Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Quantifizierung von Unsicherheiten in mikroskopischer Verkehrssimulation

Parga Cacheiro, Dominic

Studiengang:	Informatik
Prüfer/in:	Jun.-Prof. Dr. rer. nat. Dirk Pflüger
Betreuer/in:	Fabian Franzelin, M.Sc.
Beginn am:	9. Feburar 2017
Beendet am:	9. August 2017
CR-Nummer:	G.2; G.3; J.1; J.4; J.6

Kurzfassung

Straßenverkehr ist allgegenwärtig. Jeden Tag fahren viele Menschen mit dem Auto durch die Straßen und dabei können sehr interessante Phänomene betrachtet werden, die den Verkehrsfluss verbessern oder verschlechtern. Beim Planen und Designen von Verkehrsnetzen ist es wichtig, solche Einflüsse zu berücksichtigen. Aus diesem Grund sind Verkehrssimulationen notwendig.

Es gibt im wesentlichen zwei große Modelltypen. Die makroskopischen Modelle sehen den Verkehr als großes System und versuchen, dessen Phänomene aus einem globalen Blickwinkel zu erklären. Die mikroskopischen Modelle betrachten die Fahrzeuge einzeln, d. h. die Fahrzeuge agieren nicht von einer globalen Logikeinheit gesteuert, sondern selbstbestimmt und aus einem lokalen Blickwinkel heraus. In dieser Arbeit wird das Nagel-Schreckenberg-Modell verwendet, ein mikroskopisches Modell zur Beschreibung von Fahrverhalten auf einer einspurigen Straße, das mittels einfacher stochastischer Mittel Phänomene wie *Staus aus dem Nichts* hervorbringt. Dieses Modell wurde um eine allgemein gültige Kreuzungslogik erweitert.

Im Rahmen dieser Arbeit wird das so entstandene Verkehrsmodell um die Mehrspurigkeit ergänzt. Das beinhaltet eine Überarbeitung der bestehenden Verkehrslogik und das Einführen von Spurwechseln inklusive *Stauinversion*. Im Anschluss wird auf Implementierungsdetails eingegangen und das Modell mit realen Daten verglichen, um qualitative Aussagen über den Verkehr zu machen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Algorithmen

1 Einleitung

Straßenverkehr ist allgegenwärtig. Jeden Tag fahren viele Menschen mit dem Auto durch die Straßen und dabei können sehr interessante Phänomene betrachtet werden, die den Verkehrsfluss verbessern oder verschlechtern. Beim Planen und Designen von Verkehrsnetzen ist es wichtig, solche Einflüsse zu berücksichtigen. Aus diesem Grund sind Verkehrssimulationen notwendig. Um das zu veranschaulichen, wird im Folgenden das Braess Paradoxon beschrieben.

Figur 1.1 zeigt ein einfaches Straßennetz. Auf diesem Straßennetz möchten 6000 Fahrzeuge links starten und rechts ankommen. Dabei gibt es nur zwei mögliche Routen: entweder die Route über die Landstraße (oben, mit einem Baum veranschaulicht) und anschließend durch die Kleinstadt (rechts, durch ein Häuschen visualisiert), oder die Route durch die Großstadt (links, durch Hochhäuser dargestellt) und anschließend über die Autobahn (unten, mit einem Autobahnsymbol gekennzeichnet). Die beiden Städte trennt ein Gebirge voneinander, das zentral durch die Dreiecke dargestellt ist. Desweiteren gibt es Funktionen für die einzelnen Routenabschnitte, die die benötigte Fahrzeit in Abhängigkeit von der Anzahl Fahrzeuge einfach modellieren:

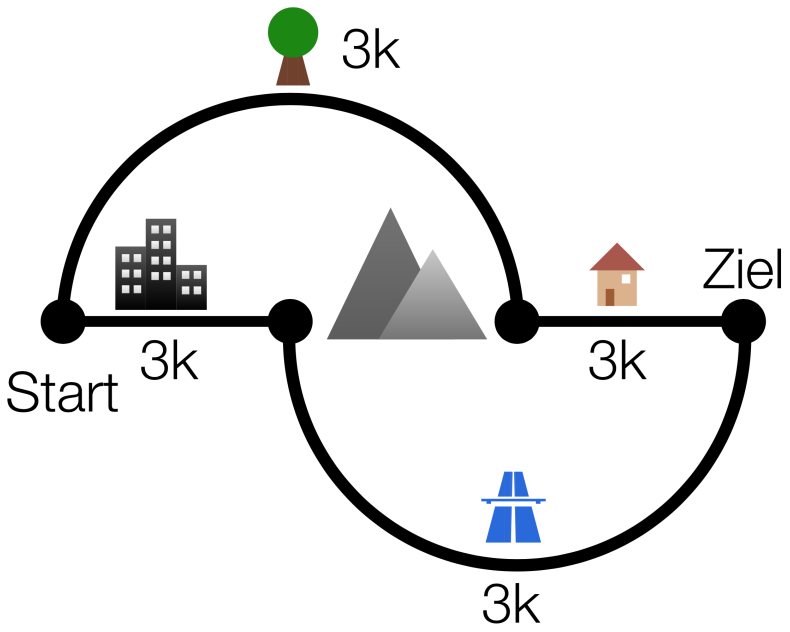


Abbildung 1.1: Ein einfaches Straßennetz zur Veranschaulichung des Braess Paradoxon

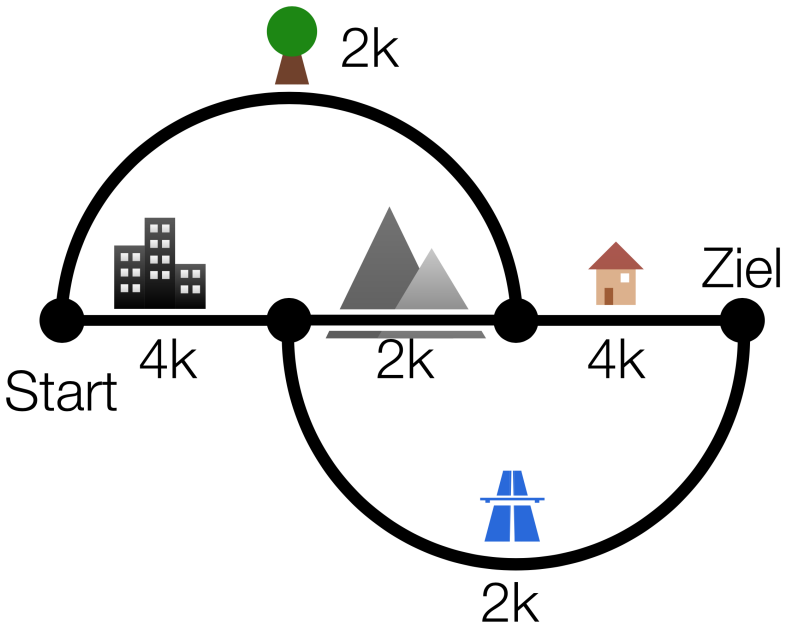


Abbildung 1.2: Ein einfaches Straßennetz zur Veranschaulichung des Braess Paradoxon nach Bau des Tunnels

Autobahn mit $30\,\text{min} + 1\,\text{min}/1000\,\text{Fahrzeuge}$

Landstraße mit $35\,\text{min} + 2\,\text{min}/1000\,\text{Fahrzeuge}$

Kleinstadt mit $7\,\text{min} + 4\,\text{min}/1000\,\text{Fahrzeuge}$

Großstadt mit $18\,\text{min} + 3\,\text{min}/1000\,\text{Fahrzeuge}$

Die Fahrt über eine Autobahn ist hier schneller absolviert als über die Landstraße, allerdings ist die Landstraße bei Verkehrsaufkommen auch schneller überlastet, weshalb hier der verkehrsabhängige Teil höher ist. Analog wird bei den Städten argumentiert: durch die Kleinstadt kommen die Fahrer schneller als durch die Großstadt, allerdings ist sie bei Verkehrsaufkommen schneller überlastet als die Großstadt, die dafür besser ausgelegt ist. Aufgrund von guten Informationsmöglichkeiten über Radio usw. wird sich auf den beiden Routen ein Gleichgewicht bilden, bei dem die Fahrzeit über beide Routen gleich lang ist. Schließlich wird jeder Fahrer, der mitbekommt, dass seine Route langsamer ist, beim nächsten Mal auf die schnellere Route wechseln. Es bildet sich ein Gleichgewicht bei den in der Abbildung gezeigten $3000\,\text{Fahrzeugen}$ pro Route mit einer Fahrzeit von $60\,\text{min}$. Jede andere Verteilung auf die Routen würde dafür sorgen, dass eine Route schneller und die andere Route langsamer als $60\,\text{min}$ ist und aus den eben genannten Gründen würde sich der Verkehr ausgleichen.

Die beiden Städte vereinbaren, einen Tunnel durch das Gebirge zu bauen, um die Fahrzeit zu verkürzen. Der Tunnel ist super gebaut und ist vom aktuellen Verkehr vollkommen unabhängig.

Tunnel mit 9 min

Obwohl das Verkehrsnetz jetzt um einen sehr zeitsparenden Tunnel erweitert wurde, stellt sich mit der in Abbildung 1.2 gezeigten Fahrzeugverteilung heraus, dass die Fahrzeit jeder Route bei 62 min liegt, also höher als zuvor. Die Ursache liegt darin, dass durch den Tunnel die Städte mehr belastet werden, die den Flaschenhals des Systems bilden.

Das Phänomen lässt sich auch anders herum beobachten, dass bei Auslassen einer Straße der umliegende Verkehrsfluss besser wird. Dieses nur sehr simple Beispiel zeigt, wie komplex diese Phänomene auf realen Straßennetzen sein können. Simulationen bieten hier den Vorteil, beliebige Szenarien ohne Einfluss auf den realen Verkehr testen zu können.

Im Vorfeld dieser Arbeit wurden einige Paper auf deren beschriebene Verkehrslogik durchgeschaut. Dabei fällt auf, dass der Fokus auf der Untersuchung des Verkehrsflusses auf einer Straße liegt. In [BZBP09] wird der Leser sehr schön und ausführlich an das Thema der mikroskopischen Verkehrssimulation herangeführt. Es werden für den Verkehr charakteristische Eigenschaften definiert und der Verkehrsfluss wird daraufhin analysiert und mit realen Daten verglichen, woraus wieder Erkenntnisse für die verwendeten Simulationsparameter geschlossen werden. In [TK11] wird ebenfalls der Verkehrsfluss untersucht. Hier wird auf unterschiedliche Typen an Fahrzeugen/Fahrern und deren Einfluss auf den Verkehrsfluss untersucht. Beispiele für solche Typen sind der normale Verkehr verglichen mit öffentlichen Verkehrsmitteln oder Lastkraftwagen.

Der Verkehrsfluss auf einer Straße wird sehr ausführlich untersucht. Die logische Modellierung der Kreuzungen wird allerdings immer auf randomisierte Vorfahrt oder wie in [BZBP09] und [MicSimUrbanTraffic9797] auf Ampel-ähnliche Phasenmodelle beschränkt, in denen abwechselnd kollisionsfrei Fahrzeuge bestimmter Straßen die Kreuzung gleichzeitig überqueren dürfen.

In dieser Arbeit soll der Fokus nicht auf der Analyse von Verkehr und Verkehrsflüssen liegen, die auf eine oder wenige Straßen beschränkt ist. Ziel dieser Arbeit ist es, die in [MTS16] gegebene, mikroskopische, einspurige Kreuzungslogik herzunehmen und mehrspurig auszubauen. Dazu gehört die Logik für einen realistischen Spurwechsel inklusive realen Verkehrsphänomenen wie *Staus aus dem Nichts* oder *Stauinversion*.

Im Anschluss an die Theorie wird auf Implementierungsdetails und größere Hürden bei der Implementierung ausgeschweift. Dabei geht es um die Performanz der Echtzeit-Simulation, um die nebenläufige Implementierung der Kreuzungslogik und Ideen zur weiteren Optimierung.

Als letztes soll ein gewähltes Verkehrsszenario mit dem entsprechenden Szenario aus dem realen Alltag unter Verwendung von Methoden der Unsicherheitsquantifizierung verglichen werden. In dieser Arbeit werden über die Ergebnisse qualitative Aussagen getroffen und die Simulation unter dem verwendeten erweiterten Nagel-Schreckenberg-Modell bewertet.

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Mikroskopische Verkehrssimulation: Die in [MTS16] eingeführte Verkehrslogik wird in diesem Kapitel erläutert, mit Algorithmen veranschaulicht und anschließend auf ein mehrspuriges Straßennetz angepasst.

Kapitel 3 – Microtrafficsim - Der Verkehrssimulator: Während es im vorherigen Kapitel um die Logik in ihrer Theorie ging, wird in diesem Kapitel mehr auf die Implementierungsdetails eingegangen.

Kapitel 4 – Verkehr mit Unsicherheitsquantifizierung: Die Funktion des gewählten Modells und des implementierten Simulators soll hier mit Hilfe von Methoden der Unsicherheitsquantifizierung untersucht werden. Dazu wird ein Verkehrsszenario definiert, das mit unterschiedlichen Werten für die gewählten Eingabeparameter simuliert wird. Der Einfluss dieser Parameter auf den Verkehr wird gemessen und qualitativ bewertet.

Kapitel 5 – Zusammenfassung und Ausblick Die Ergebnisse dieser Arbeit und während der Arbeit gesammelte Erkenntnisse werden noch einmal aufgegriffen und abschließend beschrieben.

2 Mikroskopische Verkehrssimulation

Dieses Kapitel beschäftigt sich mit den theoretischen Grundlagen, auf denen die Verkehrssimulation beruht. Zunächst werden grundlegende Begriffe und Vorüberlegungen erläutert, z. B. den Aufbau des verwendeten Straßennetzes. Anschließend wird das Verhalten der Fahrzeuge jeweils auf der Straße und an einer Kreuzung beschrieben.

2.1 Wichtige Konventionen

Für den Straßenverkehr gilt je nach Ort „Rechts-vor-Links“ oder „Links-vor-Rechts“ . Davon abhängig wird entweder auf der rechten oder auf der linken Seite überholt. Um auf diese Unterscheidung verzichten zu können, wird in dieser Arbeit von ”äußerer” und ”innerer” Straßenspur gesprochen. Im Fall ”Rechts-vor-Links” ist die rechteste Spur die äußerste Spur, im Fall ”Links-vor-Rechts” ist es die linkeste.

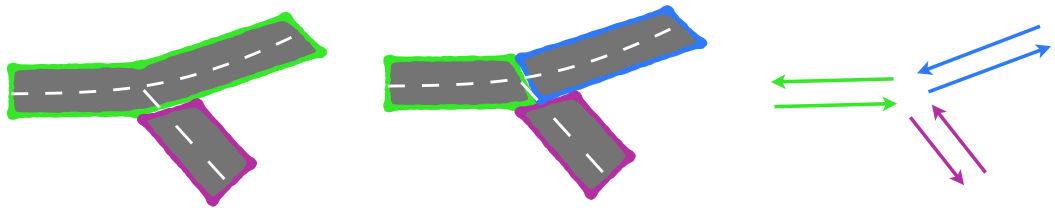
Desweiteren werden in der Arbeit die Begriffe *Simulation* oder *Szenario* verwendet. Eine Simulation ist die Berechnung eines gewählten Szenarios, das eine Beschreibung von Eingabeparametern darstellt.

2.2 Aufbau des Straßengraphen

Straßendaten und Vorfahrtseigenschaften können von [OSM17] heruntergeladen werden. Diese Daten werden in ein Straßennetz übersetzt. Ein *Straßennetz* ist als gerichteter Graph modelliert. Für die Verkehrssimulation werden überall dort Knoten generiert, wo

- zwei Straßen kreuzen, die unterschiedliche Anzahl an Spuren haben oder
- mindestens drei Straßen kreuzen.

Das bedeutet, dass eine Kante des Straßengraphen nur einen Teil und nicht zwingend eine komplette Straße im umgangsprachlichen Sinne repräsentiert. Außerdem sind die Kanten gerichtet, Straßen werden also pro Richtung separat als Kante gespeichert.



(a) T-Kreuzung aus der Realität **(b)** T-Kreuzung nach der Knotenbildung **(c)** T-Kreuzung nach der Knoten- und Kantenbildung

Abbildung 2.1: Abbildung 2.1a zeigt zwei Straßen im herkömmlichen Sinn. Die Mitte zeigt drei Straßen nach der Knotenbildung und und die letzte Abbildung zeigt die daraus entstandenen Kanten.

2.3 Mikroskopisches Nagel-Schreckenberg-Modell

Um auf dem Straßengraphen Fahrzeuge fahren lassen zu können, muss das Modell definiert werden, welches das Verhalten der Fahrzeuge beschreibt. In dieser Arbeit wird hierfür das Nagel-Schreckenberg-Modell [NaSch92] als Grundlage verwendet und erweitert. Es ist ein mikroskopischer Ansatz, d. h. die Fahrzeuge agieren nicht von einer globalen Logikeinheit gesteuert, sondern einzeln, selbstbestimmt und aus einem lokalen Blickwinkel heraus.

2.3.1 Kanten als einspuriger zellulärer Automat

Das Nagel-Schreckenberg-Modell betrachtet die Kanten einspurig und als einen zellulären Automaten. Ein zellulärer Automat ist ein theoretisches Konstrukt aus Zellen, die bestimmte Zustände annehmen können. Bei der Verkehrssimulation sind diese Zustände binär, was so interpretiert wird, dass die Zelle von Fahrzeugen besetzt oder unbesetzt sein kann. Jede Kante wird abhängig von ihrer Länge (in Metern) in Zellen diskretisiert. Eine Zelle misst dabei $7{,}5\,\mathrm{m}$, was laut dem Modell einer Fahrzeuglänge mit Sicherheitsabstand entspricht. Bei einer Länge unter $7{,}5\,\mathrm{m}$ wird aufgerundet. Zusätzlich zu den Zellen und Zuständen gehört zu einem zellulären Automaten noch eine Übergangsfunktion. Diese beschreibt den Übergang des Zustands einer Zelle in einen anderen Zustand, was hier mit einem Voranschreiten der Zeit gleichgesetzt

Abbildung 2.2: Eine Kante, aufgeteilt in Zellen. Jede Zelle darf maximal ein Fahrzeug beinhalten. Die Fahrzeuge haben diskrete Geschwindigkeiten in $\mathrm{Zellen/s}$.

wird. Jeder Zeitschritt im Nagel-Schreckenberg-Modell misst eine Sekunde. Für die Übergangsfunktion wird für die Fahrzeuge eine Geschwindigkeit v in $\mathrm{Zellen/s}$ modelliert. Eine Zelle pro Zeitschritt entspricht demnach einer Geschwindigkeit von $7{,}5\,\mathrm{m/s} = 27\,\mathrm{km/h}$. So können für (in Deutschland) gängige Geschwindigkeitsbegrenzungen diskrete Modellgeschwindigkeiten festlegen:

$1\,\mathrm{Zellen/s} = 27\,\mathrm{km/h} \approx 30\,\mathrm{km/h}$
$2\,\mathrm{Zellen/s} = 54\,\mathrm{km/h} \approx 50\,\mathrm{km/h}$
$3\,\mathrm{Zellen/s} = 81\,\mathrm{km/h} \approx 70\,\mathrm{km/h}$
$4\,\mathrm{Zellen/s} = 108\,\mathrm{km/h} \approx 100\,\mathrm{km/h}$
$5\,\mathrm{Zellen/s} = 135\,\mathrm{km/h} \approx 130\,\mathrm{km/h}$

2.3.2 Ein Zeitschritt im Nagel-Schreckenberg-Modell

Ein Zeitschritt im Nagel-Schreckenberg-Modell besteht aus vier Teilschritten [NaSch92, Seite 2222], die im Folgenden erläutert werden und in Abbildung 2.3 dargestellt sind.

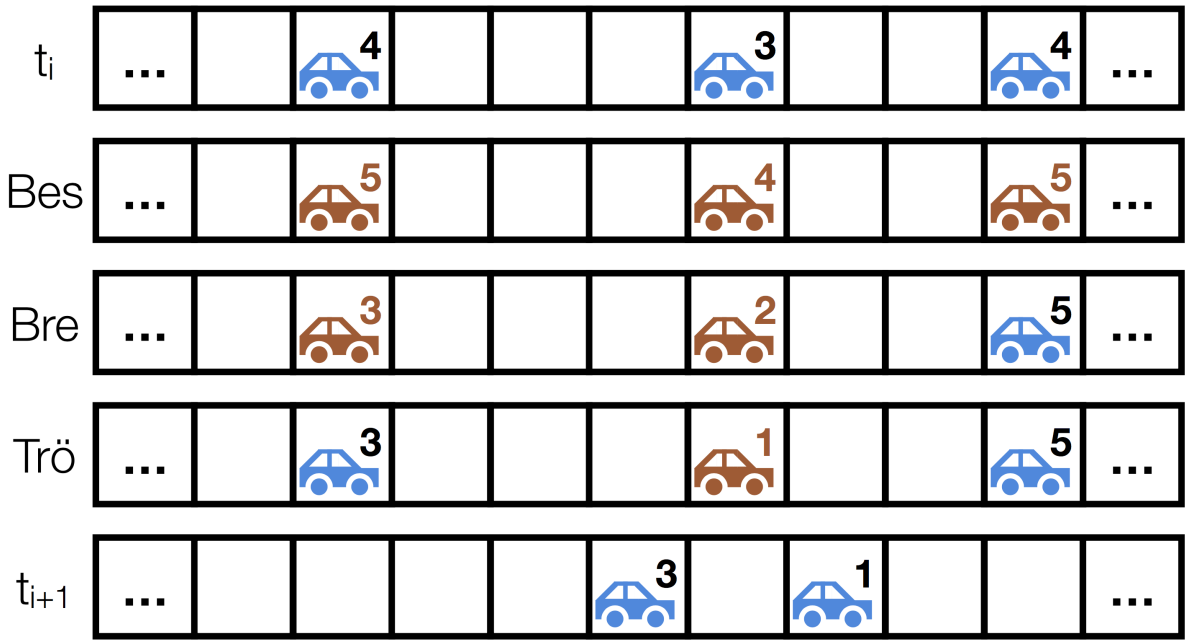


Abbildung 2.3: Gezeigt ist ein Zeitschritt im Nagel-Schreckenberg-Modell. Die Fahrzeuge im hellen rot-Ton sind diejenigen, die ihre Attribute im aktuellen Zeitschritt gemäß dem Modell verändert haben.

1. **Beschleunigung:** Jedes Fahrzeug i beschleunigt unabhängig voneinander. $v_{\max}$ ist dabei eine Größe, die von der Maximalgeschwindigkeit der aktuellen Kante abhängt.
   $\forall i\colon v_i \leftarrow \min\{v_i + 1,\ v_{\max}\}$
2. **Bremsen:** Wenn ein vorausfahrendes Fahrzeug vorhanden ist, dann wird so gebremst, dass eine Kollision vermieden wird. So werden Kollisionen vermieden.
   $\forall i\colon v_i \leftarrow \min\{\text{Distanz zum vorausfahrenden Fahrzeug},\ v_i\}$
3. **Trödeln:** Mit einer Wahrscheinlichkeit $0 \leq p_{\text{Trödeln}} < 1$ wird das Fahrzeug stärker als nötig abgebremst.
   $\forall i\colon v_i \leftarrow \max\{0,\ v_i - 1\}$
4. **Bewegen:** Jedes Fahrzeug i bewegt sich v_i Zellen weiter.

Das stochastische Trödeln ist hier ein sehr wichtiger Teilschritt. Es modelliert nichtdeterministisches Fahrverhalten, bedingt durch Trägheit oder Unaufmerksamkeit des Fahrers. Dieses nichtdeterministische Verhalten ist Ursache für *Staus aus dem Nichts*: ein Fahrer bremst mehr als nötig ab oder beschleunigt zu spät nach einem Bremsvorgang, der hintere Fahrer hält womöglich noch zu geringen Sicherheitsabstand und muss abrupt abbremsen. Bei ausreichender Verkehrsdichte entsteht so ein Stau. Weitere Informationen dazu finden sich im Originalpaper [NaSch92, Seite 2225].

2.4 Kreuzungslogik als Erweiterung des Nagel-Schreckenberg-Modells

Das Nagel-Schreckenberg-Modell beschreibt die Logik zwischen zwei Knoten sehr gut. Allerdings deckt es nicht die Interaktion der Fahrzeuge an Kreuzungen ab. Hierfür wird sich der beschriebenen Logik aus [MTS16, Seite 5-7] bedient. Entsprechend gibt dieses gesamte Unterkapitel bis auf wenige Ergänzungen den Inhalt sinngemäß wieder.

2.4.1 Bremsvorgang

Ein wichtiger Grundgedanke bei der Kreuzungslogik ist der, dass die gesamte Komplexität in der Bremsfunktion des Nagel-Schreckenberg-Modells konzentriert wird. Das hat den Vorteil, dass die restlichen Funktionen einfach auszuführen sind. Vorher, für die Beschleunigung, werden nur Attribute angepasst. Nach dem Bremsen aktualisieren die Fahrzeuge nur noch die Position, was sich allein auf die zugrundeliegende Datenstruktur beschränkt. Ein Pseudocode für den Bremsvorgang ist in Algorithmus 2.1 dargestellt. Neben Abschnitten, die aus dem Nagel-Schreckenberg-Modell stammen (Zeilen 5 und 25), regelt der Großteil der Prozedur die Kollisionsvermeidung beim Überqueren der Kreuzung.

Algorithmus 2.1 Logik für das Bremsen eines Fahrzeugs

```
1:  procedure BREMSEN()
2:      mussBremsen ← true
3:
4:      if ¬istVorderstesFahrzeug() then                    // bremst für vorderes Fahrzeug
5:          distanz ← vorderfahrzeug.zellenposition − zellenposition
6:      else
7:          distanz ← spur.länge − zellenposition           // bremst für das Ende der Straße
8:
9:          if geschw ≥ distanz then
10:             if ¬istDieRouteBeendet() ∧ hatVorfahrt() then
11:                 if nächsteSpur.hatFahrzeuge() then
12:                     tmp ← nächsteSpur.erstesFahrzeug.zellenposition
13:                 else
14:                     tmp ← nächsteSpur.länge
15:                 end if
16:
17:                 distanz ← distanz + tmp
18:             end if
19:         else
20:             mussBremsen ← false                         // ohne Probleme fahren
21:         end if
22:     end if
23:
24:     if mussBremsen then
25:         geschw ← min{geschw, distanz − 1}
26:     end if
27: end procedure
```

Der Pseudocode zeigt, dass nur die vordersten Fahrzeuge einer Kante einen Knoten passieren und damit die Kante wechseln dürfen. Alle anderen müssen auf das vorausfahrende Fahrzeug bremsen (Zeile 4). Desweiteren wird nicht auf das Ende der aktuellen Kante gebremst, ohne den Platz auf der nachfolgenden Kante zu prüfen (Zeile 10-18). Die Fahrzeuge können also ungehindert fahren, wenn sie „freie Fahrt“ haben. Das ist z. B. auf Autobahnen ein sehr wichtiger Aspekt.

2.4.2 Registrierung im Knoten

„Ein Knoten handhabt alle Konflikte zwischen Fahrzeugen, die ihn überqueren möchten.“ [MTS16, frei übersetzt nach Seite 5, Abschnitt 5.3.2, erster Satz] Dabei wird die Handhabung in zwei Schritte unterteilt.

Der erste Aspekt ist die Registrierung und Deregistrierung von Fahrzeugen im Knoten. Nähert sich das vorderste Fahrzeug einer Kante dem Knoten, so meldet es sich am Knoten an. Dieser Vorgang findet im Bewegungsschritt (nach dem Trödeln) im Nagel-Schreckenberg-Modell statt. „Nah genug“ ist in diesem Fall, wenn das Fahrzeug mit der aktuell erlaubten Maximalgeschwindigkeit den Knoten erreichen kann.

Bei der Registrierung wird das Fahrzeug mit bereits registrierten Fahrzeugen paarweise verglichen. Ein Vergleich basiert auf den Vorfahrtsregeln und ist in Algorithmus 2.2 schematisch beschrieben. Jedes Fahrzeug hat einen Parameter $prioritätszähler$. Für jedes Fahrzeug, dem Vorfahrt gewährt werden muss, wird dieser Zähler dekrementiert. In jedem anderen Fall wird der Zähler inkrementiert. Insbesondere wird er auch bei Gleichstand, d. h. wenn sich die Wege der beiden Fahrzeuge über den Knoten nicht kreuzen, inkrementiert. Sei c die Anzahl der registrierten Fahrzeuge in einem Knoten, dann werden folgende zwei Fälle unterschieden:

1. Gilt für ein Fahrzeug $prioritätszähler = c - 1$, so hat dieses am höchsten priorisierte Fahrzeug gegenüber allen anderen Fahrzeugen Vorfahrt. Wird diese Bedingung vorausgesetzt, dann können durch das Inkrementieren bei Gleichstand auch mehrere Fahrzeuge gleichzeitig fahren. Gäb es bei einem dieser Fahrzeuge eine Kollision, so hätte eins der beteiligten Fahrzeuge einen kleineren $prioritätszähler$ und wäre nicht mehr am höchsten priorisiert.
2. Es kann auch vorkommen, dass mehrere höchst priorisierten Fahrzeuge den selben $prioritätszähler < c-1$ haben. In diesem Fall hat der Knoten einen Deadlock. Das kann an einer Kreuzung vorkommen, an der alle beteiligten Fahrzeuge dem Nebenfahrzeug wegen „Rechts-vor-Links“ Vorfahrt gewähren müssen. Um diesen Deadlock zu lösen, wird unter den am höchsten priorisierten Fahrzeugen zufällig eins gewählt, dem Vorfahrt gewährt wird.

Im Algorithmus 2.2 wird cmp berechnet und zurückgegeben. cmp ist im Allgemeinen ein Indikator dafür, wie die beiden Vergleichsobjekte zueinander stehen. Unter der definierten Vergleichsoperation bedeutet $cmp = 0$, dass $fahrzeug_0$ und $fahrzeug_1$ gleich sind; $cmp > 0$ bedeutet, dass $fahrzeug_0 > fahrzeug_1$ gilt und $cmp < 0$, dass $fahrzeug_0 < fahrzeug_1$ gilt.

Wenn ein Fahrzeug einen Knoten überquert, dann fährt es einen imaginären Weg ab. Der Algorithmus prüft zuerst, ob sich die Wege der übergebenen Fahrzeuge kreuzen. Im Abschnitt 2.4.3 wird näher erläutert, was ein Weg ist und wie bestimmt wird, ob sich zwei Wege kreuzen. Wenn sich zwei Wege nicht kreuzen, so können beide Fahrzeuge ohne gegenseitigen Einfluss den Knoten überqueren. Ist dem nicht so, muss bestimmt werden, welches Fahrzeug Vorfahrt hat. Zunächst wird die aktuelle Kante der Fahrzeuge verglichen. Fährt ein Fahrzeug auf einer Vorfahrtsstraße, das andere nicht, dann hat das Fahrzeug auf der Vorfahrtsstraße auf jeden Fall Vorfahrt. Kommen beide von einer Vorfahrtsstraße, dann muss das Ziel verglichen werden. Ein Fahrzeug, das die Vorfahrtsstraße verlassen möchte, muss einem Fahrzeug Vorfahrt gewähren, das auf der Vorfahrtsstaße bleiben will. Ist das ebenfalls identisch, dann wird entweder zufällig oder nach „Rechts-vor-Links“ entschieden.

Algorithmus 2.2 Vergleich zweier Fahrzeuge nach den Verkehrsregeln

```
1:  function VERGLEICHE(fahrzeug_1, fahrzeug_2)
2:      weg_1 ← BESTIMMEKREUZUNGSWEG(fahrzeug_1)
3:      weg_2 ← BESTIMMEKREUZUNGSWEG(fahrzeug_2)
4:
5:      if WEGEKREUZENSICH(weg_1, weg_2) then
6:          cmp ← weg_1.startkante.priorität − weg_2.startkante.priorität
7:
8:          if cmp = 0 then                          // beide kommen von einer Hauptstraße
9:              cmp ← weg_1.zielkante.priorität − weg_2.zielkante.priorität
10:
11:             if cmp = 0 then                      // beide wollen auf eine Hauptstraße
12:                 if Vorfahrt ist zufällig then
13:                     cmp ← RANDOM({−1, 1})        // nicht 0, da sich Wege kreuzen
14:                 else
15:                     cmp ← BERECHNERECHTSVORLINKS(weg_1, weg_2,)
16:                 end if
17:             end if
18:         end if
19:     else
20:         cmp ← 0                                  // beide dürfen fahren
21:     end if
22:
23:     return cmp
24: end function
```

2.4.3 Kreuzungsindizes zur Berechnung der Vorfahrt bei kreuzenden Wegen

Der Aufbau von Kreuzungen kann beliebig komplex sein. Zu versuchen, diese Kreuzungen zu klassifizieren, ist eine viel zu komplexe Herangehensweise an die Kreuzungslogik. Aus diesem Grund hat sich [MTS16] das Ziel gesetzt gehabt, eine Kreuzungslogik zu entwickeln, die für jeden möglichen Kreuzungsaufbau korrekt ist, ohne den Aufbau zu klassifizieren. Hierfür werden die Vektoren benötigt, mit denen die Kanten verknüpft sind. Diese geben die Richtung einer Kante an, mit der sie bei einem Knoten ankommt bzw. diesen verlässt. Mit Hilfe dieser Vektoren lassen sich die Kanten in einem Vorverarbeitungsschritt im Uhrzeigersinn (oder dagegen) sortieren. Dabei ist zu beachten, dass die Vektoren entweder aller eingehenden oder aller ausgehenden Kanten umgedreht werden, damit die Sortierung korrekt ist. Es wird sich herausstellen, dass die Sortierrichtung ausreichen wird, um in der Logik zwischen „Rechts-vor-Links“ (gegen den Uhrzeigersinn) und „Links-vor-Rechts“ (mit dem Uhrzeigersinn) zu

Algorithmus 2.3 Berechnung eines Fahrzeug-Weges über einen Knoten

```
1:  function BESTIMMEKREUZUNGSWEG(fahrzeug)
2:      weg ← [ ]                                   // der Weg ist ein leeres Array
3:      start ← fahrzeug.startkante.index
4:      ziel ← fahrzeug.zielkante.index
5:      i ← start
6:
7:      while i ≠ ziel do
8:          weg ← weg + [i]
9:          i ← (i + 1)  mod m                      // m ist die Anzahl aller Kanten im Knoten
10:     end while
11:     weg ← weg + [ziel]
12:
13:     return weg
14: end function
```

unterscheiden. Der Übersichtlichkeit halber wird im Folgenden automatisch von „Rechts-vor-Links“ gesprochen. „Links-vor-Rechts“ funktioniert analog.

Jeder Knoten assoziiert jede seiner Kanten anhand der Sortierung aufsteigend mit einem eindeutigen, ganzzahligen Index. Mit Hilfe dieser Indizes kann nun das Index-Array weg eines Fahrzeugs über den Knoten beschrieben werden, wie in Algorithmus 2.3 dargestellt. „Bildlich gesprochen entspricht der Inhalt der Variablen weg einem Lauf um den Knoten gegen den Uhrzeigersinn, beginnend bei $fahrzeug.startkante$ und endend bei $fahrzeug.zielkante$.“ [MTS16, frei übersetzt nach Seite 6, Abschnitt 5.3.3] Zur besseren Übersicht wird auf die Datenstruktur zur Speicherung der Indizes im Knoten verzichtet und die Attribute $startkante$ und $zielkante$ dem Auto zugeschrieben, die direkt Zugriff auf den Index ermöglichen.

Mit Hilfe der berechneten Wege können die Funktionen WEGEKREUZENSICH(weg_1, weg_2) und BERECHNERECHTSVORLINKS$(weg_1, weg_2,)$ definiert werden.

Seien $u_0 := weg_1[0] = start$ und $u_n := ziel$, analog für weg_2 dann w_0 und w_m. Dann wird die Funktion WEGEKREUZENSICH(weg_1, weg_2) durch einen deterministischen, endlichen Automaten (DEA) in Abbildung 2.4 beschrieben. Hinter dem Automaten steckt folgender Grundgedanke: Sei k das Weg-Array, welches mit BESTIMMEKREUZUNGSWEG$(fahrzeug)$ berechnet wird, wenn bei u_0 gestartet und bei $u_0 + (m-1) \mod m$ gestoppt wird. Dann ist aufgrund der Straßengeometrie genau eine der folgenden Aussagen gültig.

1. Kreuzen sich die beiden Wege weg_1 und weg_2, so kann bei u_0 gestartet werden und vor Erreichen von u_n würde entweder w_0 oder w_m besucht werden. Mit Entfernen von irrelevanten Kanten ergibt sich $k' = u_0 w_{\{0,m\}} u_n w_{\{0,m\}}$.
2. Kreuzen sich die beiden Wege weg_1 und weg_2 nicht, so kann bei u_0 gestartet und als nächstes würde entweder u_n direkt oder zuvor $w_{\{0,m\}} w_{\{0,m\}}$ erreicht werden. Mit

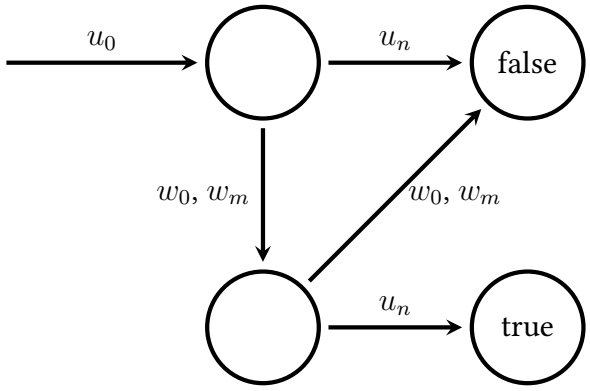


Abbildung 2.4: Ein DEA zur Bestimmung, ob sich zwei Wege kreuzen.

Entfernen von irrelevanten Kanten ergibt sich hier $k'' = u_0 u_n w_{\{0,m\}} w_{\{0,m\}}$ bzw. $k'' = u_0 w_{\{0,m\}} w_{\{0,m\}} u_n$.

Es ist zu beachten, dass bei zwei Fahrzeugen mit der selben Zielkante ebenfalls eine Kreuzung der Wege erkannt werden muss. Zwei Fahrzeuge mit der selben Startkante gibt es nicht, denn es darf sich pro Kante nur ein Fahrzeug registrieren. Da die Kanten gerichtet sind, ist es außerdem nicht möglich, dass die Zielkante eines Fahrzeugs mit der Startkante des anderen Fahrzeugs zusammenfällt.

Als letztes fehlt noch die Erklärung zu der Funktion BERECHNERECHTSVORLINKS(weg_1, weg_2,), welche sich stark an der Erklärung in [MTS16, Seite 6] orientieren wird. Die Funktion wird nur aufgerufen, wenn sich die beteiligten Wege weg_1 und weg_2 schneiden. Daraus folgt direkt, dass die beiden Wege ein eindeutiges gemeinsames Pattern haben. Der Grund dafür liegt in der Konstruktion der Wege:

- In einem Weg gibt es keine doppelten Indizes.
- Ein Weg ist für gegebenen Start- und Zielindex eindeutig konstruiert.
- Die Länge eines Weges ist limitiert.

Seien $weg_1 := u_0 \dots u_n$ und $weg_2 := w_0 \dots w_m$. Angenommen, die Reihenfolge der Indizes beim Ablaufen aller Kanten ist $u_0 w_0 u_n w_m$, dann ist das gemeinsame Pattern $w_0 u_n$. Weil w_0 der erste Index im Pattern ist und zu $fahrzeug_2$ gehört, hat $fahrzeug_2$ Vorfahrt. Bildlich gesprochen: Wird weg_1 gegen den Uhrzeigersinn um den Knoten abgelaufen, dann wird zuerst die Startkante von weg_2 getroffen, bevor die Zielkante von weg_1 erreicht wird. Zur Validierung wird in zwei Fälle unterschieden, die in Abbildung 2.5 dargestellt sind. Im ersten Fall kommt $fahrzeug_2$ von rechts (es gilt „Rechts-vor-Links“). In diesem Fall muss $fahrzeug_1$ Vorfahrt gewähren. Kommt $fahrzeug_2$ von links, dann bedeutet das, die Zielkante von $fahrzeug_1$ läge links und $fahrzeug_1$ müsse wieder Vorfahrt gewähren, da es ein Linksabbieger ist.

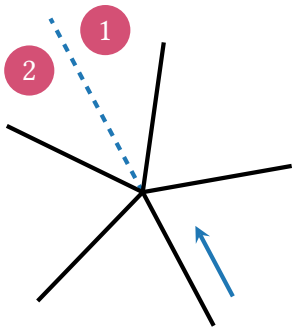


Abbildung 2.5: „Ein Fahrzeug, das Vorfahrt gewähren muss, kommt von unten rechts. Beim Vergleich mit einem priorisierten Fahrzeug wird bei dessen Herkunft zwischen den beiden gezeigten Spezialfällen unterschieden.“ [MTS16, frei übersetzt nach Seite 7, Abschnitt 5.3.3]

2.5 Routenberechnung mittels A*-Suche

Die Fahrzeuge fahren eine vorher berechnete Route ab. Eine *Route* entspricht einer geordneten Liste von Kanten, die nacheinander abgefahren werden. Demnach verläuft eine Route von Knoten zu Knoten. "Die Routen werden mit Hilfe der A*-Suche [HNR68] gefunden, was vor allem bei großen Straßennetzen deutlich schneller ist als ein Dijkstra. Eine Version der A*-Suche findet die kürzeste Route, die andere die schnellste anhand der Kantenlänge und ihrer maximal erlaubten Geschwindigkeit."[MTS16, Seite 5]

2.6 Erweiterung des bisherigen Modells auf Mehrspurigkeit

Das einspurige Nagel-Schreckenberg-Modell, welches in [MTS16] um eine Kreuzungslogik ergänzt wurde, ist im Rahmen dieser Bachelorarbeit auf Mehrspurigkeit erweitert worden. In der Theorie kommt nun hinzu, dass Fahrzeuge auf unterschiedlichen Spuren derselben Kante miteinander interagieren müssen, um einen Spurwechsel vollziehen zu können. Um das zu realisieren, wurde das Nagel-Schreckenberg-Modell um einen Zwischenschritt erweitert. Algorithmus 2.4 zeigt diesen Zwischenschritt und wird direkt nach dem Beschleunigen ausgeführt.

Es wird auf manche Details verzichtet, um die elementare Logik in den Vordergrund zu heben. Zu den verzichteten Details zählen zum einen Informationen über die Verarbeitung in der hinterlegten Datenstruktur. Zum anderen sind manche Abfragen ausgelassen worden. Manchmal wird das vorausfahrende Fahrzeug ermittelt, ohne zu prüfen, ob man nicht selbst das vorderste Fahrzeug der Kante ist, da sich in diesem Fall die gesamte Rechnung erübrigt.

Algorithmus 2.4 Logik für das Spurwechseln eines Fahrzeugs - Teil 1

```
1:  param spur : aktuelle Spur, auf der sich dieses Fahrzeug befindet
2:  param korrekteSpur : äußerste Spur, die noch zur gewünschten Folgekante führt
3:  param p_Spurwechsel : Wahrscheinlichkeit für den Versuch, die Spur zu wechseln

4:  procedure WECHSELTSPUR()
5:      if RANDOM() < p_Spurwechsel then
6:          if SOLLTESICHRICHTIGEINORDNEN() then
7:              TENDIERTZURÄUSSERSTENSPUR()        // ⇔ Tendiert zum „Rechtsfahrgebot“
8:          else
9:              TENDIERTZURINNERSTENSPUR()         // ⇔ Tendiert zum Überholen
10:         end if
11:     end if
12: end procedure

13: function SOLLTESICHRICHTIGEINORDNEN()
14:     distanz ← spur.länge − zellenposition
15:     return maxGeschw ≥ distanz
16: end function
```

Außerdem wird hier nicht geprüft, ob das Routenziel erreicht wird. In diesem Fall ist die Spur immer korrekt.

2.6.1 Spurwechsel

Auch hier wird wieder eine Wahrscheinlichkeit, $p_{\text{Spurwechsel}}$, eingeführt. Sie modelliert die Bereitschaft eines Fahrzeugs zu überholen. Es lässt sich weiterhin im Code erkennen, dass das Spurwechseln im Wesentlichen zwischen der Situation an einem Knoten und der Situation mitten auf der Straße unterscheidet. Die Unterscheidung erfolgt danach, ob ein Fahrzeug mit der maximalen Geschwindigkeit den Knoten erreichen kann.

Die Spur kann auf zwei verschiedene Arten gewechselt werden.

- TENDIERTZURÄUSSERSTENSPUR()
  Kurz vor oder direkt am Knoten soll die Spur so gewechselt werden, dass das Fahrzeug sich korrekt einordnet. Dabei tendieren die Fahrzeuge zur äußersten Spur, die noch für die eigene Route korrekt ist. Im Falle, dass zu weit innen gefahren wird, prüft das Fahrzeug, ob es das potentiell vordere Fahrzeug der äußeren Nachbarspur überholen möchte. Falls nicht, wird die Spur nach außen gewechselt.
- TENDIERTZURINNERSTENSPUR()
  Ist das Fahrzeug nicht nahe beim nächsten Knoten, so tendiert es zum Überholen. Beim

Algorithmus 2.5 Logik für das Spurwechseln eines Fahrzeugs - Teil 2

```
17: procedure TENDIERTZURINNERSTENSPUR()
18:     if MÖCHTEÜBERHOLEN(vorderfahrzeug) then
19:         WECHSELTZURINNERENSPUR()
20:     else
21:         TENDIERTZURÄUSSERSTENSPUR()
22:     end if
23: end procedure

24: procedure TENDIERTZURÄUSSERSTENSPUR()
25:     if spur ist weiter innen als korrekteSpur then
26:         nebenspur ← äußere Nachbarspur
27:         nebenfahrzeug ← nebenspur.VORDERESFAHRZEUGVON(zellenposition)
28:
29:         if ¬ MÖCHTEÜBERHOLEN(nebenfahrzeug) then
30:             WECHSELTZURÄUSSERENSPUR()
31:         end if
32:     else if spur ist weiter außen als korrekteSpur then
33:         WECHSELTZURINNERENSPUR()
34:     end if
35: end procedure

36: function MÖCHTEÜBERHOLEN(nebenfahrzeug)
37:     distanz ← nebenfahrzeug.zellenposition − zellenposition
38:
39:     if (maxGeschw > distanz then
40:         if geschw > nebenfahrzeug.geschw) then
41:             return true
42:         end if
43:     end if
44:
45:     if nebenfahrzeug.letzteGeschwWarNull then
46:         return true
47:     end if
48:
49:     return false
50: end function
```

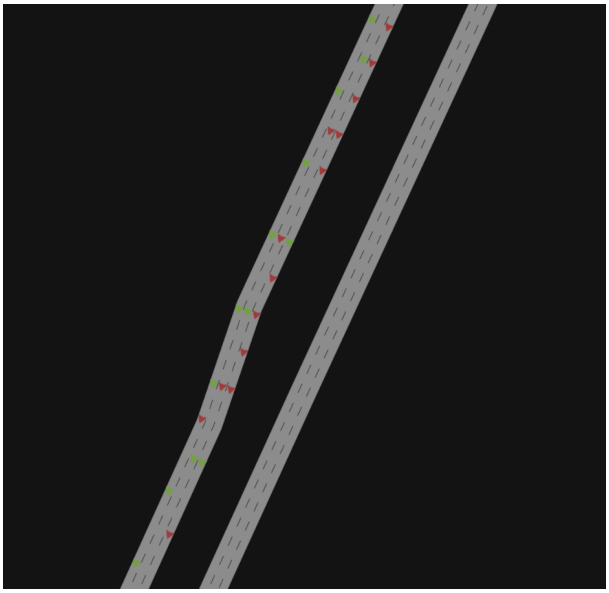


Abbildung 2.6: Stauinversion im Simulator; rote Fahrzeuge, vermehrt auf der inneren Spur zu finden, stehen, während grüne Fahrzeuge mit $30\,\mathrm{km/h}$ außen überholen

Überholen werden Abstand und Geschwindigkeiten der relevanten Fahrzeuge berücksichtigt. Wenn kein Fahrzeug überholt werden soll (z. B. wenn der gesamte Stau überholt wurde), dann wird sich wie bei TENDIERTZURÄUSSERSTENSPUR() verhalten.

Die Abfrage in MÖCHTEÜBERHOLEN(), ob das Fahrzeug im letzten Zeitschritt gestanden hat, ist für das Fahrverhalten sehr einflussreich. Mit dieser Abfrage tendieren die Fahrzeuge bei Stau zur linken Spur, weil sie versuchen, stehende Fahrzeuge zu überholen. Das macht nicht nur den Verkehr in Stausituationen lebendiger, indem Fahrzeuge häufiger die Spur wechseln. In Realität lässt sich außerdem noch das Phänomen der *Stauinversion* beobachten, was mit dieser Abfrage abgedeckt wird und in Abbildung 2.6 zu sehen ist. Dieses beschreibt, wie der Verkehrsfluss bei hohem Verkehrsaufkommen auf der inneren Spur höher konzentriert ist als auf der äußeren Spur, obwohl hier die schnelleren Fahrzeuge fahren sollten.

Wichtig ist hier noch zu erwähnen, dass bei den Prozeduren WECHSELTZURINNERENSPUR() und WECHSELTZURÄUSSERENSPUR() geprüft wird, ob auf der entsprechenden Nachbarspur Platz ist. Ein sehr wichtiger Aspekt dabei ist die Situation, wenn auf z. B. einer dreispurigen Kante zwei Fahrzeuge auf den äußeren beiden Spuren auf die gemeinsame mittlere Spur wechseln möchten und kollidieren. Um das zu beheben, wurde in der Prozedur WECHSELTZURINNERENSPUR() zusätzlich zur inneren Nachbarspur noch die übernächste Nachbarspur auf ausreichend Platz geprüft. So wird sichergestellt, dass ein Fahrzeug, das nach dem „Rechtsfahrgebot“ handelt, höhere Priorität beim Spurwechsel als das überholende Fahrzeug hat.

2.6.2 Erweiterung der Kreuzungslogik

Auch wenn sich die größten Änderungen im Nagel-Schreckenberg-Modell abspielen, so muss die Kreuzungslogik doch um wenige Punkte erweitert werden. Zum einen wird die Vorfahrt pro Kante, nicht pro Spur berechnet. Dadurch kann nur ein Fahrzeug pro Kante abbiegen. In der Realität kommt es aber sehr häufig vor, dass mehrere Spuren einer Straße gleichzeitig fahren dürfen. Indem die Indizes pro Spur und nicht pro Kante bestimmt werden, lässt sich dieses Problem einfach lösen.

Ein weiterer Aspekt, der überarbeitet werden muss, ist die Registrierung im Knoten. Bisher sollte sich jedes Fahrzeug registrieren, das am Ende einer Kante steht. Jetzt muss im Nagel-Schreckenberg-Modell vorher noch geprüft werden, ob das Fahrzeug überhaupt auf der richtigen Spur ist. Diese Aufgabe dem Knoten zu überlassen ist im mikroskopischen Sinne nicht sinnvoll, da der Knoten sonst eine zu aktive Rolle übernimmt. Schließlich ist es Aufgabe des Fahrers, die korrekte Spur zu wählen.

2.6.3 Beseitigung von Deadlocks

Mit der bisherigen Implementierung des Nagel-Schreckenberg-Modells können am Ende einer Kante zwei Fahrzeuge den Wunsch haben, auf die Spur des jeweils anderen Fahrzeugs zu wechseln. Das führt zu häufigen und sinnfreien Staus. Um das zu beheben, wurden der Straßengraph und die Bremsfunktion im Nagel-Schreckenberg-Modell erweitert.

- Der Straßengraph beinhaltet nur noch Kanten mit mindestens zwei Zellen.
- Die Bremsfunktion bremst Fahrzeuge, die auf einer falschen Spur stehen, nicht direkt vor einem Knoten ab, sondern mit einer freien Zelle dazwischen. Der so frei gewordene Bereich direkt vor einem Knoten wird als *kritischer Abschnitt* behandelt. Es dürfen nur Fahrzeuge in diesen Abschnitt, die auf der für ihre Route korrekten Spur fahren (oder deren Route nach der Kante zu Ende ist). Das sorgt dafür, dass richtig eingeordnete Fahrzeuge die Kante verlassen können und der Verkehrsfluss gefördert wird. Fahrzeuge, die nicht korrekt eingeordnet sind, dürfen nur in die kritische Zone, wenn diese leer ist. Außerdem muss solch ein Fahrzeug das äußerste Fahrzeug in der Kante sein. Ist das nicht der Fall, dann können mehrere falsch eingeordnete Fahrzeuge gleichzeitig in den leeren kritischen Bereich und die Blockade kann wieder auftreten.

Was immernoch bleibt, sind nach wie vor kreuzungsübergreifende Deadlocks. Diese können vermutlich durch dynamisches Routing in den Griff bekommen werden, d. h. die Fahrzeuge können ihre Route während der Fahrt anpassen.

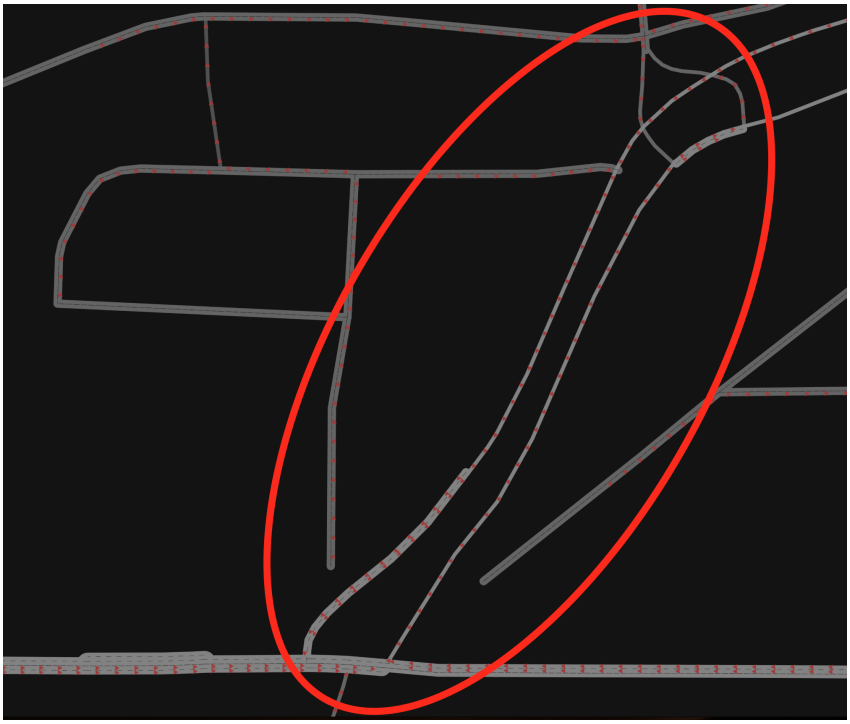


Abbildung 2.7: Es ist ein Deadlock zu erkennen, der dadurch zustande kommt, dass Fahrzeuge der Kante K_1 auf Kante K_2 abbiegen möchten, die voll ist. K_2 kann aber nicht leer werden, weil die Fahrzeuge hier entweder direkt auf die Fahrzeuge von K_1 oder auf Fahrzeuge anderer Kanten warten müssen, die wiederum direkt oder indirekt auf K_1 warten.

3 Microtrafficsim - Der Verkehrssimulator

Das letzte Kapitel hat das verwendete Modell und die Logik darin erläutert. Dabei wurden Implementierungsdetails bewusst außen vor gelassen. In diesem Kapitel soll der Fokus auf genau jenen Details liegen. Der verwendete Simulator ist unter https://github.com/sgs-us/microtrafficsim zu finden. Aufgrund der Projektgröße kann nicht auf jeden Aspekt eingegangen werden. Deshalb werden die für die Logik am wichtigsten und relevantesten Punkte hervorgehoben.

Zu Beginn dieser Bachelorarbeit war der Simulator schon vorhanden und wurde im Rahmen der Arbeit verbessert und erweitert. Es wird in jedem Fall, wenn im Rahmen der Arbeit etwas verbessert oder erweitert wurde, explizit darauf hingewiesen. Ist das nicht der Fall, so handelt es sich um eine Beschreibung des vorhandenen Programmcodes.

3.1 Echtzeitsimulation durch hohe Performanz

Neben guter Strukturierung und möglichst hoher Modularität wurde bei der Implementierung auch Wert auf die Performanz gelegt. So ist die Simulation auf einem guten Laptop (8 bis $16\,\mathrm{GB}$ RAM, CPU mit 4 Kernen und um die $2\,\mathrm{GHz}$) in Echtzeit möglich (je nach gewählten Simulationsparametern bis zu 100-fach schneller). Flaschenhals hierbei war immer die Vorverarbeitung, die Routenberechnung eines Szenarios oder die Ausführung des Nagel-Schreckenberg-Modells ab einer zu hohen Fahrzeugzahl.

Zur Vorverarbeitung gehört das Parsen von OSM-xml-Files, was nicht Teil dieser Arbeit ist. Bei sehr großen Straßennetzen, z. B. in der Größe des Regierungsbezirks Stuttgart (Fläche von ca. $\frac{1}{3}$ des Bundeslandes Baden-Württemberg), braucht das Parsing wenige Minuten auf einem oben beschriebenen Laptop.

3.1.1 Performanz bei der Routenberechnung

Die Zeit für die Routenberechnung eines Szenarios ist, gegeben durch die verwendeten Algorithmen zur Wegsuche, stark von der Anzahl Knoten ($\Leftrightarrow$ Kartengröße) und der zu berechnenden Anzahl Routen abhängig. Bei wenigen $10\,000$ Fahrzeugen auf einer der Anzahl entsprechen

großen Karte dauert die Berechnung auf einem oben beschriebenen Laptop ebenfalls wenige Minuten.

3.1.2 Überblick über das implementierte Multithreading und Optimierungspotential für die Laufzeit

Ein sehr wichtiger Mechanismus, um die Simulation in Echtzeit zu ermöglichen, ist das Multithreading. Daraus resultiert, dass bei der Implementierung einige Details beachtet werden müssen. Manche Datenstrukturen müssen entweder kopiert oder nebenläufig verarbeitet werden können. Aus der Überlegung heraus, dass die Routen pro Fahrzeug definiert sind und auf die Threads verteilt werden, wurde der Ansatz gewählt, auch die Fahrzeuge selbst bei der Berechnung des Nagel-Schreckenberg-Modells auf die Threads zu verteilen. Ab einer Fahrzeugzahl von $100\,000$ kann die Simulation das Modell nicht mehr viel schneller als Echtzeit ($\Leftrightarrow$ ein Simulationsschritt pro Sekunde) beschleunigen. Die verwendeten Datenstrukturen der Kanten, die im Rahmen dieser Arbeit für die Mehrspurigkeit komplett neu entworfen werden mussten, werden für jeden Zugriff (mit logarithmischer Laufzeit) blockiert. Würden nicht Fahrzeuge, sondern Kanten auf die Threads verteilt werden, so würde enorm viel Synchronisierungsarbeit gespart werden, denn im Nagel-Schreckenberg-Modell werden alle Geschwindigkeiten so berechnet, dass die Fahrzeuge keine Kollision haben. Nur noch an Knotenübergängen müssten die Kantendatenstrukturen blockiert werden.

Durch das Multithreading wird die Ausführung der Simulation abhängig von der nichtdeterministischen Ausführungsreihenfolge der Threads. Daraus resultierendes Verhalten ist nichtdeterministisch, was für die Analyse im Kapitel 4 – Verkehr mit Unsicherheitsquantifizierung ein Problem ist. Aus diesem Grund ist der Determinismus für die Simulation nicht nur wichtig, sondern musste im Rahmen dieser Arbeit immer wieder erneut garantiert werden.

Nicht nur das Multithreading hat die Zeit verringert, bis die Simulation rechnet. Eine Serialisierung wurde außerhalb der Bachelorarbeit implementiert, die dafür sorgt, dass der Straßengraph oder Routen binär abgespeichert werden können. Innerhalb der Bachelorarbeit wurde diese Serialisierung verwendet, um Routen deterministisch abzuspeichern und neu zu laden. Die Komplexität bestand dabei darin, die Routen mit den selben Fahrzeugen zu verbinden, mit denen sie zuvor schon verknüpft waren. Das konnte durch die Überarbeitung der Kanten-IDs und deren Zuordnung zu den Fahrzeugen erreicht werden. Hierfür wurden Metarouten eingeführt, die speichersparend in einem möglichen Simulationsszenario abgespeichert werden, über die später eindeutig iteriert werden kann.

3.2 Implementierung der Verkehrslogik

3.2.1 Implementierung des Nagel-Schreckenberg-Modells

Ohne Kreuzungen ist die Implementierung des Nagel-Schreckenberg-Modells relativ unkompliziert. Die Einführung von Kreuzungen macht sie etwas komplexer. Das beginnt damit, dass die Knoten einen konsistenten Zustand der Fahrzeuge benötigen, um die Vorfahrt effizient bestimmen zu können. Aus diesem Grund werden am Ende des Nagel-Schreckenberg-Modells zwei Prozeduren eingeführt:

5. **Nach dem Bewegen:** Jedes Fahrzeug aktualisiert für die Berechnung relevante Parameter (z. B. ein boolean, ob im letzten Schritt $v = 0$ war) und registriert oder deregistriert im Knoten. Das Fahrzeug registriert sich, wenn es mit der aktuell maximal erlaubten Geschwindigkeit das Kantenende erreichen würde. Außerdem darf das Fahrzeug nicht am Ende der Route sein, denn die Registrierung ist nur relevant, wenn das Fahrzeug auch wieder aus dem Knoten austreten möchte. Desweiteren ist wichtig, dass sich nur das vorderste Fahrzeug auf der Kante im Knoten registrieren lässt. So muss nicht der Knoten entscheiden, welches Fahrzeug das vorderste ist, was dem mikroskopischen Ansatz entspricht. Ist eine dieser Eigenschaften nicht gegeben, dann muss sich das Fahrzeug deregistrieren. Dieser Schritt ist wichtig aufgrund der Spurwechsel: es ist möglich, dass das Fahrzeug seit dem Spurwechsel eines anderen Fahrzeugs nicht mehr das vorderste der Kante ist.

6. **Knoten aktualisieren:** In [MTS16] war es so, dass die Fahrzeuge direkt beim Eintreten in den Knoten mit allen bereits registrierten Fahrzeugen verglichen wurde. Das ließ sich mit einer einfachen Synchronisation gut implementieren. Die Aktualisierung der Knoten musste aber im Rahmen dieser Arbeit verändert werden. Bei einem Spurwechsel kann es passieren, dass sich das vorderste Fahrzeug auf einer Spur verändert, obwohl das vorher vorderste Fahrzeug schon am Knoten registriert ist. Das zieht nicht nur eine Deregistrierung nach sich, bei der erneut ein Fahrzeug mit jedem registrierten Fahrzeug verglichen wird, um den $prioritätszähler$ rückgängig zu verändern. Das neue vorderste Fahrzeug muss sich auch noch am Knoten registrieren.

   Um den Rechenaufwand bei der De-/Registrierung zu reduzieren, der durch Spurwechsel öfters aufgebracht werden muss, wurde eine Datenstruktur im Knoten eingeführt, die jedes Fahrzeug speichert, das sich in dem aktuellen Simulationsschritt registrieren möchten. Erst, wenn die Knoten aktualisiert werden, werden auch alle neuen Fahrzeuge mit bereits registrierten verglichen. Die verwendete Datenstruktur ist außerdem eine Prioritätswarteschlange, die die Fahrzeuge nach ihrer ID sortiert. Das ist für eine deterministische Ausführung der Simulation wichtig, weil die Fahrzeuge von verschiedenen Threads verarbeitet werden und so die Registrierungsreihenfolge von der Ausführungsreihenfolge der Threads abhängig ist.

Ein weiterer Punkt bei der Registrierung im Knoten ist zu Beginn einer Simulation, wenn die Fahrzeuge noch nicht auf dem Straßengraphen fahren. Hierfür wurde die Eigenschaft abgespeichert, ob ein Fahrzeug bereits *gespawnt* ist. In Videospielen (wie auch hier in der Simulation) werden Objekte oder Figuren nicht „geboren“ , sondern müssen irgendwo/irgendwann in die digitale Umgebung eingebunden werden (meist vor dem Betrachter versteckt). Der Begriff *spawnen* beschreibt eben dieses „plötzliche“ Auftauchen von Objekten oder Figuren in einer digitalen Umgebung. Nicht gespawnte Fahrzeuge haben im Knoten immer Vorfahrt zu gewähren. Bei zwei noch nicht gespawnten Fahrzeugen wird die Vorfahrt für eine deterministische Ausführung mit Hilfe ihrer ID entschieden.

Die Bewegungsprozedur wurde wie oben erläutert in zwei Prozeduren aufgespalten. Die eine Prozedur verändert die Position der Fahrzeuge, hat also zur Ausführungszeit inkonsistente Datenstrukturen zu verantworten. Die zweite Prozedur verändert den Zustand der Fahrzeuge, nicht aber deren Position, und setzt einen konsistenten, stabilen Zustand der Datenstrukturen voraus. Aus der selben Begründung heraus wurde die Prozedur WECHSELTSPUR(), die Tei dieser Bachelorarbeit ist, in zwei Prozeduren gespalten. Die eine Prozedur, WIRDSPURWECHSELN(), berechnet (ohne Fahrzeugbewegung), ob ein Fahrzeug die Spur wechseln möchte, und merkt sich diese Entscheidung. Die zweite Prozedur, WECHSELTSPUR(), führt dann nur noch den Wechsel der Spur ohne weitere Prüfung auf den Datenstrukturen der Kanten aus. Hierbei ist wichtig, das Fahrzeug bei einem Spurwechsel aus dem Knoten zu deregistrieren, denn die Bewertung der Vorfahrt ist maßgeblich von der Spur des Fahrzeugs abhängig.

3.2.2 Konnektoren zur Verknüpfung von Spuren am Knoten

Die Veränderungen bei der Registrierung eines Fahrzeugs im Knoten wurden bereits erläutert. Es wurde noch nicht darauf eingegangen, wie bestimmte Abbiegeregeln implementiert sind. Nicht jede eingehende Kante an einem Knoten erlaubt das Abbiegen auf jede ausgehende Kante. Es gibt beliebige Einschränkungen, z. B. wird an manchen Kreuzungen abhängig von der eingehenden Kante nur in bestimmte Richtungen eingebogen, obwohl mehrere Kanten aus dem Knoten herausführen. Um das zu implementieren, wurden Konnektoren verwendet. Sie repräsentieren ein Tupel aus eingehender und ausgehender Spur. Mit Einführung der Mehrspurigkeit musste die Speicherung der Konnektoren auf Spuren angepasst werden. Intern sind Abbiegebedingungen daher als Tripel abgespeichert: eingehende Spur - ausgehende Kante - ausgehende Spur. Hierfür werden jetzt Baumstrukturen verwendet, die Mappings von einer eingehenden Spur über eine ausgehende Kante auf eine ausgehende Spur beinhalten. Diese Konnektoren sind relevant für die Berechnung, ob ein Fahrzeug auf einer Kante auf der korrekten Spur eingeordnet ist.

Abbildung 3.1: Man sieht, dass die Straße und damit die beiden assoziierten Kanten oben links und unten links vom markierten Knoten diesen sowohl verlässt als auch in den Knoten eingeht.

3.2.3 Verbesserungen beim Abspeichern von Kanten und anderen Logikelementen

Viele Elemente der Logik, also Kanten, Knoten, Fahrzeuge, usw. wurden über einen eindeutigen Hashcode abgespeichert, um deterministische Iterationsreihenfolgen zu ermöglichen. Die Identifikation über Hashcodes wurde im Rahmen der Arbeit durch Schlüssel ersetzt, die für ein Logikelement eindeutig ist. So können Hashcodes in davon abhängigen Datenstrukturen zur Laufzeitoptimierung frei verwendet werden, ohne dass der Determinismus in der Ausführung verloren geht. Außerdem sind die Schlüssel leicht erweiterbar. Ein Fall, wo das relevant war, ist in Abbildung 3.1 dargestellt. Dieses Beispiel ist eines von vielen, das zeigt, wieso das Straßennetz komplex ist. Es wurde behandelt, indem die Richtung jeder gerichteten Kante in ihren eindeutigen Schlüssel aufgenommen wurde.

3.3 Validierung und Testen der Implementierung

Ein großer und häufig in der Praxis unterschätzter Aspekt der Software-Entwicklung ist das Schreiben von Testfällen. Dabei sind sie für die Zuverlässigkeit und Korrektheit einer

Software sehr relevant, so auch hier beim Verkehrssimulators. Aus diesem Grund wurden verschiedene Kreuzungsszenarien so implementiert, dass ihre Korrektheit automatisch geprüft wird. Im Rahmen dieser Arbeit wurden einige Tests implementiert, die den Determinismus prüfen. Dazu gehört das automatische Simulieren von Simulationsschritten, bei denen in bestimmten Abständen der Zustand eines jeden Fahrzeuges gespeichert und in weiteren Simulationsdurchläufen als Referenzwert mit aktuell gespeicherten Zuständen verglichen wird. Außerdem gibt es Tests, die das deterministische Speichern und Laden von Routen testen. Bestehende Tests zu verwendeten mathematischen Vektorfunktionen wurden ebenfalls ergänzt.

4 Verkehr mit Unsicherheitsquantifizierung

Nachdem die Simulation modelliert und implementiert wurde, soll sie nun mit realen Daten verglichen werden. Der Verkehr ist aber ein recht komplexes Phänomen, obwohl er auf überschaubar vielen Regeln basiert. Um das zu verdeutlichen, reicht das Beispiel der Hauptverkehrszeit. Quasi jeder, der mit dem Auto morgens zur Arbeit fährt, wird wissen, dass viel los ist. Schließlich sind da viele Menschen unterwegs. Dennoch verhält es sich jeden Tag anders. Das bedeutet, trotz der Regelmäßigkeit im Verkehr muss mit Abweichungen gerechnet werden. Z. B. sind mal etwas mehr oder weniger Fahrzeuge unterwegs. Ebenso kann das Fahrverhalten der Fahrer variieren, z. B. je nach Wetter.

Für solch Abweichungen vom Regelfall eignen sich Methoden zur Unsicherheitsquantifizierung sehr gut, wie es in Abbildung 4.1 schematisch dargestellt ist. Bei solchen Methoden geht es darum, den Wert der Eingabeparameter (Verkehrsdichte, Fahrverhalten) mit einer Zufallsvariable zu beschreiben, deren Stochastik durch eine Wahrscheinlichkeitsverteilung eindeutig definiert ist. Nach diesen Wahrscheinlichkeitsverteilungen werden Parameterkombinationen gebildet, die in diesem Fall von dem zu untersuchenden Verkehrsmodell verarbeitet werden sollen. Die Ausgabedaten können z. B. Fahrzeiten oder Geschwindigkeiten sein. Dadurch dass die Eingabe eine Zufallsvariable ist, ist auch die Ausgabe eine Zufallsvariable und kann mit Mitteln der Stochastik untersucht werden. Mit der Unsicherheitsquantifizierung wird so versucht, den Einfluss der Stochastik der Eingabe auf die Ausgabe zu quantifizieren: Welcher Parameter hat den größten Einfluss und wie viel? Wie groß ist die erwartete Zielgröße bzw. wie groß ist die mittlere Abweichung davon? Mit welcher Wahrscheinlichkeit tritt ein bestimmtes Ereignis ein, z. B. mit welcher Wahrscheinlichkeit dauert eine Fahrt von Tübingen nach Stuttgart länger als 2h?

4.1 Vorbereitung des Szenarios

In dieser Arbeit liegt der Fokus auf qualitativen Aussagen. Das Modell soll also ein Szenario ablaufen lassen, dessen Ausgabe mit gemessenen, realen Daten verglichen wird, um dann daraus gezogene Schlüsse zu untermauern.

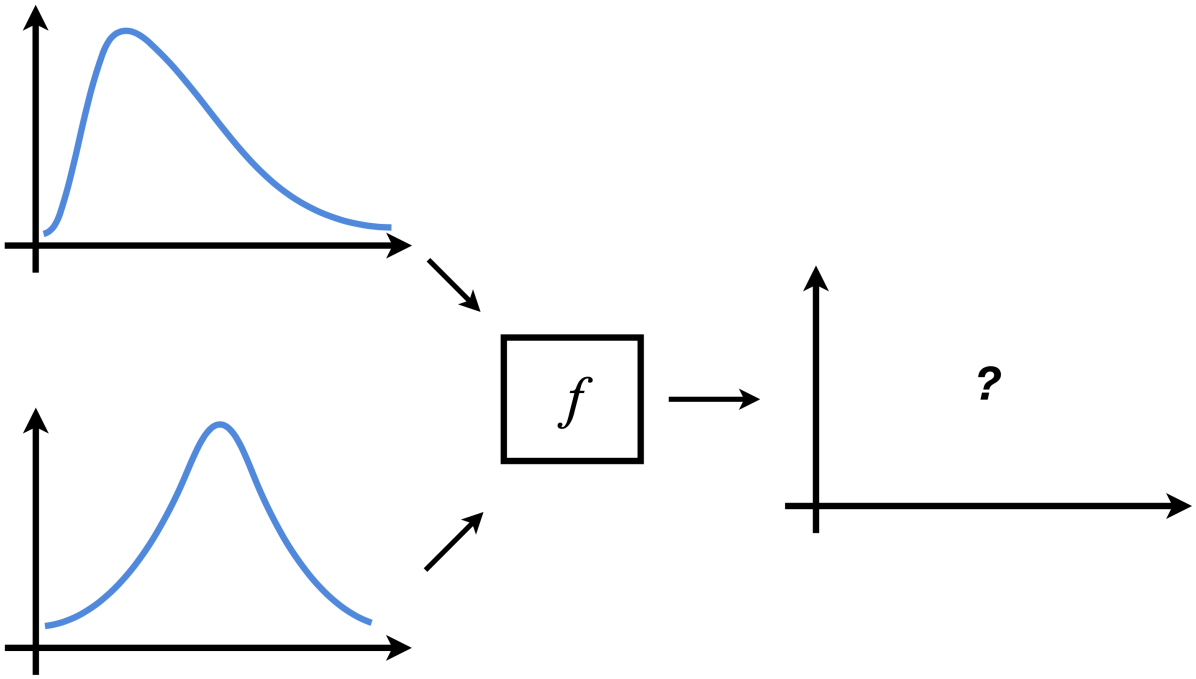


Abbildung 4.1: Links sind die Wahrscheinlichkeitsverteilungen der Eingabeparameter dargestellt. Mittig findet sich die Funktion, die diese Parameter verarbeitet und die rechts dargestellten Ausgabedaten berechnen soll.

4.1.1 Szenario - Von Tübingen zur Universität Stuttgart-Vaihingen

Das gewählte Szenario ist eine einzelne Fahrt von Tübingen zur Universität Stuttgart-Vaihingen. Alle anderen Verkehrsteilnehmer beginnen und enden zufällig auf dem Straßennetz. Der Maßstab bietet sich deshalb an, weil er sehr vielseitig ist. Es gibt viele Kleinstädte und ein paar größere Städte wie Stuttgart, welche alle über Schnellstraßen miteinander verbunden sind. Ein größerer Maßstab würde Aussagen über den Verkehr zu makroskopisch betrachten, während ein kleinerer Maßstab die Situation zwischen Schnellstraßen und Städten nicht so genau erfassen kann.

Für den Simulator wurden zwei gleichverteilte Eingabeparameter definiert:

- Die Anzahl der Fahrzeuge, die zum Zeitschritt 0 losfahren, werden gesetzt. Sie liegt im Bereich $[100, 50\,000]$ und hat standardmäßig den Wert $20\,000$. Der Wertebereich erschien deshalb sinnvoll, weil Quellen wie [Blech1616] an großen Knotenpunkten des Stuttgarter Straßennetzes ganz grob um die $100\,000$ Fahrzeuge am Tag zählen. Werten die Hauptverkehrszeiten morgens und abends mit jeweils 35-40 % gewichtet und ein wenig Spielraum addiert, so resultiert das hier gewählte Maximum von $50\,000$ Fahrzeugen zur Hauptverkehrszeit.

Abbildung 4.2: Der Kartenausschnitt, der simuliert wurde, erstreckt sich vom Breitengrad 48,348 900 0 bis 48,865 600 0 und vom Längengrad 8,736 900 0 bis 9,433 800 0. Die gemessene Route von Tübingen zur Universität Stuttgart-Vaihingen ist eingezeichnet.

- Die Wahrscheinlichkeit $p_{\text{Spurwechsel}}$ wird gesetzt. Sie wurde über das Intervall $[0, 1]$ gleichverteilt und hat den Standardwert $0{,}8$.

Die Wahrscheinlichkeit zum Trödeln $p_{\text{Trödeln}}$ wurde auf $0{,}2$ gesetzt [BZBP09, Seite 194]. In Anbetracht dessen, dass nur die $20\,\%$ Trödler nicht zum Überholen neigen, erschien $0{,}8$ als Überholfaktor sinnvoll. Die Routen wurden gleichverteilt zufällig aus einer vorher berechneten Routenmenge gewählt, die zu $70\,\%$ schnellste und zu $30\,\%$ kürzeste Routen beinhaltet.

Die realen Daten wurden regelmäßig zur Hauptverkehrszeit zwischen sieben und zehn Uhr morgens gemessen.

4.1.2 Monte Carlo für die Parameterkombinationen und Fehlerabschätzung

Zur Bestimmung geeigneter Parameterkombinationen und zur Schätzung des Erwartungswertes und der Varianz werden Monte Carlo Methoden verwendet. Sie eignen sich deshalb hervorragend für dieses Problem, weil sie keine Glattheitsanforderungen an die Funktion stellen. Die Zielfunktion muss nur auszuwerten sein. Das hat nicht nur den Vorteil, dass an die Funktion keine Bedingungen gestellt werden, sondern dass auch funktionsunabhängig implementiert werden kann.

Auch schön bei Monte Carlo Methoden ist die Tatsache, dass sich ihre Korrektheit auf eine recht grundlegende Theorie der Stochastik zurückführen lässt. Im Folgenden wird ein kurzer Aufriss dieser Herleitung gegeben [MCQuad15, Seite 5].

Sei $u : \mathbb{R}^d \to \mathbb{R}$ eine d-dimensionale Funktion, die auf einem Intervall $I = I_1 \times \cdots \times I_d$ mit $I_1, \ldots, I_d \subseteq \mathbb{R}$ definiert ist. Das Volumen von I sei V_I. Dann sei z_D die Dichtefunktion einer stetigen, gleichverteilten Zufallsvariablen Z über I.

$$z_D(x) := \begin{cases} \frac{1}{V_I} & x \in I \\ 0 & sonst \end{cases} \tag{4.1}$$

Wird z_D mit der Gewichtsfunktion

$$g(x) := V_I \cdot u(x) \tag{4.2}$$

gewichtet, so ergibt sich für den Erwartungswert und die Varianz von Z:

$$\begin{aligned} \mu_Z &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} g(x) z_D(x) \mathrm{d}x = \int_I V_I \cdot u(x) \cdot \frac{1}{V_I} \mathrm{d}x = \int_I u(x) \mathrm{d}x \\ \sigma_Z^2 &= \mathbb{E}\left((Z - \mu_Z)^2\right) = \int_I \left(g(x) - \mu_Z\right)^2 z_D(x) = \int_I \left(V_I \cdot u(x) - \mu_Z\right)^2 z_D(x) \end{aligned} \tag{4.3}$$

Nach dem *Zentralen Grenzwertsatz* lässt sich der Erwartungswert von Z ebenfalls berechnen, indem das arithmetische Mittel M_k von k-vielen Z-verteilten Gewichten berechnet wird, wobei k für eine hohe Genauigkeit hinreichend groß sein muss.

$$\mu_Z = \mathbb{E}\left(\lim_{k\to\infty} M_k\right) \tag{4.4}$$

Bei den folgenden Umformungen wird die Linearität des Erwartungswertes $\mathbb{E}$ und der Summe $\sum$ ausgenutzt. Der letzte Umformungsschritt darf gemacht werden, weil $u(x_i)$ und damit auch die Summe über alle i einen festen Wert darstellt.

$$\mu_Z = \mathbb{E}\left(\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} g(x_i)\right) = V_I \cdot \mathbb{E}\left(\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} u(x_i)\right) = V_I \cdot \lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} u(x_i) \tag{4.5}$$

Die Konvergenz ist demnach nicht nur garantiert und nur vom Definitionsbereich der Zielfunktion abhängig, sondern insbesondere auch unabhängig von deren Dimension und Glattheit. Dass später der Erwartungswert ausschließlich von den Eingabeparametern abhängt und nicht noch stochastisch abhängig vom Modell ist, muss das Modell deterministisch sein. Damit einhergehend muss auch der Simulator, der das hier vorgestellte Verkehrsmodell implementiert, detereministisch sein. Das ist hier wie im Kapitel 3 garantiert worden.

4.1.3 Auswahl der Parameterwerte und Abschätzung

Ein noch fehlender Punkt ist die Auswahl der Parameterwerte. Um die Konvergenz der Monte Carlo Methoden zu verbessern, kann die Fehlerabschätzung mit der Tschebyscheff-Ungleichung zu Hilfe genommen werden [MCQuad15, Seite 6]. Angewendet auf die M_k mit Erwartungswert μ_Z und Varianz $\sigma^2_{M_k}$ lautet sie:

$$\forall \epsilon > 0\colon P\left(|M_k - \mu_Z| \geq \epsilon\right) \leq \frac{\sigma^2_{M_k}}{\epsilon^2} \tag{4.6}$$

Es lässt sich hier sehr schön erkennen, dass für die selbe Fehlerwahrscheinlichkeit ein kleinerer Fehler ϵ mit einer kleineren Varianz $\sigma^2_{M_k}$ kompensiert werden muss. Da es sich bei M_k um eine gesampelte Zufallsvariable handelt, kann die Varianz über die korrigierte Stichprobenvarianz berechnet werden:

$$\sigma^2_{M_k} = \frac{1}{k-1}\sum_{i=1}^{k}\left(g(x_i) - M_k\right)^2 \approx \frac{1}{k-1}\sum_{i=1}^{k}\left(g(x_i) - \mu_Z\right)^2 \tag{4.7}$$

Hier lässt sich sehen, dass die Varianz wesentlich von der Anzahl der Auswertungspunkte abhängt. Um die benötigte Anzahl an Samples bei gleicher Genauigkeit zu reduzieren, gibt es verschiedene Verfahren, um die Auswertungspunkte besser auf dem auszuwertenden Intervall zu verteilen. In dieser Arbeit wird für die Parameterwerte das Latin-Hypercube-Verfahren angewendet. Dieses Verfahren versucht, möglichst dimensionsunabhängig zu bleiben. Um das zu erreichen, werden insgesamt k (und nicht wie beim naiven Ansatz k^d) Auswertungspunkte bestimmt, die geschickt auf den Dimensionen angeordnet werden. Mehr dazu findet sich in [MCQuad15].

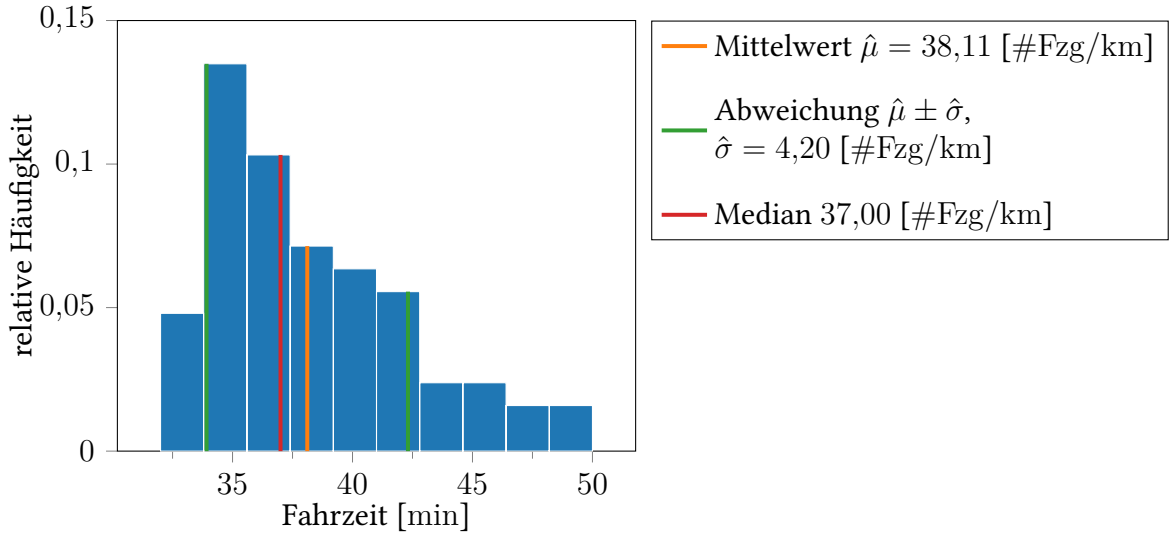


Abbildung 4.3: Dichteverteilung der realen Fahrzeiten

4.2 Simulieren des Szenarios und Auswertung

Nachdem jetzt die Theorie für das simulierte Szenario erläutert wurde, kann simuliert und ausgewertet werden. Zunächst soll es darum gehen, das Modell in Kombination mit dem Szenario auf seine Plausibilität zu prüfen, bevor anschließend Schlüsse aus den Simulationsdaten gezogen werden.

Zur besseren Verständlichkeit wird der Fahrer der realen Daten *Fabian*, der Fahrer der Simulation *Stefan* genannt.

4.2.1 Real gemessene Fahrzeiten

In Abbildung 4.3 sind die real gemessenen Fahrzeiten für die Strecke von Tübingen zur Universität Stuttgart-Vaihingen. Die erwartete Fahrzeit beträgt ca. $38\,\mathrm{min}$ bei einer Strecke von ca. $43\,\mathrm{km}$. Rein rechnerisch ergibt sich damit eine Durchschnittsgeschwindigkeit von $\frac{43\,\mathrm{km}}{38\,\mathrm{min}} \approx 68\,\mathrm{km/h}$. Anhand der Daten lässt sich ebenfalls erkennen, dass Fabian meistens in der *Freiflussphase* zwischen acht und neun Uhr fährt. Außerdem wird die Simulation nach 7200 Schritten gestoppt, was nach dem Nagel-Schreckenberg-Modell einer Fahrzeit von zwei Stunden entspricht.

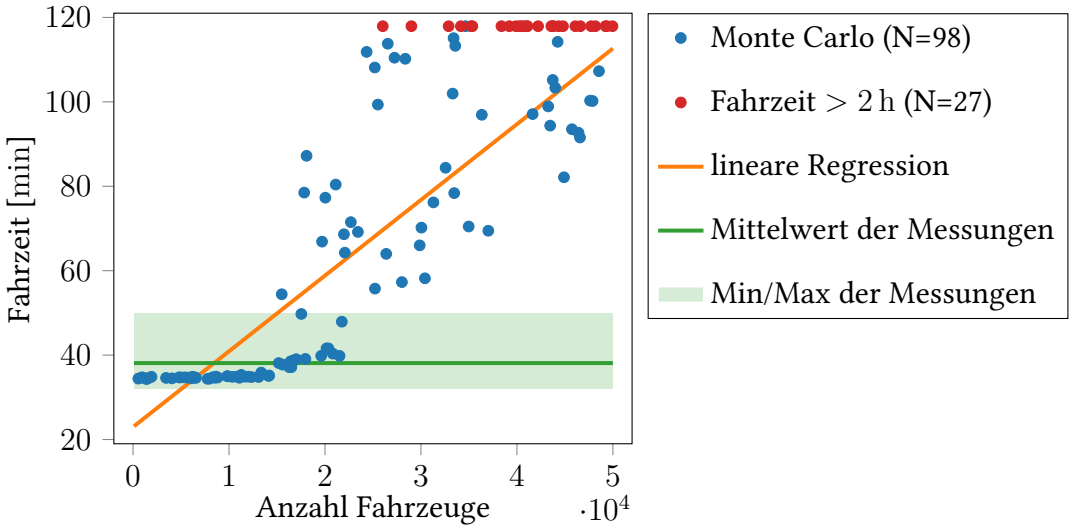


Abbildung 4.4: Die Abbildung zeigt die Fahrzeit von Stefan abgetragen zur Anzahl Fahrzeuge. Der grüne Bereich zeigt die Fahrzeit von Fabian. Es ist zu sehen, dass viele Datenpunkte von Stefan in Fabians grünem Bereich liegen.

4.2.2 Variable Anzahl Fahrzeuge

Für dieses Szenario wird der Überholfaktor konstant auf $0{,}8$ gesetzt. Zur Erinnerung: Der Trödelfaktor liegt konstant bei $0{,}2$ [BZBP09, Seite 194]. Die Anzahl Fahrzeuge variiert gleichverteilt zwischen 100 und $50\,000$.

In Abbildung 4.4 ist die Fahrzeit von Stefan zur Anzahl Fahrzeuge abgetragen. Es ist gut zu sehen, dass Stefan für eine Anzahl an Fahrzeugen bis ungefähr $18\,000$ in der Freiflussphase fährt. Erst ab dann beginnt die Fahrzeit für mehr Fahrzeuge zu schwanken. Trotz der Schwankungen gibt es bis knapp über $21\,000$ Fahrzeugen immernoch eine Tendenz zu der Fahrzeit, die bei freier Fahrt erreicht wird. Erst ab ca. $21\,000$ wird die Fahrzeit instabil. Diese Beobachtung deckt sich mit denen der Publikation [BZBP09, Seite 193], wo der Verkehrsfluss erst ab einer gewissen Verkehrsdichte einbricht.

Auch interessant ist die Beobachtung, dass kein Datenpunkt unter ca. $35\,\mathrm{min}$ liegt. Bei der Strecke von $43\,\mathrm{km}$ entspricht das einer Geschwindigkeit von $\frac{43\,\mathrm{km}}{35\,\mathrm{min}} \approx 74\,\mathrm{km/h}$. In Figur 4.5 entspricht das der maximal erreichten mittleren Geschwindigkeit. Anders herum gibt es keine Durchschnittsgeschwindigkeit unter ca. $22\,\mathrm{km/h}$, denn nach zwei Stunden hat die Simulation abgebrochen. Das ist interessant, weil es zeigt, dass das Nagel-Schreckenberg-Modell, auf dem die Simulation basiert, trotz der hohen Diskretisierung kohärent zur Realität ist.

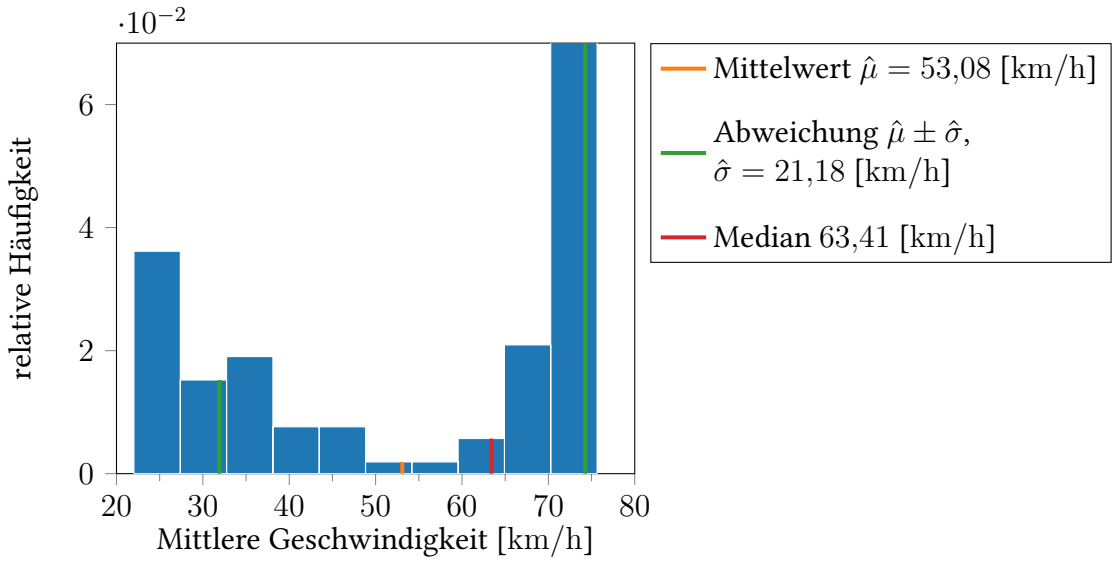


Abbildung 4.5: Die Abbildung zeigt die mittlere Geschwindigkeit von Stefan und deren Häufigkeit.

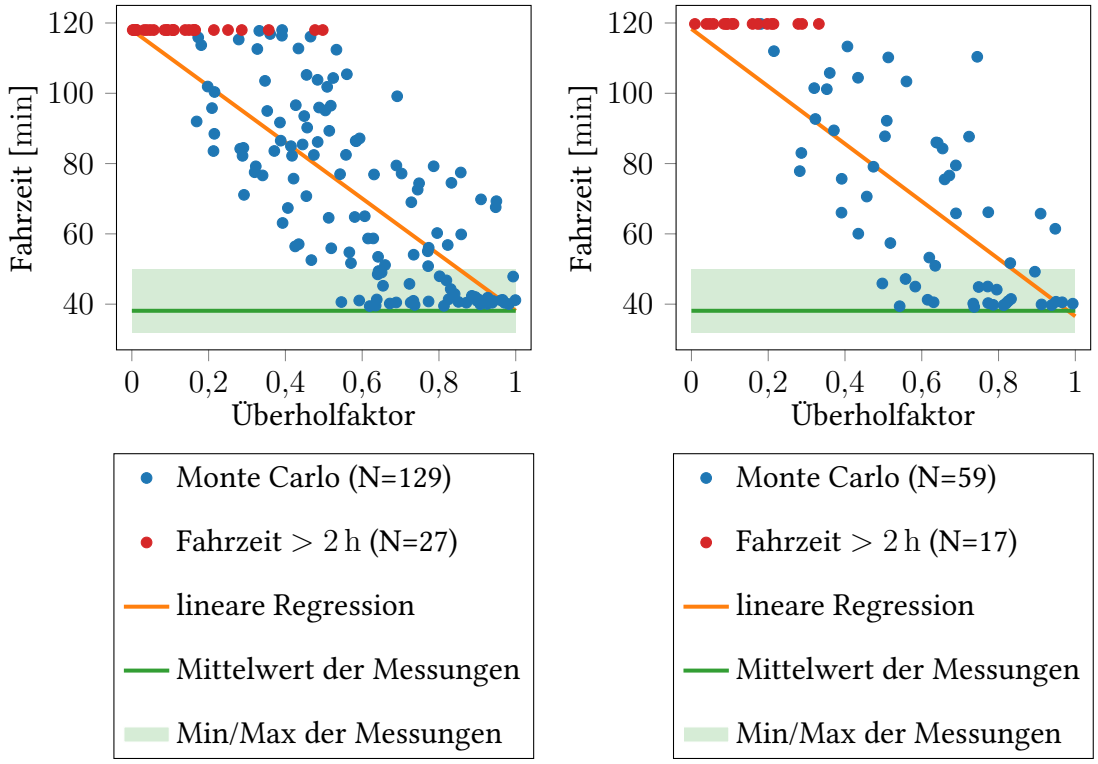
4.2.3 Variabler Überholfaktor

In diesem Szenario variiert der Überholfaktor gleichverteilt zwischen 0 und 1 und die Anzahl Fahrzeuge ist konstant auf $20\,000$ gesetzt (bei Trödelfaktor $0{,}2$).

Abbildung 4.6 zur Fahrzeit bestätigt wieder die Kohärenz zur Realität, denn auch hier gibt es wieder keine Fahrzeit unter ca. $37\,\mathrm{min}$. Es lässt sich ganz deutlich erkennen, wie die Intervallgrenzen der Fahrzeit mit steigendem Überholfaktor proportional abnehmen. Gleichzeitig lässt sich sehen, dass die Fahrzeit stark schwankt, auch bei konstantem Seed. Der Einfluss des Überholfaktors auf die Fahrzeit hält sich daher in Grenzen.

Desweiteren lässt sich sehen, dass bei einem Überholfaktor unter $50\,\%$ die Simulation öfters nach zwei Stunden Simulationszeit abbricht. Das lässt sich dadurch erklären, dass bis zu einem Überholfaktor von $50\,\%$ das Straßennetz tendentiell eher einspurig befahren wird. Bis zu einem Überholfaktor von ungefähr $20\,\%$ ist das Straßennetz total überlastet.

In der Abbildung 4.7 und 4.8 lässt sich wieder die Kohärenz zwischen Verkehrsdichte und Stefans mittlerer Geschwindigkeit erkennen. Mit höherer Überholwahrscheinlichkeit kann Stefan schneller fahren, weil das Straßennetz effizienter ausgenutzt wird, und damit wird die Fahrzeit geringer, wie in Abbildung 4.6 zuvor schon dargestellt. Es gibt ganz klar eine Häufung bei einem Überholfaktor um $0{,}9$ mit vergleichsweise wenigen Ausreißern. Die mittlere Geschwindigkeit beträgt hier ungefähr $66\,\mathrm{km/h}$. Zur Erinnerung: Fabians berechnete Durchschnittsgeschwindigkeit lag bei $68\,\mathrm{km/h}$.



(a) Wechselnder Seed **(b)** Konstanter Seed

Abbildung 4.6: Die Abbildungen zeigen die Fahrzeit abgetragen zum Überholfaktor. Die Tendenz ist bei beiden Plots gleich. Der variable Seed ist über die Eingabeparameter eindeutig definiert. Beim konstanten Seed wurde trotz variablem Überholfakter der selbe Seed verwendet.

4.2.4 Variable Anzahl Fahrzeuge und Überholfaktor

Wie im Kapitel zu Monte Carlo Methoden bereits erläutert wurden Parameterkombinationen aus dem gewünschten Intervall gleichverteilt ausgewählt und simuliert. In Abbildung 4.9 lässt sich diese Verteilung schön beobachten. Es fallen direkt ein paar Eigenschaften auf. Am markantesten ist wohl die lineare Trennung zwischen dem bunten Bereich, in dem Stefan ankam, und dem farblosen Bereich, in dem Stefan seine Fahrt immer nach zwei Stunden abgebrochen hat. Ganz eindeutig ist hier zu erkennen, dass der Überholfaktor (und damit die Tendenz zur Mehrspurigkeit) ein entscheidender Faktor dafür ist, wie viele Fahrzeuge es braucht, bis das Straßennetz um Stefan herum überlastet ist. Der Überholfaktor kann eine höhere Anzahl Fahrzeuge kompensieren. Gleichzeitig zeigt die Abbildung aber auch, wie zuvor in Abbildung 4.6 (Variabler Überholfaktor $\mapsto$ Fahrzeit) erläutert wurde, dass der Einfluss des Überholfaktors trotzdem nur mäßig ist. Das lässt sich daran festmachen, dass die Fahrzeit trotz eines hohen Überholfaktors nicht immer der Freiflussphase entspricht. Die Isolinien der Fahrzeit im Bereich mit Überholfaktor ca. $70\,\%$ und Anzahl Fahrzeuge ca. $20\,000$ zeigen, dass

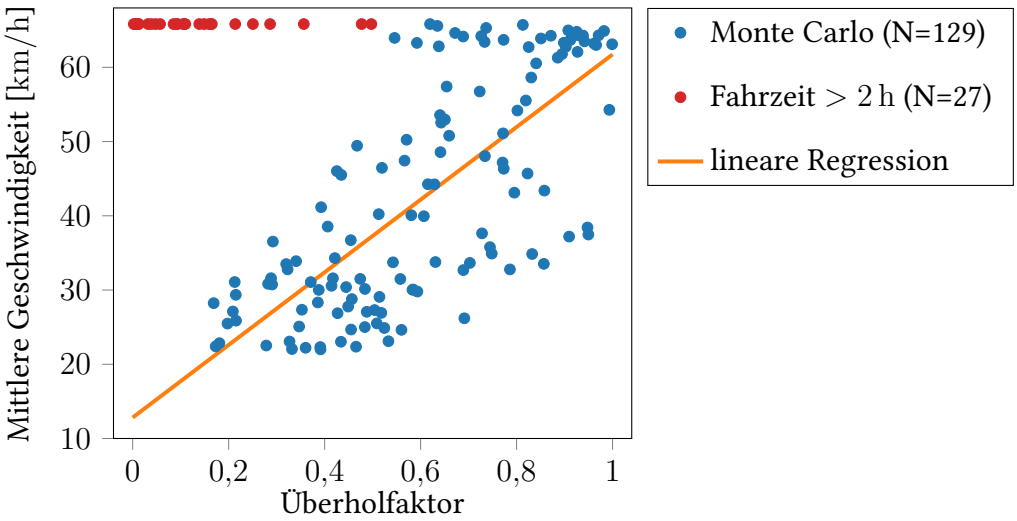


Abbildung 4.7: Die Abbildung zeigt die mittlere Geschwindigkeit abgetragen zum Überholfaktor.

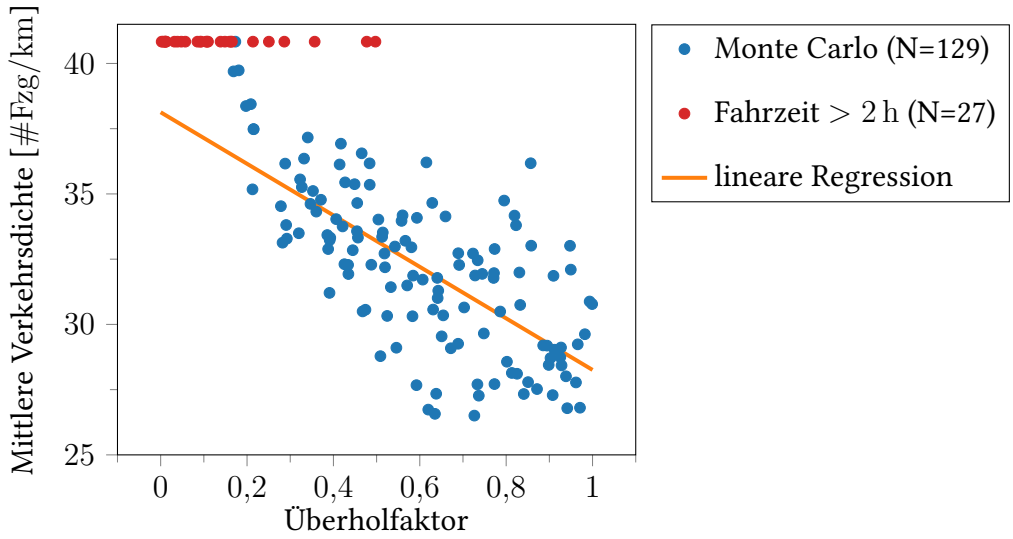
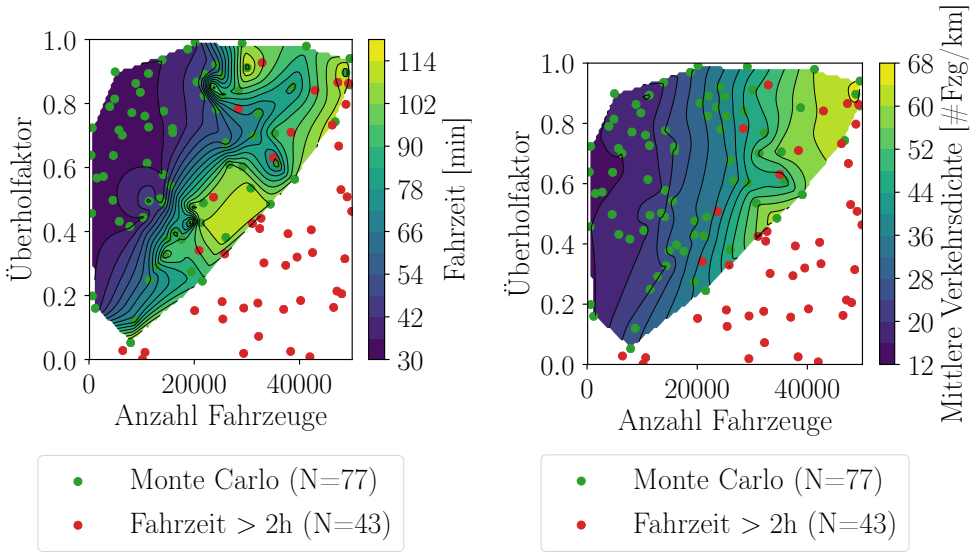


Abbildung 4.8: Die Abbildung zeigt die mittlere Verkehrsdichte abgetragen zum Überholfaktor.



(a) Variable Anzahl Fahrzeuge, Überholfaktor $\mapsto$ Fahrzeit

(b) Variable Anzahl Fahrzeuge, Überholfaktor $\mapsto$ mittlere Verkehrsdichte

Abbildung 4.9: Die Abbildung zeigt die Fahrzeit abgetragen zur Fahrzeuganzahl und zum Überholfaktor. Die gleichmäßige Verteilung der Auswertpunkte durch das Latin-Hypercube-Verfahren über dem zweidimensionalen Intervall (Anzahl Fahrzeuge und Überholfaktor) wird ersichtlich. Die Farbe für die Fahrzeit wurde mit Hilfe der eingezeichneten Auswertungspunkte interpoliert.

sich die Anzahl Fahrzeuge dominanter auf die Fahrzeit auswirkt als es der Überholfaktor tut. Abbildung 4.9b zeigt das noch deutlicher.

5 Zusammenfassung und Ausblick

Im folgenden Abschnitt findet sich eine kurze Zusammenfassung der Arbeit, bevor ein Überblick über die Optimierungsvorschläge und ein Ausblick gegeben wird.

5.1 Zusammenfassung

Der Verkehr ist ein komplexes System. Es wurde mit Hilfe des Braess Paradoxons aufgezeigt, dass eine Simulation für das Design eines Straßennetzes unerlässlich ist, da schon kleine Änderungen am Straßennetz zu großen Veränderungen im Verkehrsfluss führen können.

Bevor sich den mikroskopsichen Modellen zugewandt wurde, ist erläutert worden, dass Straßendaten von [OSM17] bezogen werden können. Diese Daten müssen für eine Simulation aufbereitet werden, was schematisch beschrieben worden ist. Aus der Aufbereitung resultiert ein Straßengraph mit Knoten an genau der Stelle, wo sich Straßen kreuzen, und mit gerichteten Kanten, die eine Fahrtrichtung eines Straßenabschnittes zwischen zwei Knoten repräsentieren. Die Kanten wurden gemäß dem Nagel-Schreckenberg-Modell diskretisiert und nach [MTS16] an Knoten miteinander verknüpft. Die hierfür verwendeten Prozeduren und Funktionen wurden ausführlich in Pseudocode dargestellt. Ein wichtiger Aspekt hierbei war die Registrierung von Fahrzeugen in einem Knoten und die geschickte und realistische Berechnung ihrer Vorfahrt gemäß den Vorfahrtsregeln.

Der wesentliche Teil dieser Bachelorarbeit bestand nun darin, das so entstandene Modell inklusive Simulator auf Mehrspurigkeit zu erweitern, um der Realität näher zu kommen. Hierfür wurde das Nagel-Schreckenberg-Modell in Kombination mit der bisherigen Kreuzungslogik um den Spurwechsel erweitert. Beim Spurwechsel wurde zwischen Kanten- und Knotenlogik unterschieden. Im ersten Fall tendieren Fahrzeuge zum Überholen, was so implementiert wurde, dass das Phänomen der *Stauinversion* sichtbar wurde. Im zweiten Fall tendieren Fahrzeuge zum Rechtsfahrgebot (bzw. Linksfahrgebot) und ordnen sich möglichst rechts (links) in ihrer aktuellen Kante ein.

All die neuen Features wurden zu dem bestehenden Simulator hinzugefügt, was einige Veränderungen im Programmcode nach sich zog. Zur korrekten Analyse musste die Simulation deterministisch implementiert und regelmäßig darauf geprüft werden. Datenstrukturen, wie z. B. die der Kanten, wurden teilweise komplett neu aufgesetzt, um die neuen Anforderungen zu erfüllen. Dabei wurde stets darauf geachtet, die Performanz nicht zu sehr zu reduzieren,

damit die Simulation in Echtzeit auf herkömmlichen Laptops laufen kann. Ein wichtiger Aspekt, um das zu gewährleisten, war die nebenläufige Implementierung des erweiterten Nagel-Schreckenberg-Modells.

Zum Schluss der Arbeit wurde auf die Bewertung des Modells eingegangen. Hierzu wurde die Fahrzeit unterwegs von Tübingen zur Universität Stuttgart-Vaihingen beobachtet und mit den Fahrzeiten aus der Simulation verglichen. Es kamen dafür Methoden der Unsicherheitsquantifizierung und Monte Carlo Methoden zum Einsatz. Ergebnis war, dass das gewählte Modell trotz der hohen Diskretisierung die Realität gut wiederspiegelt und Rückschlüsse auf die Realität gezogen werden können, wie z. B., dass die Tendenz zum Überholen deutlich weniger Einfluss auf die Fahrzeit hat als die Verkehrsdichte.

5.2 Mögliche Optimierungen in Modell und Implementierung

Nicht jedes implementierte Feature war für die Bachelorarbeit direkt relevant. Allerdings gibt es ein paar Veränderungen im Code, die je nach Anwendungsfall hätten relevant sein können. Dieser Abschnitt geht kurz auf solche Veränderungen ein.

5.2.1 Umgesetzte Optimierungen

Wie anfangs des Kapitels über die Implementierung erwähnt sollte der Programmcode möglichst modular gestaltet werden. Aus dieser Motivation heraus wurde das „Fahrzeug“ im Nagel-Schreckenberg-Modell in der Bachelorarbeit verändert. Es gibt jetzt die Unterscheidung zwischen *Fahrer* und *Fahrzeug*. Der Fahrer bündelt alle Eigenschaften, die emotionaler Natur sind, wie z. B. der Trödelfaktor der die gewünschte Maximalgeschwindigkeit. Das Fahrzeug dagegen beinhaltet alle Eigenschaften einer Maschine, also maximal mögliche Beschleunigung oder maximal mögliche Geschwindigkeit. Der Vorteil dieser Unterscheidung liegt auf der Hand: Es ist möglich, einen Fahrer zu modellieren, der wie ein Raser fahren möchte. Er hält sich nicht an die maximal erlaubte Geschwindigkeitsbegrenzung, beschleunigt schneller und hat einen geringeren Trödelfaktor. Sitzt dieser Fahrer in einem Fahrzeug mit geringer Motorleistung, so wird es sich in der Simulation auch langsam verhalten. Anders herum kann ein sehr sportliches Fahrzeug definiert werden, in das ein gemütlicher Fahrer gesetzt wird. So können später viele verschiedene Fahrzeug- und Fahrertypen unabhängig voneinander modelliert werden.

5.2.2 Geplante Optimierungen

Nicht alle gewünschten Optimierungen fanden aus Zeitgründen Einzug in diese Arbeit. Daher werden hier geplante bzw. gewünschte Optimierungen erläutert.

Die Wege, die bei der Kreuzungslogik mit den Kreuzungsindizes gebildet werden, sind in der Form sehr verbos aufgebaut. Für die Logik würde es vollkommen ausreichen, die beteiligten Start- und Zielindizes aufsteigend zu sortieren. Anschließend wird solange das so entstandene Pattern geshiftet, bis im Pattern der Startindex vor seinem jeweiligen Zielindex steht. Der zweite Startindex entspricht dann dem ersten Index des *gemeinsamen Patterns*, welches im Abschnitt 2.4.3 über die Kreuzungsindizes beschrieben wird. Seien wieder $weg_1 := u_0 \dots u_n$ und $weg_2 := w_0 \dots w_m$; u_0 bzw. w_0 ist der jeweilige Startindex und u_n bzw. w_m der Zielindex. Angenommen, die aufsteigende Sortierung sei genau umgekehrt, nämlich $u_n w_n u_0 w_0$. Dann gibt es genau vier mögliche Permutationen dieses Patterns durch Shiften:

$$u_n w_n u_0 w_0 \overset{Links-Shift}{\rightarrow} w_n u_0 w_0 u_n \overset{Links-Shift}{\rightarrow} u_0 w_0 u_n w_n \overset{Links-Shift}{\rightarrow} w_0 u_n w_n u_0 \tag{5.1}$$

Es wird durch das Shiften immer ein Pattern geben, bei dem für jeden Weg der Startindex vor dem Zielindex erscheint. Der Vorteil liegt darin, dass bei der bisherigen Implementierung viele Indizes verglichen werden müssen, besonders auch, weil die Laufzeit des in Kapitel 2 beschriebenen Algorithmus 2.3 BESTIMMEKREUZUNGSWEG($fahrzeug$) vom Wert der Indizes abhängig ist.

Eine weitere, schöne Ergänzung für den Simulator stellt die Einführung von Lichtsignalanlagen dar, die in die existierende Kreuzungslogik bei der Vergleichs-Funktion zweier Fahrzeuge im Knoten einfach zu ergänzen ist. Komplex wird das Thema nur, wenn kreuzungsübergreifende Logik betrachtet wird, wie z. B. Grünphasen.

Der Simulator unterstützt die Möglichkeit, Fahrzeuge erst nach einer bestimmten Anzahl Zeitschritte zu spawnen. Allerdings sind die Möglichkeiten hier recht begrenzt und die Funktion zu nutzen ist noch etwas sperrig. Eine gute Lösung hierfür ist die Einführung von *spawning Flächen*. Das könnten Polygonflächen auf einer Karte sein, in denen in bestimmten (oder zufälligen) Zeitabständen Fahrzeuge auftauchen. Die Bestimmung der Zeitabstände durch eine gegebene Funktion würde das noch weiter optimieren. Das würde erlauben, Tagsezeiten auf recht simple Weise in den Simulator mit einzubringen.

Um das Verhalten der Fahrzeuge noch individueller zu gestalten, könnten verschiedene Fahrzeug- und Fahrerprofile mit unterschiedlichen Trödel- und Überholfaktoren pro Fahrerprofil entworfen werden. So zum Beispiel ein Lastkraftwagenfahrer, der Überholverbot hat und damit einen Überholfaktor von 0. Würde das Nagel-Schreckenberg-Modell genauer diskretisiert werden, indem eine Zellenlänge unter $7{,}5\,\mathrm{m}$ gewählt wird, so könne die Beschleunigung und Fahrzeuglänge ebenfalls individueller gestaltet sein.

Literaturverzeichnis

[Blech1616] W.-D. Obst. *Mehr Blechkolonnen auf den Straßen.* http://www.stuttgarter-nachrichten.de/inhalt.stau-region-stuttgart-mehr-blechkolonnen-auf-den-strassen.6f8890a4-88d4-4cfe-971a-d77573952465.html. Jan. 2016 (zitiert auf S. 38).

[BZBP09] H.-J. Bungartz, S. Zimmer, M. Buchholz, D. Pflüger. *Modellbildung und Simulation.* 12. Springer-Verlag Berlin Heidelberg, 2009, S. 179–210 (zitiert auf S. 13, 40, 43).

[HNR68] P. E. Hart, N. J. Nilsson, B. Raphael. „A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), S. 100–107 (zitiert auf S. 24).

[MCQuad15] D. Parga Cacheiro. *Einführung in Monte Carlo Quadratur.* Techn. Ber. Simulation Software Engineering, IPVS, Universität Stuttgart, 2015 (zitiert auf S. 40, 41).

[MicSimUrbanTraffic9797] J. Esser, M. Schreckenberg. „Microscopic simulation of urban traffic based on cellular automata". In: *International Journal of Modern Physics C* 8.5 (1997) (zitiert auf S. 13).

[MTS16] M. Luz, D. Parga Cacheiro, J.-O. Schmidt. *Microscopic Traffic Simulation.* Techn. Ber. Simulation Software Engineering, IPVS, Universität Stuttgart, 2016 (zitiert auf S. 13, 14, 18, 19, 21–24, 33, 49).

[NaSch92] K. Nagel, M. Schreckenberg. „A cellular automaton model for freeway traffic". In: *Journal de physique I* 2.12 (1992), S. 2221–2229 (zitiert auf S. 16–18).

[OSM17] OpenStreetMap contributors. *OpenStreetMap. Planet dump retrieved from https://planet.osm.org.* https://www.openstreetmap.org. 2017 (zitiert auf S. 15, 49).

[TK11] M. Treiber, A. Kesting. „Verkehrsdynamik und-simulation". In: *Physik Journal* 10.11 (2011), S. 91–107 (zitiert auf S. 13).

Alle URLs wurden zuletzt am 07. 08. 2017 geprüft.

Erklärung

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

Ort, Datum, Unterschrift