeben. Eine umfassende theoretische Einführung findet sich in [75]. Neben der reinen Spezifikationsprache bieten TIOAs auch Methoden zur Analyse und Unterstützung zur Durchführung von Korrektheitsbeweisen, dazu sei ebenfalls auf die zuletzt genannte Quelle verwiesen.

Ein TIOA ist ein Tupel $\mathcal{A} = (X, Q, \Theta, A, \mathcal{D}, \mathcal{T})$ mit:

- X als Menge der Zustandsvariablen. Diese sind intern, d.h. nur die spezifische Instanz des Automaten hat darauf Zugriff (es existiert kein Shared Memory Konzept, stattdessen wird ein Benachrichtungskonzept zwischen den Automaten verwendet).
- Q als Menge der möglichen Zustände des Automaten. Jedes Element enthält für jedes $x \in X$ einen gültigen Wert.
- $\Theta \in Q$ als Startzustand.
- Ein Tripel A, bestehend aus den drei disjunkten Menge I, O und H als Input-, Output- und Hidden-Aktionen. Aktionen verändern den Zustand des Automaten. Die Aktionen $I \cup O$ werden als externe Aktionen bezeichnet und stellen das nach außen sichtbare Interface dar. Die Hidden-Aktionen stellen rein interne Aktionen dar und sind nach außen nicht sichtbar. Aktionen haben (ähnlich zu Funktionen oder Methoden imperativer Programme) Parameterlisten. Eine Teilmenge der Parameter kann durch den Automaten fixiert werden.
- $\mathcal{D}$ als Menge sogenannter *diskreter Transitionen*. Formal handelt es sich um das 3-Tupel $(x, a, x') \in D \subseteq Q \times E \times Q$. Von einem Ausgangszustand x wird durch eine externe Aktion ein neuer Zustand x' erreicht.

- $\mathcal{T}$ als Menge der sog. *Trajektorien*, welche die zeitliche Zustandsveränderung beschreibt. Formal, handelt es sich um Differenzialungleichungen, die eine kontinuierliche Veränderung beschreiben.

Die Ausführung der Automaten basiert auf folgenden Grundlagen:

- Die Ausführung einer einzelnen Aktion geschieht in Nullzeit. Diese Annahme basiert auf der Beobachtung, dass in verteilten Systemen die lokale Rechnenzeit und Verarbeitungszeit der Nachrichten vernachlässigbar ist und stattdessen die Nachrichtenkomplexität im Vordergrund steht.
- Output-Aktionen entsprechen dem Versenden und Input-Aktionen dem Empfangen von Nachrichten. Hidden-Aktionen dienen dazu Zustände unabhängig von Nachrichten ändern zu können.
- Bei Output- und Hidden-Aktionen kann der Automat Vorbedingungen formulieren. Sind diese erfüllt, werden diese als „freigegeben“ bezeichnet.
- Input-Aktionen können ihre Ausführung nicht verhindern, d.h. der Automat kann hierfür keine Vorbedingungen formulieren. Input-Aktionen sind daher per definitionem stets freigegeben. Dies basiert auf der Vorstellung, dass ein verteiltes System eine eingehende Nachricht in irgendeiner Form verarbeiten muss.
- Nur freigegebene Aktionen können ausgeführt werden. Die Ausführungsreihenfolge ist zufällig.
- Aktionen sind atomar, können somit nicht unterbrochen werden und es wird pro Automat stets nur eine Aktion gleichzeitig ausgeführt.
- Generell werden freigegebene Aktionen asynchron ausgeführt, der genaue Ausführungszeitpunkt ist daher nicht vorhersagbar. Input-Aktionen werden bei Signalisierung (im Sinne einer eintreffenden Nachricht) sofort ausgeführt. Freigegebene Output- und Hidden-Aktionen können zu beliebigen Zeitpunkten ausgeführt werden.
- Automaten kennen die Realzeit. Mittels Trajektorien können Variablen im Verhältnis zu der aktuellen Realzeit verändert werden. In dieser Arbeit wird dieses (ausdruckstarke Mittel) aber nur dazu verwendet, eine Ausführung freigegebener Aktionen spätestens zu einer Deadline zu garantieren. Aufgrund der Asynchronität könnten freigegebene Aktionen zu jedem Zeitpunkt vorher ausgeführt worden sein, falls nicht, wird dadurch eine Ausführung erzwungen.

TIOAs kommunizieren mittels eines abstrakten Nachrichtenkonzepts. Ein Sender-Automat sendet einem Empfänger-Automat ein Nachricht, macht dabei aber keine Aussage wie diese Nachricht übertragen wird. Dazu werden zusätzliche Automaten eingesetzt, die den Nachrichtenkanal modellieren. Diese Nachrichtenkommunkation ist dadurch umgesetzt, dass der Sender eine Output-Aktion ausführt, wodurch automatisch unmittelbar eine passende Input-Aktion eines Empfängers ausgeführt wird. Ob eine Input-Aktion passend ist, wird mittels eines Matchings festgestellt: Übereinstimmen muss der Name der Ouput- und Input-Aktion, als auch die fixierten Parameter (dieser Aspekt wird weiter unten noch exemplifiziert).

TIOAs kennen insgesamt zwei Möglichkeiten der Zustandsänderung von $x \in Q$ zu $x' \in Q$:

1. Zum Einen kann ein Zustandswechsel asynchron durch eine diskrete Transition auftreten, nämlich wenn eine Aktion $a \in A$ ausgeführt wird.

2. Zum Anderen kann mittels Trajektorien eine kontinuierliche Veränderung von Variablen (und damit des Zustandes) festgelegt werden.

Der Automat 14 zeigt einen TIOA für einen asynchronen Kommunikationskanal, der Verlust- und Fehlerfreiheit garantiert (vgl. Annahme 5.1), sowie sicherstellt, dass jede Nachricht spätestens zu einer vorgegeben Deadline b ihr Ziel erreicht.

Zeile 1 definiert einen TIOA mit dem Namen „TimedChannel“ und den drei Parametern b (garantierte, maximale Dauer der Übertragung auf einem Kanal), i (Startknoten des Kanals) und j (Endknoten des Kanals). Mit der **where**-Klausel wird die Nicht-Negativität von b gefordert. Anschließend werden die Zustandsvariablen definiert und deklariert (Zeile 3–5). In dieser TIOA-Pseudocode-Variante wird dies nur informell gemacht, wo nötig geben ergänzende Kommentare Typinformationen wieder.

Benötigt für den Kanal wird eine Queue und eine reellwertige Clock-Variable zur Verfolgung der aktuellen Zeit. Die Queue wird als eine leere Menge initialisiert. Die Elemente der Queue sind 2-Tupel (als Liste organisiert), wobei das erste Element die zu übertragende Nachricht und das zweite Element die einzuhaltende Deadline darstellt.

Als nächstes folgen die diskreten Transitionen (Zeile 7–15). Es wird eine Input- und eine Output-Aktion definiert. Das **Preconditions**-Schlüsselwort leitet die Vorbedingungen ein. In dem mit dem **Effects**-Schlüsselwort gekennzeichneten Bereich einer jeden Aktion a sind in klassischer imperativer Form die Auswirkungen auf den ursprünglichen Zustand x angegeben.

Inhaltlich sind die Aktionen des Kanals simpel. Die Input-Aktion fügt die weiterzuleitende Nachricht, zusammen mit der Deadline einfach der Versandqueue hinzu. Die Output-Aktion wird bei nicht-leerer Queue freigegeben. Der zweite Teil der Vorbedingung legt den Wert des Parameters m fest: Es wird die Output-Aktion $receive(m, i, j)$ aktiviert (und damit eine Input-Aktion $receive(m, i, j)$ des Empfängers ausgelöst), wenn m ältestes Element der Queue ist. Dadurch ist der Nachrichtenparameter fixiert. So wird die Nachricht m an den Automaten übermittelt, der eine Input-Aktion $receive(m, i, j)$ bereitstellt, wobei i, j durch die Parameterliste des Empfängerautomaten ebenfalls fixiert sind.

Der Effekt der Output-Aktion ist dann simpel, denn es muss nur das erste Element der Queue gelöscht werden (Zeile 15). Im imperativen Teil des Automaten ist demnach gar keine Übertragungslogik notwendig. Hingegen geschieht dies implizit durch die Output-Input-Beziehung der Automaten untereinander.

Die Trakjektorie zeigt mit dem **stop when**-Schlüsselwort (Zeile 18) an, dass die Zeit nicht über den Zeitpunkt hinweg laufen kann, an dem das Prädikat hinter dem Schlüsselwort wahr wird. Demnach wird spätestens zu diesem Zeitpunkt der Automat in beliebiger Reihenfolge alle freigegebenen Output- oder internen Aktionen ausführen.

Damit wird die Output-Aktion spätestens zum Zeitpunkt des Eintretens der Deadline ausgeführt. Da diese eine Ausführungszeit von Null besitzt, wird die korrespondierende Input-Aktion des Empfängers innerhalb der Deadline aufgerufen. Die Nachricht kommt folglich stets rechtzeitig an. Aufgrund der Atomarität wird auch die exakte Reihenfolge der Nachrichten eingehalten, da stets das älteste Queueelement abgearbeitet wird und die Aktion dabei nicht unterbrochen werden kann. Ferner ist der Kanal verlustfrei, da alle der Queue hinzugefügten Nachrichten ihr Ziel erreichen.

Neben dem Kanal werden zusätzlich Sender benötigt, die die Input-Aktion des Kanals auslösen. Prinzipiell könnte jedem Automat diese Logik hinzugefügt werden, das soll hier

Automaton 14 TimedChannel

```
1:  Automaton TimedChannel(i, j, b) where b ≥ 0
2:
3:      States:
4:          queue ← ∅                                   ▷ type of [Message, Deadline]
5:          clock ← 0
6:
7:      Transitions:
8:          Input send(m, i, j)
9:              Effects
10:                 queue ← queue ⊢ [m, clock+b]
11:         Output receive(m,i,j)
12:             Preconditions
13:                 queue ≠ ∅ ∧ queue[0].message = m
14:             Effects
15:                 queue ← queue[1...]
16:
17:     Trajectory:
18:         stop when ∃i ∈ ℕ: queue[i].deadline = clock
19:         evolve d(clock) = 1
20: End
```

aber aus Übersichtlichkeitsgründen getrennt werden. Es werden im Folgenden mehrere Sender vorgestellt, von denen die Automaten, die die eigentlichen Algorithmen umsetzen, erben. Die Sender-Automaten besitzen jeweils eine eigene Queue, die von den ableitenden Automaten dann nur noch befüllt werden muss.

Automat 15 stellt einen Sender-Automaten dar, der als bald als möglich die Nachrichten in der Variable *sendQueue* an den Kanal übergibt. Die Output-Aktion ist freigegeben sobald eine Nachricht der Queue hinzugefügt wurde. Aufgrund der stets zutreffenden Trajektorie wird zu jedem Zeitpunkt die Freigabe geprüft und gegebenenfalls die Aktion ausgeführt. Der lokale Effekt besteht nur in dem Entfernen des ältesten Queue-Elementes.

Der zweite Sender, Automat 16, kapselt die benötigte Flooding-Logik. Der Automat erbt von Automaten 15 durch das **extends**-Schlüsselwort. Dabei handelt es sich um eine semantische Erweiterung, die in der ursprünglichen Beschreibung TIOAs nicht existent ist. Verwendet wird diese Erweiterung hier ausschließlich zur Verbesserung der Lesbarkeit. Variablen gelten als „protected“ markiert, Aktionen als „public“. Dadurch zählt einfach die Output-Aktion *Sender.send* zu den externen Aktionen des Automaten *Flooder* und dieser hat auch Zugriff auf die Variable *sendQueue* des Automaten *Sender*[1]. Zudem gelten auch die Trajektorien für die erbenden Automaten. Der *Flooder*-Automat definiert zusätzlich die Variable $floodQueue$, welche die später erbenden Algorithmen befüllen können. Aufgrund der geerbten Trajektorie wird die interne Aktion *Flooder.sendUpdate* ausgeführt sobald eine Nachricht in die $floodQueue$ gelangt. Die Flooding-Logik ist einfach: Eine Nachricht wird an alle Ziele $j \in D$ gesendet, sobald das Ziel nicht ausgeschlossen ist ($j \in E$) und das *Travel Limit* einer Nachricht nicht überschritten ist: $l - \zeta_{ij} \geq 0$, mit ζ_{ij} als Kosten für die Passage des Links (i, j) für eine

[1] Nameskollisionen werden hier durch eindeutige Benennung umgangen.

Automaton 15 Sender

```
1:  Automaton Sender(i)
2:
3:      States:
4:          sendQueue ← ∅                                   ▷ of type [Message, Destination]
5:
6:      Transitions:
7:          Output send(m, i, j)
8:              Preconditions
9:                  sendQueue ≠ ∅∧ sendQueue[0].msg = m ∧ sendQueue[0].dst = j
10:             Effects
11:                 sendQueue ← sendQueue[1...]
12:     Trajectory:
13:         stop when true
14: End
```

Automaton 16 Flooder

```
1:  Automaton Flooder(i,D) extends Sender(i)
2:
3:      States:
4:          floodQueue ← ∅                          ▷ of type [Message, ExcludedNodes, Limit]
5:
6:      Transitions:
7:          Internal sendUpdate()
8:              Preconditions
9:                  floodQueue ≠ ∅
10:             Effects
11:                 sendQueue ← sendQueue ∪{[[m, l − ζ_ij], j] | (∃[m, E, l] ∈ floodQueue:
12:                                 j ∈ D \ E) ∧ (l − ζ_ij ≥ 0)}
13:                 floodQueue ← ∅
14: End
```

Nachricht m. (Nicht zu verwechseln mit ω_{ij} als Kosten für die Passage eines Tokens.) Damit wird das Hop Limit der Algorithmen in Kapitel 5 umgesetzt, indem für alle (i, j) die Kosten auf 1 gesetzt werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Index

Literaturverzeichnis

[1] ADAC: *Staubilanz 2012*. – URL http://www.adac.de/infotestrat/adac-im-einsatz/motorwelt/Staubilanz2012.aspx. – Zugriffsdatum: 12.12.2014

[2] Ahuja, Ravindra K. ; Magnanti, Thomas L. ; Orlin, James B. u. a.: *Network flows: theory, algorithms, and applications*. Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 1988

[3] Anshelevich, Elliot ; Ukkusuri, Satish: Equilibria in dynamic selfish routing. In: *Algorithmic Game Theory*. Springer, 2009, S. 171–182

[4] Arel, I. ; Liu, C. ; Urbanik, T. ; Kohls, A. G.: Reinforcement learning-based multi-agent system for network traffic signal control. In: *Intelligent Transport Systems, IET* 4 (2010), Nr. 2, S. 128–135. – ISSN 1751-956X

[5] Arnold Jr, Ed: Ramp metering: a review of the literature. In: *Contract* 21535 (1998), S. 6940

[6] Balmer, M. ; Meister, K. ; Rieser, M. ; Nagel, K. ; Axhausen, Kay W.: *Agent-based simulation of travel demand: Structure and computational performance of MATSim-T*. ETH, Eidgenössische Technische Hochschule Zürich, IVT Institut für Verkehrsplanung und Transportsysteme, 2008

[7] Bansal, Meenakshi ; Rajput, Rachna ; Gupta, Gaurav: *Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations*. 1998

[8] Bar-Gera, Hillel ; Nie, Yu ; Boyce, David ; Hu, Yucong ; Liu, Yang: Consistent route flows and the condition of proportionality. In: *Transportation Research Board 89th Annual Meeting*, 2010

[9] Barceló, Jaime ; Casas, Jordi: Dynamic network simulation with AIMSUN. In: *Simulation Approaches in Transportation Analysis*. Springer, 2005, S. 57–98

[10] Beckmann, Martin ; McGuire, C. ; Winsten, Christopher B.: *Studies in the Economics of Transportation*. Yale University Press, New Haven, 1956

[11] Belanovic, Pavle ; Valerio, Danilo ; Paier, Alexander ; Zemen, Thomas ; Ricciato, Fabio ; Mecklenbrauker, Christoph F.: On wireless links for vehicle-to-infrastructure communications. In: *Vehicular Technology, IEEE Transactions on* 59 (2010), Nr. 1, S. 269–282

[12] Bertsekas, Dimitri P. ; Tsitsiklis, John N.: *Parallel and distributed computation: numerical methods*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1989. – ISBN 0-13-648700-9

[13] Biswas, Subir ; Tatchikou, Raymond ; Dion, Francois: Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. In: *Communications Magazine, IEEE* 44 (2006), Nr. 1, S. 74–82

[14] Blum, Wolfgang: *Ewig lockt die Schnellstraße*. – URL http://www.sueddeutsche.de/wissen/ewig-lockt-die-schnellstrasse-1.913440. – Zugriffsdatum: 12.12.2014

[15] Bonabeau, Eric ; Dorigo, Marco ; Theraulaz, Guy: *Swarm intelligence: from natural to artificial systems*. Bd. 4. Oxford university press New York, 1999

[16] Boyce, David ; Ralevic-Dekic, Biljana ; Bar-Gera, Hillel: Convergence of Traffic Assignments: How Much is Enough? The Delaware Valley Case Study. In: *16th Annual International EMME/2 Users' Conference, Albuquerque, New Mexico*, 1988

[17] Braess, D.: Über ein Paradoxon aus der Verkehrsplanung. In: *Unternehmensforschung* 12 (1968), Nr. 1, S. 258–268

[18] Brent, Richard P.: *Algorithms for minimization without derivatives*. Englewood Cliffs, N.J. Prentice-Hall, 1973 (Prentice-Hall series in automatic computation). – URL http://opac.inria.fr/record=b1082765. – ISBN 0-13-022335-2

[19] Buch, Norbert ; Velastin, Sergio A. ; Orwell, James: A review of computer vision techniques for the analysis of urban traffic. In: *Intelligent Transportation Systems, IEEE Transactions on* 12 (2011), Nr. 3, S. 920–939

[20] Campbell, M. E.: *Route selection and traffic assignment*. Highway Research Board, Washington, DC, 1950

[21] Cassidy, Michael J.: Freeway on-ramp metering, delay savings, and diverge bottleneck. In: *Transportation Research Record: Journal of the Transportation Research Board* 1856 (2003), Nr. 1, S. 1–5

[22] Chawla, Shuchi ; Roughgarden, Tim: Single-source stochastic routing. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2006, S. 82–94

[23] Chen, C.L.P. ; Zhou, Jin ; Zhao, Wei: A Real-Time Vehicle Navigation Algorithm in Sensor Network Environments. In: *Intelligent Transportation Systems, IEEE Transactions on* 13 (2012), Nr. 4, S. 1657–1666. – ISSN 1524-9050

[24] Claes, R. ; Holvoet, T. ; Weyns, D.: A Decentralized Approach for Anticipatory Vehicle Routing Using Delegate Multiagent Systems. In: *Intelligent Transportation Systems, IEEE Transactions on* 12 (2011), June, Nr. 2, S. 364–373. – ISSN 1524-9050

[25] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to algorithms.* MIT press, 2001

[26] Dafermos, Stella C. ; Sparrow, Frederick T.: The traffic assignment problem for a general network. In: *Journal of Research of the National Bureau of Standards, Series B* 73 (1969), Nr. 2, S. 91–118

[27] Dantzig, George B.: *Linear programming and extensions.* Princeton University Press, 1998

[28] Dargay, Joyce ; Gately, Dermot ; Sommer, Martin: Vehicle ownership and income growth, worldwide: 1960-2030. In: *The Energy Journal* (2007), S. 143–170

[29] De Fabritiis, Corrado ; Ragona, Roberto ; Valenti, Gaetano: Traffic estimation and prediction based on real time floating car data. In: *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on* IEEE (Veranst.), 2008, S. 197–203

[30] Deb, Kalyanmoy: Multi-objective optimization. In: *Multi-objective optimization using evolutionary algorithms* (2001), S. 13–46

[31] Deneubourg, J.-L. ; Aron, Serge ; Goss, Simon ; Pasteels, Jacques M.: The self-organizing exploratory pattern of the argentine ant. In: *Journal of insect behavior* 3 (1990), Nr. 2, S. 159–168

[32] Desai, P. ; Loke, S.W. ; Desai, A. ; Singh, J.: Multi-agent based vehicular congestion management. In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*, 2011, S. 1031–1036

[33] Dial, Robert B.: A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration. In: *Transportation Research Part B: Methodological* 40 (2006), Nr. 10, S. 917–936

[34] DiCaro, G. ; Dorigo, M.: Mobile agents for adaptive routing. In: *Proc. Thirty-First Hawaii International Conference on System Sciences* Bd. 7, 1998, S. 74–83

[35] Dijkstra, E. W.: A note on two problems in connexion with graphs. In: *Numerische Mathematik* 1 (1959), S. 269–271

[36] Domencich, Thomas A. ; McFadden, Daniel: *Urban Travel Demand-A Behavioral Analysis.* North-Holland Publishing Company, 1975 (Contributions to Economic Analysis)

[37] Dorigo, M. ; Birattari, M. ; Stutzle, T.: Ant colony optimization. In: *Computational Intelligence Magazine, IEEE* 1 (2006), Nov, Nr. 4, S. 28–39. – ISSN 1556-603X

[38] Dorigo, Marco ; Gambardella, Luca M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. In: *Evolutionary Computation, IEEE Transactions on* 1 (1997), Nr. 1, S. 53–66

[39] Dorigo, Marco ; Maniezzo, Vittorio ; Colorni, Alberto: The Ant System: Optimization by a colony of cooperating agents. In: *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B* 26 (1996), Nr. 1, S. 29–41

[40] Downs, Anthony: The law of peak-hour expressway congestion. In: *Traffic Quarterly* 16 (1962), Nr. 3

[41] Du, Jie ; Barth, Matthew J.: Next-generation automated vehicle location systems: Positioning at the lane level. In: *Intelligent Transportation Systems, IEEE Transactions on* 9 (2008), Nr. 1, S. 48–57

[42] Duffell, J. R. ; Kalombaris, A.: Empirical studies of car driver route choice in Herfordshire. In: *Traffic Engineering and Control* 29 (1988), Nr. 7/8

[43] Enslow, Philip H.: *What is a"distributed"data processing system?* School of Information and Computer Science, Georgia Institute of Technology, 1978

[44] Escalera, Arturo de la ; Armingol, J. M. ; Mata, Mario: Traffic sign recognition and analysis for intelligent vehicles. In: *Image and vision computing* 21 (2003), Nr. 3, S. 247–258

[45] Farooq, M.: *Bee-Inspired Protocol Engineering: From Nature to Networks*. Springer Berlin Heidelberg, 2009 (Natural Computing Series). – ISBN 9783540859543

[46] Feinler, Elizabeth J. ; Postel, Jonathan: ARPANET Protocol Handbook. In: *NASA STI/Recon Technical Report N* 77 (1976), S. 16237

[47] Fisk, Caroline ; Pallottino, Stefano: Empirical evidence for equilibrium paradoxes with implications for optimal planning strategies. In: *Transportation Research Part A: General* 15 (1981), Nr. 3, S. 245–248

[48] Floyd, Robert W.: Algorithm 97: Shortest Path. In: *Commun. ACM* 5 (1962), Juni, Nr. 6, S. 345–. – URL http://doi.acm.org/10.1145/367766.368168. – ISSN 0001-0782

[49] Fontanelli, Stefano ; Bini, Enrico ; Santi, Paolo: Dynamic route planning in vehicular networks based on future travel estimation. In: *Vehicular Networking Conference (VNC), 2010 IEEE* IEEE (Veranst.), 2010, S. 126–133

[50] Frank, Marguerite ; Wolfe, Philip: An algorithm for quadratic programming. In: *Naval research logistics quarterly* 3 (1956), Nr. 1-2, S. 95–110

[51] Fredman, Michael L. ; Tarjan, Robert E.: Fibonacci heaps and their uses in improved network optimization algorithms. In: *J. ACM* 34 (1987), Juli, Nr. 3, S. 596–615. – ISSN 0004-5411

[52] Galib, Syed ; Moser, Irene: Road traffic optimisation using an evolutionary game. In: *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation* ACM (Veranst.), 2011, S. 519–526

[53] Gawron, Christian: *Simulation-based traffic assignment: Computing user equilibria in large street networks*, Universität Köln / Fachbereich Physik, Dissertation, 1999

[54] Giagkos, Alexandros ; Wilson, Myra S.: BeeIP: bee-inspired protocol for routing in mobile ad-hoc networks. In: *From Animals to Animats 11*. Springer, 2010, S. 263–272

[55] Gipps, Peter G.: A behavioural car-following model for computer simulation. In: *Transportation Research Part B: Methodological* 15 (1981), Nr. 2, S. 105–111

[56] Google, Inc.: *Blog*. 07 2012. – URL http://googleblog.blogspot.hu/2012/08/the-self-driving-car-logs-more-miles-on.html. – Zugriffsdatum: 12.12.2014

[57] Griffin, Timothy G. ; Sobrinho, Joäo Luís: Metarouting. In: *ACM SIGCOMM Computer Communication Review* Bd. 35 ACM (Veranst.), 2005, S. 1–12

[58] Häger, Ulf ; Rehtanz, Christian ; Lehnhoff, Sebastian: Analysis of the robustness of a distributed coordination system for power flow controllers. In: *17th international Power Systems Computation Conference (PSCC), Stockholm, Sweden* Bd. 22, 2011, S. 26

[59] Hart, P. E. ; Nilsson, N. J. ; Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *Systems Science and Cybernetics, IEEE Transactions on* 4 (1968), Nr. 2, S. 100–107. – ISSN 0536-1567

[60] Hartenstein, Hannes ; Laberteaux, Kenneth P.: A tutorial survey on vehicular ad hoc networks. In: *Communications Magazine, IEEE* 46 (2008), Nr. 6, S. 164–171

[61] Hedrick, C.: *RFC 1058 - Routing Internet Protocol (RIP), 1988*. – URL https://tools.ietf.org/html/rfc1058. – Zugriffsdatum: 12.12.2014

[62] Hennessy, Dwight A. ; Wiesenthal, David L.: Traffic congestion, driver stress, and driver aggression. In: *Aggressive behavior* 25 (1999), Nr. 6, S. 409–423

[63] Herescu, Oltea M. ; Palamidessi, Catuscia: On the Generalized Dining Philosophers Problem. In: *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*. New York, NY, USA : ACM, 2001 (PODC '01), S. 81–89. – URL http://doi.acm.org/10.1145/383962.383994. – ISBN 1-58113-383-9

[64] Hoar, R. ; Penner, J. ; Jacob, C.: Evolutionary swarm traffic: if ant roads had traffic lights. In: *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on* Bd. 2, 2002, S. 1910–1915

[65] Hoffmann, Bastian: *Anbindung des mikroskopischen Aimsun-Verkehrssimulators an ein generisches Routing-Framework*, TU Dortmund, Fak. Informatik, LS3 III, Diplomarbeit, 2013

[66] Höfler, L.: Was ist und wie wirkt Stau? Definition und Wirkungen eines Massenphänomäns. In: *Der öffentliche Sektor / Forschungsmemorandum* 3 (2005), S. 41–51

[67] Horowitz, Alan J.: *Delay/Volume relations for travel forecasting based upon the 1985 highway capacity manual*. Federal Highway Administration, US Department of Transportation, 1991

[68] Ikeda, Takeo ; Yoshii, Masaaki ; Doi, Youichi ; Mitoh, Kunihiko: *Adaptive in-vehicle route guidance system*. Juli 9 1991. – US Patent 5,031,104

[69] ISO: *Traffic and Traveller Information (TTI) – TTI messages via traffic message coding – Part 1: Coding protocol for Radio Data System – Traffic Message Channel (RDS-TMC) using ALERT-C*. ISO 14819-1:2003. 2003

[70] Jacob, C. ; Abdulhai, B.: Integrated traffic corridor control using machine learning. In: *Systems, Man and Cybernetics, 2005 IEEE International Conference on* Bd. 4, 2005, S. 3460–3465 Vol. 4

[71] Johnson, Donald B.: Efficient Algorithms for Shortest Paths in Sparse Networks. In: *J. ACM* 24 (1977), Januar, Nr. 1, S. 1–13. – URL http://doi.acm.org/10.1145/321992.321993. – ISSN 0004-5411

[72] Kahlert, Roland: *Anbindung des SUMO-Verkehrssimulators an ein generisches Routingframework*, TU Dortmund, Fak. Informatik, LS3 III, Diplomarbeit, 2012

[73] Kanoh, Hitoshi: Dynamic route planning for car navigation systems using virus genetic algorithms. In: *International Journal of Knowledge-based and Intelligent Engineering Systems* 11 (2007), Nr. 1, S. 65–78

[74] Kasinger, Holger ; Denzinger, Jörg ; Bauer, Bernhard: Decentralized coordination of homogeneous and heterogeneous agents by digital infochemicals. In: *Proceedings of the 2009 ACM symposium on Applied Computing* ACM (Veranst.), 2009, S. 1223–1224

[75] Kaynar, Dilsun K. ; Lynch, Nancy ; Segala, Roberto ; Vaandrager, Frits: The theory of timed I/O automata. In: *Synthesis Lectures on Distributed Computing Theory* 1 (2010), Nr. 1, S. 1–137

[76] Kerner, Boris S.: *Introduction to modern traffic flow theory and control. The long road to three-phase traffic theory.* Berlin: Springer, 2009. – xiii + 265 S. – ISBN 978-3-642-02604-1/hbk; 978-3-642-02605-8/ebook

[77] Knobloch, Fabian: *Konzeption und Entwicklung eines Generischen Routing Frameworks für die Simulation von Straßenverkehr in großen Netzen*, TU Dortmund, Fak. Informatik, LS3 III, Diplomarbeit, 2010

[78] Koch, Ronald ; Skutella, Martin: Nash equilibria and the price of anarchy for flows over time. In: *Algorithmic Game Theory.* Springer, 2009, S. 323–334

[79] Kolata, G.: What if they closed 42nd Street and nobody noticed? In: *New York Times* (25. Dezember 1990), S. 38

[80] Kong, Q.-J. ; Zhao, Qiankun ; Wei, Chao ; Liu, Yuncai: Efficient Traffic State Estimation for Large-Scale Urban Road Networks. In: *Intelligent Transportation Systems, IEEE Transactions on* 14 (2013), March, Nr. 1, S. 398–407

[81] Krauss, Stefan: *Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics*, Universität zu Köln., Dissertation, 1998

[82] Kurose, James F. ; Ross, Keith W.: *Computer networking.* Pearson Education, 2012

[83] Kwong, Karric ; Kavaler, Robert ; Rajagopal, Ram ; Varaiya, Pravin: Real-time measurement of link vehicle count and travel time in a road network. In: *Intelligent Transportation Systems, IEEE Transactions on* 11 (2010), Nr. 4, S. 814–825

[84] Lamport, Leslie: The temporal logic of actions. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16 (1994), Nr. 3, S. 872–923

[85] Lamport, Leslie ; Shostak, Robert ; Pease, Marshall: The Byzantine generals problem. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4 (1982), Nr. 3, S. 382–401

[86] Lewis, Franck L.: Wireless sensor networks. In: *Smart environments: technologies, protocols, and applications* (2004), S. 11–46

[87] Li, Qinglan ; Beaver, Jonathan ; Amer, Ahmed ; Chrysanthis, Panos K. ; Labrinidis, Alexandros ; Santhanakrishnan, Ganesh: Multi-criteria routing in wireless sensor-based pervasive environments. In: *International Journal of Pervasive Computing and Communications* 1 (2005), Nr. 4, S. 313–326

[88] Lohmann, Tim: *Test und Konzeption von Routingalgorithmen für ein generisches Routing Framework*, TU Dortmund, Fak. Informatik, LS3 III, Diplomarbeit, 2010

[89] Lott, Christopher ; Teneketzis, Demosthenis: Stochastic routing in ad-hoc networks. In: *Automatic Control, IEEE Transactions on* 51 (2006), Nr. 1, S. 52–70

[90] Luettel, Thorsten ; Himmelsbach, Michael ; Wuensche, H.-J.: Autonomous Ground Vehicles—Concepts and a Path to the Future. In: *Proceedings of the IEEE* 100 (2012), Nr. 13, S. 1831–1839

[91] Lynch, Nancy A.: *Distributed algorithms*. Morgan Kaufmann, 1996

[92] Malakooti, Behnam ; Thomas, Ivan: A distributed composite multiple criteria routing using distance vector. In: *Networking, Sensing and Control, 2006. ICNSC'06. Proceedings of the 2006 IEEE International Conference on* IEEE (Veranst.), 2006, S. 42–47

[93] MATSim: *Homepage*. – URL http://matsim.org. – Zugriffsdatum: 12.12.2014

[94] McQuillan, John M. ; Richer, Ira ; Rosen, Eric: The new routing algorithm for the ARPANET. In: *Communications, IEEE Transactions on* 28 (1980), Nr. 5, S. 711–719

[95] Milner, Robin ; Parrow, Joachim ; Walker, David: A calculus of mobile processes, i. In: *Information and computation* 100 (1992), Nr. 1, S. 1–40

[96] Miura, J. ; Kanda, T. ; Shirai, Y.: An active vision system for real-time traffic sign recognition. In: *Intelligent Transportation Systems, 2000. Proceedings. 2000 IEEE*, 2000, S. 52–57

[97] Mogridge, Martin J. H.: The self-defeating nature of urban road capacity policy: A review of theories, disputes and available evidence. In: *Transport Policy* 4 (1997), Nr. 1, S. 5–23

[98] Moy, John: Open shortest path first (ospf) version 2. In: *IETF: The Internet Engineering Taskforce RFC* 2328 (1998)

[99] Murata, Tadao: Petri nets: Properties, analysis and applications. In: *Proceedings of the IEEE* 77 (1989), Nr. 4, S. 541–580

[100] Nagel, K. ; Schreckenberg, M.: A Cellular Automaton Model for Freeway Traffic. In: *Physique* 2 (1992)

[101] Nash, John F.: Equilibrium points in n-person games. In: *Proceedings of the national academy of sciences* 36 (1950), Nr. 1, S. 48–49

[102] Nash, John F.: *Non-cooperative games*, Princeton University, Dissertation, Mai 1950

[103] Olariu, Stephan ; Khalil, Ismail ; Abuelela, Mahmoud: Taking VANET to the clouds. In: *International Journal of Pervasive Computing and Communications* 7 (2011), Nr. 1, S. 7–21

[104] OpenStreetMap: *Homepage*. – URL http://www.openstreetmap.de

[105] Papageorgiou, Markos ; Diakaki, Christina ; Dinopoulou, Vaya ; Kotsialos, Apostolos ; Wang, Yibing: Review of road traffic control strategies. In: *Proceedings of the IEEE* 91 (2003), Nr. 12, S. 2043–2067

[106] Park, Jungme ; Murphey, Yi L. ; Kristinsson, Johannes ; McGee, Ryan ; Kuang, Ming ; Phillips, Tony: Intelligent speed profile prediction on urban traffic networks with machine learning. In: *Neural Networks (IJCNN), The 2013 International Joint Conference on* IEEE (Veranst.), 2013, S. 1–7

[107] Patriksson, P.: *The traffic assignment problem: models and methods*. VSP International Science Publishers, 1994

[108] Perkins, Charles E. ; Bhagwat, Pravin: Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In: *ACM SIGCOMM Computer Communication Review* Bd. 24 ACM (Veranst.), 1994, S. 234–244

[109] Peterson, Anders: *The Origin-Destination Matrix Estimation Problem: Analysis and Computations*, Dept. of Science and Technology, Linköpings, Dissertation, 2007

[110] Poli, Riccardo ; Kennedy, James ; Blackwell, Tim: Particle swarm optimization. In: *Swarm intelligence* 1 (2007), Nr. 1, S. 33–57

[111] Prashanth, LA ; Bhatnagar, Shalabh: Reinforcement learning with function approximation for traffic signal control. In: *Intelligent Transportation Systems, IEEE Transactions on* 12 (2011), Nr. 2, S. 412–421

[112] Promnoi, S. ; Tangamchit, P. ; Pattara-Atikom, W.: Road traffic estimation based on position and velocity of a cellular phone. In: *ITS Telecommunications, 2008. ITST 2008. 8th International Conference on* IEEE (Veranst.), 2008, S. 108–111

[113] Raabe, Niclas: *Konzeption und Weiterentwicklung eines generischen Routingframeworks für Straßenverkehrs-Simulationen*, TU Dortmund, Fak. Informatik, LS3 III, Diplomarbeit, 2012

[114] Ran, Bin ; Boyce, David: *Dynamic urban transportation network models: theory and implications for intelligent vehicle-highway systems*. Springer, 1994

[115] Rekhter, Y. ; Li, T ; Hares, S.: *RFC 4271 - Border Gateway Protocol 4 (BGP), 2006*. – URL https://tools.ietf.org/html/rfc4271. – Zugriffsdatum: 12.12.2014

[116] Ribeiro, Alejandro ; Sidiropoulos, N. ; Giannakis, Georgios: Optimal distributed stochastic routing algorithms for wireless multihop networks. In: *Wireless Communications, IEEE Transactions on* 7 (2008), Nr. 11, S. 4261–4272

[117] Ridge, Enda ; Kudcnko, Daniel ; Kazakovi, Dimitar u. a.: Moving Nature-Inspired Algorithms to Parallel, Asynchronous and Decentralised Environments. In: *Self-organization And Autonomic Informatics 1* 1 (2005), S. 35

[118] Rothengatter, W.: *External costs of transport*. 2004. – Online: http://www.uic.asso.fr/html/environnement/cd_external/docs/externalcosts_en.pdf

[119] Roughgarden, Tim: The price of anarchy is independent of the network topology. In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing* ACM (Veranst.), 2002, S. 428–437

[120] Ruey-Shun, Chen ; Duen-Kai, Chen ; Szu-Yin, Lin: Actam: Cooperative multi-agent system architecture for urban traffic signal control. In: *IEICE transactions on Information and Systems* 88 (2005), Nr. 1, S. 119–126

[121] Saleem, Muhammad ; Farooq, Muddassar: Beesensor: A bee-inspired power aware routing protocol for wireless sensor networks. In: *Applications of Evolutionary Computing*. Springer, 2007, S. 81–90

[122] Schmitt, E. J. ; Jula, H.: Vehicle Route Guidance Systems: Classification and Comparison. In: *Proc. IEEE Intl. Tranp. Sys.* (2006), S. 242–247

[123] Schulz, Andreas S. ; Moses, Nicolás S.: On the performance of user equilibria in traffic networks. In: *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* Society for Industrial and Applied Mathematics (Veranst.), 2003, S. 86–87

[124] Seeley, Thomas D.: *The wisdom of the hive: the social physiology of honey bee colonies*. Harvard University Press, 2009

[125] Senge, Sebastian: Assessment of path reservation in distributed real-time vehicle guidance. In: *Intelligent Vehicles Symposium Workshops (IV Workshops), 2013 IEEE*, June 2013, S. 87–92

[126] Senge, Sebastian ; Wedde, Horst F.: Bee inspired online vehicle routing in large traffic systems. In: *ADAPTIVE 2010, The Second International Conference on Adaptive and Self-Adaptive Systems and Applications*, 2010, S. 78–83

[127] Senge, Sebastian ; Wedde, Horst F.: 2-Way evaluation of the distributed BeeJamA vehicle routing approach. In: *IEEE Intelligent Vehicles Symposium (IV), 2012* IEEE (Veranst.), 2012, S. 205–210

[128] Senge, Sebastian ; Wedde, Horst F.: Bee-inspired Road Traffic Control as an Example of Swarm Intelligence in Cyber-Physical Systems. In: *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, 2012, S. 258–265

[129] Senge, Sebastian ; Wedde, Horst F.: Minimizing vehicular travel times using the multi-agent system beejama. In: *Product-Focused Software Process Improvement*. Springer, 2012, S. 335–349

[130] Senge, Sebastian ; Wedde, Horst F.: Marginal cost pricing and multi-criteria routing in a distributed swarm-intelligence approach for online vehicle guidance. In: *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, Oct 2013, S. 1396–1401

[131] Sevilla, J. Antonio F.: *European Motor Vehicle Parc 2006*. 2008. – URL http://www.acea.be/images/uploads/files/20080129_EU_motor_vehicles_in_use_2008.pdf

[132] Shah, Sameena ; Kothari, Ravi ; Chandra, Suresh u. a.: Trail formation in ants. A generalized Polya urn process. In: *Swarm Intelligence* 4 (2010), Nr. 2, S. 145–171

[133] Stolpe, M.: Kraftfahrzeugverkehr in Deutschland. (2003). – URL http://daten.clearingstelle-verkehr.de/194/

[134] Su, Bey-Ling ; Wang, Ming-Shi ; Huang, Yueh-Ming: Fuzzy logic weighted multi-criteria of dynamic route lifetime for reliable multicast routing in ad hoc networks. In: *Expert Systems with Applications* 35 (2008), Nr. 1, S. 476–484

[135] SUMO: *Homepage*. – URL http://www.sumo-sim.org. – Zugriffsdatum: 12.12.2014

[136] Tanenbaum, Andrew S. ; Van Steen, Maarten: *Distributed systems*. Bd. 2. Prentice Hall, 2002

[137] Tarapata, Zbigniew: Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. In: *International Journal of Applied Mathematics and Computer Science* 17 (2007), Nr. 2, S. 269–287

[138] Tarjan, R. E.: Depth-first search and linear graph algorithms. In: *SIAM Journal on Computing* 1 (1972), Nr. 2, S. 146–160

[139] Tatomir, B. ; Rothkrantz, L.: Hierarchical Routing in Traffic Using Swarm-Intelligence. In: *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE*, 2006, S. 230–235

[140] TomTom Inc.: *Homepage*. – URL http://www.tomtom.com. – Zugriffsdatum: 12.12.2014

[141] TomTom Inc.: *TomTom Congestion Index 2012*. 2012. – URL http://www.tomtom.com/de_de/congestionindex. – Zugriffsdatum: 12.12.2014

[142] toolset, Veromodo T.: *Homepage*. – URL http://www.veromodo.com. – Zugriffsdatum: 12.12.2014

[143] Toth, Paolo ; Vigo, Daniele: *The vehicle routing problem*. Bd. 9. Siam, 2002

[144] Transportation, U.S. D. of: *Describing the Congestion Problem*. – URL http://www.fhwa.dot.gov/congestion/describing_problem.htm. – Zugriffsdatum: 12.12.2014

[145] Treiber, Martin ; Kesting, Arne: Verkehrsdynamik und-simulation. In: *Verkehrsdynamik und-simulation: Daten, Modelle und Anwendungen der Verkehrsflussdynamik, Springer-Lehrbuch, ISBN 978-3-642-05227-9. Springer-Verlag Berlin Heidelberg, 2010* 1 (2010)

[146] Tsitsiklis, John N. ; Stamoulis, George D.: On the average communication complexity of asynchronous distributed algorithms. In: *J. ACM* 42 (1995), März, Nr. 2, S. 382–400

[147] TSS/Aimsun: *Homepage.* – URL http://www.aimsun.com

[148] Valiant, Greg ; Roughgarden, Tim: Braess's paradox in large random graphs. In: *Proceedings of the 7th ACM conference on Electronic commerce* ACM (Veranst.), 2006, S. 296–305

[149] Vickrey, William S.: Congestion theory and transport investment. In: *The American Economic Review* 59 (1969), Nr. 2, S. 251–260

[150] Vlahogianni, Eleni I. ; Golias, John C. ; Karlaftis, Matthew G.: Short-term traffic forecasting: Overview of objectives and methods. In: *Transport reviews* 24 (2004), Nr. 5, S. 533–557

[151] Vu, Anh ; Ramanandan, Arvind ; Chen, Anning ; Farrell, Jay A. ; Barth, Matthew: Real-time computer vision/DGPS-aided inertial navigation system for lane-level vehicle navigation. In: *Intelligent Transportation Systems, IEEE Transactions on* 13 (2012), Nr. 2, S. 899–913

[152] Wardrop, John G.: Some theoretical aspects of road traffic research. In: *ICE Proceedings: Engineering Divisions* Bd. 1 Ice Virtual Library (Veranst.), 1952, S. 325–362

[153] WAZE: *Homepage.* – URL http://www.waze.com. – Zugriffsdatum: 12.12.2014

[154] Wedde, H. F. ; Lehnhoff, S. ; van Bonn, B. ; Bay, Z. ; Becker, S. ; Boettcher, S. ; Brunner, C. ; Buescher, A. ; Fuerst, T. ; Lazarescu, A. M. ; Rotaru, E. ; Senge, S. ; Steinbach, B. ; Yilmaz, F. ; Zimmermann, T.: Highly Dynamic and Adaptive Traffic Congestion Avoidance in Real-Time Inspired by Honey Bee Behavior. In: *PEARL 2007: Informatik Aktuell - Mobilität und Echtzeit.* Boppard, Germany : Springer, Dec 2007

[155] Wedde, Horst F. ; Farooq, M. ; Pannenbaecker, T. ; Vogel, B. ; Mueller, C. ; Meth, J. ; Jeruschkat, R.: BeeAdHoc: An Energy Efficient Routing Algorithm for Mobile Ad Hoc Networks Inspired by Bee Behavior. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005).* Washington DC, USA, Jun 2005, S. 153–160

[156] Wedde, Horst F. ; Farooq, M. ; Zhang, Y.: BeeHive: An Efficient Fault Tolerant Routing Algorithm under High Loads Inspired by Honey Bee Behavior. In: Dorigo, Marco (Hrsg.) ; Birattari, M. (Hrsg.) ; Blum, C. (Hrsg.) ; Gambardella, L. M. (Hrsg.) ; Mondada, F. (Hrsg.) ; Stützle, T. (Hrsg.): *Proceedings of the Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2004)* Bd. 3172. Brussels, Belgium : Spr, Sep 2004, S. 83–94

[157] Wedde, Horst F. ; Lehnhoff, Sebastian ; Senge, Sebastian: An Intelligent Traffic Management System for Coordinated Routing Decisions Inspired by Honeybee Communication. In: Fusco, G. (Hrsg.): *Proceedings of the International Conference on Models and Technologies for Intelligent Transportation Systems, Roma, Italy, 2009-06-23*, IEEE Press, 2009, S. 81–85

[158] Wedde, Horst F. ; Lehnhoff, Sebastian ; Senge, Sebastian ; Lazarescu, Anca M.: Bee inspired bottom-up self-organization in vehicular traffic management. In: *Self-Adaptive and Self-Organizing Systems, 2009. SASO'09. Third IEEE International Conference on* IEEE (Veranst.), 2009, S. 278–279

[159] Wedde, Horst F. ; Lehnhoff, Sebastian ; van Bonn, B. ; Bay, Z. ; Becker, S. ; Boettcher, Sven ; Brunner, Christian ; Buescher, A. ; Fuerst, T. ; Lazarescu, A. M. ; Rotaru, E. ; Senge, Sebastian ; Steinbach, B. ; Yilmaz, F. ; Zimmermann, T.: A Novel Class of Multi-Agent Algorithms for Highly Dynamic Transport Planning Inspired by Honey Bee Behavior. In: *ETFA '07: Proceedings of the 12th conference on emerging technologies and factory automation.* Patras : IEEE Press, Sep 2007

[160] Wedde, Horst F. ; Lehnhoff, Sebastian ; van Bonn, Bernhard ; Bay, Z. ; Becker, S. ; Boettcher, Sven ; Brunner, Christian ; Buescher, A. ; Fuerst, T. ; Lazarescu, A. M. ; Rotaru, E. ; Senge, Sebastian ; Steinbach, B. ; Yilmaz, F. ; Zimmermann, T.: Highly Dynamic and Scalable VANET Routing for Avoiding Traffic Congestions. In: *VANET 2007: Proceedings of the 4th ACM International Workshop on Vehicular Ad Hoc Networks.* Montreal, Canada : ACM Press, Sep 2007

[161] Wedde, Horst F. ; Senge, Sebastian: BeeJamA: A Distributed, Self-Adaptive Vehicle Routing Guidance Approach. In: *Intelligent Transportation Systems, IEEE Transactions on* 14 (2013), Dec, Nr. 4, S. 1882–1895. – ISSN 1524-9050

[162] Wedde, Horst F. ; Senge, Sebastian ; Lohmann, Tim ; Knobloch, Fabian: Towards hybrid simulation of self-organizing and distributed vehicle routing in large traffic systems. In: *Biomedical Engineering and Informatics (BMEI), 2011 4th International Conference on* Bd. 4, 2011, S. 2313–2321

[163] Weyns, D. ; Holvoet, T. ; Helleboogh, A.: Anticipatory Vehicle Routing using Delegate Multi-Agent Systems. In: *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, 2007, S. 87–93

[164] Wie, Byung-Wook ; Tobin, Roger L.: Dynamic congestion pricing models for general traffic networks. In: *Transportation Research Part B: Methodological* 32 (1998), Nr. 5, S. 313–327

[165] Wijetunge, Udara ; Perreau, Sylvie ; Pollok, Andre: Distributed stochastic routing optimization using expander graph theory. In: *Communications Theory Workshop (AusCTW), 2011 Australian* IEEE (Veranst.), 2011, S. 124–129

[166] Will, Florian: *Agenten-basierte Kommunikation einer Vehicle- to-Infrastructure-Architektur für ein verteiltes Online- Routingprotokoll*, TU Dortmund, Fak. Informatik, LS3 III, Diplomarbeit, 2012

[167] Wooldridge, Michael: *An introduction to multiagent systems.* Wiley, 2008

[168] Yousefi, Saleh ; Mousavi, Mahmoud S. ; Fathy, Mahmood: Vehicular ad hoc networks (VANETs): challenges and perspectives. In: *ITS Telecommunications Proceedings, 2006 6th International Conference on* IEEE (Veranst.), 2006, S. 761–766