Optimization of Vehicular Traffic at Traffic-Light Controlled Intersections

Dissertation
zur Erlangung des akademischen Grades

doctor rerum naturalium
(Dr. rer. nat.)

von Stephan Sorgatz, Master of Science

geboren am 07. April 1988 in Bernburg (Saale)

genehmigt durch die Fakultät für Mathematik
der Otto-von-Guericke-Universität Magdeburg

Gutachter:	Prof. Dr. Sebastian Sager Dr. Holger Poppe Prof. Dr. Volker Kaibel
eingereicht am:	26. September 2016
Verteidigung am:	02. Dezember 2016

The results, opinions or conclusions of this dissertation are not necessarily those of the Volkswagen AG.

Abstract

Assisted and autonomous driving is of current interest and offers great possibilities to improve traffic in terms of different parameters via automatization and cooperation. This thesis aims to investigate multiple methods of optimizing or simply enhancing traffic flow at traffic-light controlled intersections in an urban setting. The developed methods differ in their focus of optimization, say, whether each car is considered individually, or the global traffic is to be optimized. Therefore, different methods for reducing solving times of optimization algorithms, e. g., cutting-plane algorithms, column generation, and branch-and-bound methods, are reviewed, adapted for the particular optimization problems, and evaluated numerically.

A major result of this thesis is that an individual optimization of the cars yields a traffic flow which is just slightly worse than the optimal traffic flow obtained by an optimization of the system from a global point of view. The results that provide this outcome are obtained by extensive traffic simulations where the different methods are implemented and rated in terms of comparable parameters. In parallel, the solving times reveal a great advantage of the individual optimization compared to optimizing traffic flow globally.

The second major purpose of this thesis are proposals for implementations of the presented methods in real-world systems considering, e. g., technical requirements and security issues. It is observed that globally optimal traffic flow is difficult to achieve with reasonable effort. Nonetheless, the obtained solutions can serve as benchmarks for other methods and algorithms, which aim to improve traffic flow.

In addition to calculating optimal solutions, a novel driver-assistance system is presented improving the behavior of an individual car concerning traffic flow. It is already working and has successfully been tested in real-world traffic. With the aid of this system, the car is able to calculate and perform acceleration-trajectories automatically when approaching and passing a traffic-light controlled intersection. A key issue is the wireless communication with the traffic-light. Apart from the individual gain for the particular car, an introduction of the system would lead to better overall traffic flow – even for small equipment rates.

Zusammenfassung

Assistiertes und autonomes Fahren sind hochaktuelle Themen, die großes Potential bieten, Verkehr bezüglich verschiedener Parameter mittels Automatisierung und Kooperation der Fahrzeuge zu verbessern. Die vorliegende Arbeit untersucht verschiedene Methoden der mathematischen Optimierung mit dem Ziel den Verkehrsfluss an innerstädtischen Ampelkreuzungen zu optimieren bzw. zu verbessern. Diese unterscheiden sich unter anderem in ihrem jeweiligen Optimierungsziel. Genauer gesagt darin, ob jedes Auto als einzelnes betrachtet wird oder der globale Verkehrsfluss Ziel der Optimierung ist. Dafür werden verschiedene Methoden zur Beschleunigung des Lösungsprozesses aufgezeigt, auf die entsprechenden Problemstellungen angepasst und numerisch untersucht. Beispielhaft dafür sind Schnittebenenverfahren, column generation, und Branch-and-Bound Methoden.

Ein Hauptergebnis der Arbeit ist, dass eine individuelle Optimierung des Fahrzeugsverhaltens einen nur leicht schlechteren Verkehrsfluss erzeugt als die Optimierung des Systems von einem globalen Betrachtungspunkt aus. Die entsprechenden Ergebnisse werden mittels intensiver Verkehrssimulationen erzielt, worin die jeweiligen Methoden implementiert und bezüglich vergleichbarer Parameter untersucht werden. Parallel dazu zeigen Laufzeitanalysen einen deutlichen Vorteil der individuellen Optimierung des Fahrzeugverhaltens gegenüber der globalen Optimierung des Verkehrsflusses.

Das zweite Hauptanliegen dieser Arbeit ist es, die praktische Umsetzung der vorgestellten Methoden und Algorithmen zu untersuchen, u. a. im Hinblick auf technische Voraussetzungen und Sicherheitskriterien. Dabei wird festgestellt, dass global optimaler Verkehrsfluss nur sehr schwer mit adäquatem Aufwand zu erzielen ist. Dennoch dienen die berechneten Lösungen als Referenz für andere Methoden und Algorithmen, die den Verkehrsfluss an Ampelkreuzungen verbessern sollen.

Neben der Berechnung von optimalen Lösungen wird ein neuartiges Fahrerassistenzsystem vorgestellt, welches das individuelle Verhalten eines Fahrzeugs im Hinblick auf die Verkehrseffizienz verbessert. Es wurde bereits erfolgreich implementiert und im Straßenverkehr getestet. Das Assistenzsystem ermöglicht die automatische Berechnung und Umsetzung von Beschleunigungs-Trajektorien, mit deren Hilfe eine Ampelkreuzung überquert werden kann. Zentral ist dabei eine drahtlose Kommunikation mit der Infrastruktureinheit. Zusätzlich zum individuellen Nutzen für das einzelne Fahrzeug würde eine Einführung des Systems zur Verbesserung des Gesamtverkehrsflusses führen – sogar bei niedrigen Ausstattungsraten.

Danksagung

Mein Dank gilt der Vielzahl an Menschen, die mit ihren Ideen, ihrer Motivation, konstruktivem Feedback und fachlichen Diskussionen einen Anteil zu der Erstellung dieser Arbeit beigetragen haben.

Besonders hervorheben möchte ich Sebastian Sager – nicht nur für seine fachliche Anleitung und seine wertvollen Inputs, die der Grundstein für das Gelingen dieser Arbeit waren – sondern auch für seine Bereitschaft zur Betreuung dieser externen Promotion. Er ermöglichte mir darüberhinaus auch die Teilnahme an mehreren eindrucksvollen und lehrreichen Kongressen.

In diesem Sinne möchte ich auch Florian Kranke, Holger Poppe und Hans-Jürgen Stauss danken, die mir ebenso diese Erfahrungen ermöglicht haben. Darüberhinaus ist es ihnen zu verdanken, dass ich jederzeit meinen Ideen und meiner Arbeit so frei wie möglich nachgehen konnte. Ich möchte mich bei Florian und Holger außerdem für die jederzeit offene Arbeitsatmosphäre bedanken, die ich als äußerst angenehm und fruchtbar empfunden habe und die uns auch in Zukunft tolle Ergebnisse ermöglichen wird. Holger gilt auch besonderer Dank für die Bereitschaft als Gutachter für diese Arbeit zu fungieren.

Weiterhin danke ich allen, auch ehemaligen, Mitgliedern des IMO für ihre ständige Bereitschaft zu Diskussionen und zum Ideenaustausch. Danke insbesondere an Mirko Hahn für die gemeinsame zielführende Arbeit. Großer Dank gebührt auch Jan Krümpelmann und Ferdinand Thein – nicht nur für fachlich relevante Diskussionen und Jans äußerst detailliertes Feedback an die Arbeit, sondern auch für die unzähligen Gespräche über alles andere als Mathematik, die meinen Alltag erhellt haben. Insbesondere möchte ich auch Janick Frasch, Matthias Walter und Stefan Weltge danken. Janick hat mich besonders zu Beginn meiner Promotionszeit tatkräftig unterstützt. Matthias und Stefan waren jederzeit äußerst hilfreiche und motivierte Ansprechpartner – auch schon während der Studienzeit. Großer Dank gilt auch Volker Kaibel. Er hat mein Interesse an der Optimierung geweckt und steht nun, nach Betreuung der Bachelor- und Masterarbeit, auch als Gutachter für die Promotionsschrift zur Verfügung.

Meiner Familie und insbesondere meinen Eltern danke ich für ihre jederzeitige Unterstützung und dafür, dass sie mir stets ermöglicht haben das zu tun, was ich als sinnvoll erachtet habe. Mein abschließender Dank gilt Sandra. Nicht nur dafür, dass sie mich ständig motiviert, herausfordert und bereichert, sondern auch, weil sie die ohnehin schönen Tage noch erlebenswerter macht.

List of Figures

2.1 Visualization of the direct single shooting method 8
2.2 Visualization of the direct multiple shooting method 10
2.3 Visualization of the MPC-scheme 12
2.4 Visualization of the cutting plane method for a two-dimensional linear program 16
2.5 Visualization of a branch-and-bound tree 18
3.1 Scheme of the RACC's cyclic process 26
3.2 Schematic illustration of the different phases during the trajectory-based regimes 28
3.3 Developer HMI of the RACC 32
3.4 User HMI of the RACC 33
3.5 Devices which are installed at multiple intersections in the city of Braunschweig 34
3.6 Example of the information the MIB-HMI shows, when the car approaches a traffic-light a) 35
3.7 Example of the information the MIB-HMI shows, when the car approaches a traffic-light b) 36
3.8 Example of the information the MIB-HMI shows, when the car approaches a traffic-light c) 37
4.1 Simple single-intersection for the global-MILP 42
4.2 Non-convexity of the pair (s_t, χ_t) 44
4.3 Convexification of the pair (s_t, χ_t) 45
4.4 Polyhedron resulting from the constraints, which define an outer-convexification trigger 46
4.5 Polyhedron resulting from the constraints, which define an enter-leave trigger 47
4.6 Polyhedron resulting from the constraints, which define a three-way-split trigger 48
4.7 Visualization of three configurations of indicator variables 49
4.8 Car and corresponding trigger zone for a traffic-light 50
4.9 Variable offset pulse interval 54
4.10 Flow chart of the iterative solving algorithm 63
4.11 Visualization of the bounding heuristic for the tailored branch-and-bound method 68
4.12 Visualization of a situation occurring during the branching step a) . 69
4.13 Visualization of a situation occurring during the branching step b) 70
4.14 Visualization of a situation occurring during the branching step c) . 71
4.15 Visualization of a situation occurring during the branching step d) 71
4.16 Visualization of a situation occurring during the branching step d) 72
5.1 Flow chart of the greedy-algorithm 75
6.1 Overview of the different criteria measured for simulated real-world traffic and the different solving methods 82
6.2 Layout of the single-intersection network 83
6.3 Layout of the four-intersections network 84
6.4 Screenshot of the video data used for calibrating the parameters of the car-following model 86
6.5 Performed velocity by RACC 88

6.6 Performed distance by RACC 89
6.7 Visualization of the RACC using SUMO a) 91
6.8 Visualization of the RACC using SUMO b) 92
6.9 Visualization of the RACC using SUMO c) 93
6.10 Visualization of the RACC using SUMO d) 94
6.11 Visualization of the RACC using SUMO e) 95
6.12 Visualization of the RACC using SUMO f) 96
6.13 Visualization of the RACC using SUMO g) 97
6.14 Visualization of a solution of the global-MILP 100
6.15 Solving times for the tailored branch-and-bound algorithm 104
6.16 Number of visited nodes during the tailored branch-and-bound algorithm 105
6.17 Average solving times per node during the tailored branch-and-bound algorithm 106
6.18 Average waiting time of cars for real-world traffic and the different methods on the single-intersection network 111
6.19 Average fuel consumption of cars for real-world traffic and the different methods on the single-intersection network 112

List of Tables

3.1 Physical quantities used for modeling resistances in the RACC . . . 27
4.1 Possible values for the indicator variables of an enter-leave trigger 47
4.2 Possible values for the indicator variables of an three-way-split trigger 48
6.1 Bounds on the states of the vehicles for the experiments 85
6.2 Adjustable parameters for the default car-following model in SUMO with default values 85
6.3 Parameters for the default car-following model in SUMO with values identified via evaluation of real-world data 86
6.4 Obtained values for real-world traffic on the single-intersection network with different switching schemes for the traffic-lights . . . 87
6.5 Obtained values for real-world traffic on the four-intersections network with different switching schemes for the traffic-lights . . . 87
6.6 Quality of performed RACC-trajectories 90
6.7 Obtained values for traffic induced by the RACC on the single-intersection network 91
6.8 Obtained values for traffic induced by the RACC on the four-intersections network 98
6.9 Values in the experimental setting for parameters of the global-realistic-MILP 99
6.10 Obtained values for the traffic induced by solutions of the global-MILP and global-realistic-MILP 99
6.11 Obtained values for the global-MILP on the single-intersection network 102
6.12 Obtained values for the global-MILP on the four-intersections network 102
6.13 Obtained values for the global-realistic-MILP on the single-intersection network 103
6.14 Obtained values for the greedy-algorithm on the single-intersection network 107
6.15 Obtained values for the greedy-algorithm on the four-intersections network 107
6.16 Obtained values for traffic induced by the greedy-algorithm 108
6.17 Obtained values for traffic induced by the greedy-cruise-control on the single-intersection network 109
6.18 Obtained values for traffic induced by the greedy-cruise-control on the four-intersections network 109

List of Algorithms

2.1 Pseudocode of cutting plane algorithms 14
4.1 Pseudocode of the iterative solving algorithm 64
4.2 Pseudocode of the tailored branch-and-bound algorithm 67
5.1 Pseudocode of the greedy-algorithm 77

Nomenclature

Acronyms

ACC	Adaptive Cruise Control
AIM	Anwendungsplattform Intelligente Mobilität
AU	Application Unit
BMWi	Bundesministerium für Wirtschaft und Energie
C2C	Car-to-Car, Communication via wireless LAN between cars (also: V2V)
C2X	Car-to-X, Communication via wireless LAN between car and generic agent, e.g., traffic-light (also: V2X)
CCU	Communication Control Unit
CAN	Controller Area Network
CAM	Cooperative Awareness Message
DENM	Decentralized Environmental Notification Message
GDP	Generalized Disjunctive Programming
HMI	Human-Machine Interface
IVP	Initial Value Problem
ICR	Iterative Conflict Resolution
LP	Linear Program
MI(N)LP	Mixed Integer (Non-)Linear Program
MPC	Model Predictive Control
MPBVP	Multi-Point Boundary Value Problem
NLP	Nonlinear Program
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
QP	Quadratic Program
RACC	Regime-ACC
RMP	Restricted Master Problem
SQP	Sequential Quadratic Programming
SPaT	Signal Phase and Timings

Symbols

$\lVert\cdot\rVert_2$	Euclidean norm of a vector
$\lVert\cdot\rVert_{\boldsymbol{W}}$	Norm of a vector with respect to symmetric matrix $\boldsymbol{W}$
$\boldsymbol{A}^T, \boldsymbol{x}^T$	Transpose of a matrix or vector
$\boldsymbol{d}$	Vector of path constraints
f	Objective function of a general optimization problem
$\boldsymbol{f}$	ODE-system's right hand side
$\boldsymbol{f}_x$	Jacobian of the vector valued function $\boldsymbol{f}(\cdot)$ with respect to unknown x
φ	OCP's objective function
I	Index set of point constraints
M, M_k	Large positive constant for big-M formulation
N	Number of discretization intervals/stages along time horizon
n_q	Number of control parameters
n_u	Number of control functions
n_x	Number of differential states
$\boldsymbol{q}$	Vector of control parameters
$\boldsymbol{r}$	Vector of point constraints
$\mathbb{R}, \mathbb{R}^+$	Space of (positive) real numbers
$\boldsymbol{s}$	Vector of multiple shooting states
$\mathcal{T}$	Time horizon of the dynamic system
$\mathcal{T}$	Time horizon of the dynamic system
t_0	Beginning of time horizon
t_f	End of time horizon
$\boldsymbol{u}(\cdot)$	Trajectory of process controls
$\bar{\boldsymbol{u}}(\cdot)$	Reference trajectory of control states
$\boldsymbol{\mathcal{U}}$	Set of control functions
$\boldsymbol{x}(\cdot)$	Trajectory of ODE-system states
$\boldsymbol{x}$	Vector of continuous variables
$\bar{\boldsymbol{x}}(\cdot)$	Reference trajectory of system states

$\boldsymbol{x}^*$	Optimal solution of an optimization problem
$\mathcal{X}$	Set of state trajectories
$\boldsymbol{X}$	Set of feasible solutions
x_k	k-th entry of vector $\boldsymbol{x}$
$\boldsymbol{y}$	Vector of integer variables
$\boldsymbol{z}$	Vector of binary indicator variables
$\mathbb{Z}$	Space of integer numbers

Applications

C	Set of cars
$\boldsymbol{\chi}$	Binary trigger indicator
$pred(c)$	Unique predecessor of car c
$\check{s}, \hat{s}$	Lower bound and upper bound on admissible values for $\boldsymbol{s}$
$\bar{t}_c$	Arrival time of car c
$S_{tl}^{start}, S_{tl}^{end}$	Start-position and end-position of trigger zone of traffic-light tl
T	Discretized time horizon
TL	Set of traffic-lights
TZ	Set of trigger zones
T_N	End of the MILPs time-horizon
$\bar{v}_c$	Velocity of car c when entering the network
$\boldsymbol{a}(\cdot)$	Trajectory of acceleration
$\boldsymbol{j}(\cdot)$	Trajectory of jerk
$\boldsymbol{s}(\cdot)$	Trajectory of traveled distance
$\boldsymbol{v}(\cdot)$	Trajectory of velocity

Contents

1 **Introduction** 1

2 **Selected Methods in Optimization** 5
- 2.1 Optimal Control . . . 5
  - 2.1.1 Solution Methods for Optimal Control Problems . . . 6
  - 2.1.2 Model Predictive Control . . . 11
- 2.2 Mixed Integer Programming . . . 13
  - 2.2.1 Cutting Planes . . . 14
  - 2.2.2 Column Generation . . . 15
  - 2.2.3 Branch-and-Bound . . . 17
  - 2.2.4 Modeling Logical Implications . . . 19

3 **Developing a Cruise Control System** 21
- 3.1 Overview . . . 21
- 3.2 Concept of an Assistance System . . . 22
- 3.3 Controller Implementation . . . 26
- 3.4 Technical Requirements and Implementation . . . 29
  - 3.4.1 C2X-Technology . . . 29
  - 3.4.2 Cooperative Traffic-Light . . . 30
  - 3.4.3 Adaptions for the Car . . . 31
  - 3.4.4 Human-Machine Interface . . . 32
  - 3.4.5 Test Drives . . . 33
  - 3.4.6 Acceptance of the System . . . 35
- 3.5 Developing an Information System . . . 36

4 **Optimizing Traffic Flow** 39
- 4.1 Motivation and Setting . . . 40
  - 4.1.1 Macroscopic Traffic Models . . . 40
  - 4.1.2 Microscopic Traffic Models . . . 40
- 4.2 Developing a Mixed Integer Linear Program . . . 41
  - 4.2.1 Cars and Motion Model . . . 41
  - 4.2.2 Triggers . . . 43
  - 4.2.3 Traffic-Lights . . . 49
  - 4.2.4 Objective Function . . . 51
  - 4.2.5 Additional Traffic-Light Regulations . . . 52
- 4.3 Complexity Analysis . . . 56
- 4.4 Digression in Modeling Traffic-Lights . . . 59
- 4.5 Solving Strategies . . . 62
  - 4.5.1 Iterative Solving Algorithm . . . 62
  - 4.5.2 A Tailored Branch-and-Bound algorithm . . . 65

5 **An Individual Approach to Optimizing Traffic Flow** 73
- 5.1 Algorithm . . . 73
- 5.2 Extensions and Adaptions . . . 78
  - 5.2.1 A Real-World System for Optimized Traffic Flow . . . 78
  - 5.2.2 Additional Traffic-Light Regulations . . . 79

6 Numerical Results **81**
- 6.1 Traffic Simulation Software 81
- 6.2 Network and Experimental Setting 82
- 6.3 Real-World Traffic 85
- 6.4 Numerical Results for the RACC 87
  - 6.4.1 Quality of Performed Trajectories 88
  - 6.4.2 Visualization in SUMO 90
  - 6.4.3 Effects on Traffic Flow 91
  - 6.4.4 Effects on Cities 98
- 6.5 Numerical Results for the global-MILP 99
  - 6.5.1 Effects on Traffic Flow 99
  - 6.5.2 Iterative Solving Algorithm 101
  - 6.5.3 Tailored Branch-and-Bound Process 105
- 6.6 Numerical Results for the Greedy-Algorithm 107
  - 6.6.1 Runtimes 108
  - 6.6.2 Effects on Traffic Flow 108
- 6.7 Comparison of Different Approaches and Future Research 110

Bibliography **113**

1 Introduction

In the field of automotive systems, keywords such as *cooperation, connectivity, assistance* and even *autonomous/automated driving* have become more and more visible and frequent in research as well as in public awareness and discussions. In fact, many car manufacturers, research institutes, and related industries have been running a manifold of projects, e. g., [30, 83, 92, 93, 104], and making efforts to develop and evaluate the impact of systems in the context of these keywords. We want to highlight two of them: *Autonomous driving* and *cooperation*. At first glance, both of them are related to each other in a sense of *smart mobility*, which is another of these – often rather fuzzily defined – keywords. Going a little bit deeper into detail, we can easily distinguish the main properties. Autonomous driving, which is the ultimate consequence of assisted driving, simply considers the movement of the individual car. Simply put, an autonomously driving car mainly cares about moving to its destination while respecting legal regulations and hopefully not being involved in any accidents. Qualifying quantities such as *traffic flow* are of minor interest and therefore barely or not considered at all in current implementations. On the other hand, there are cooperative systems. In the course of this thesis, we will also refer to any implementation of algorithms in a car as *application*. These cooperative applications are in general independent from assisted or even autonomous driving and aim to generate benefit for the involved cars, or even infrastructural devices, by exchanging information. A very simple example for a cooperative system following this definition are blinkers for indicating the intention of a turning maneuver.

The growing field of technology and devices for wireless communication promises to facilitate the process of exchanging information between traffic participants. Among other technologies, wireless LAN, which is in the automotive context often referred to as *Car-to-Car (C2C)* and *Car-to-X (C2X)*, is of current interest. It offers sufficiently wide ranges, short delays, and direct communication between agents. Often these technologies are also referred to as *Vehicle-to-Vehicle (V2V)* and *Vehicle-to-X (V2X)*. Putting these two major concepts together seems to be beneficial for both of them: cooperative systems promise to be more efficient and consistent if the intended maneuvers or strategies are performed automatically. On the other hand, autonomous driving will be certainly much more comfortable, safe, and efficient in terms of traffic flow if an automatically driving car permanently receives information about other cars' statuses and intentions. One can imagine a manifold of situations these considerations might apply for.

Talking about traffic flow, bottlenecks often arise whenever the infrastructure is either not capable of the sheer amount of cars, e. g., if the amount of lanes is not sufficiently large, or static obstructions hinder the movement of cars. One kind of these obstructions are intersections. A high potential considering an improvement of traffic flow lies in the individual behavior of each driver. Maneuvers of deceleration and acceleration, stopping and starting, and delays due to reaction time are often rather inefficient in nowadays traffic. It seems promising to support the driver by providing information beyond his or her perception, or to let the car perform certain maneuvers automatically. Here, cooperation certainly comes into play. The consequence of this consideration are algorithms for autonomously driving cars to pass an intersection, including the objective to improve the overall traffic flow. Due to a considerable amount of technology, which is needed for exchanging information and other purposes, traffic-light controlled intersections provide at least an infrastructure that can be extended and are therefore a considerable setting for first applications.

Problem Setting and Contribution

Consider the situation of a car approaching a traffic-light controlled intersection. As soon as the lights are visible, the driver performs certain maneuvers based on the current signal state. We can distinct two major situations: the traffic-light is currently green, and the traffic-light is currently red. In the first case, the driver will keep his or her desired velocity and aim to pass the intersection. In the latter case, the driver performs a stopping-maneuver in front of the traffic-light's stopping line and accelerates as soon as the light switches to green. (Of course, in practice other influences, e. g., preceding cars or the duration of the current light phase are also considered. We will incorporate them in the course of this thesis.) Briefly speaking, the driver aims to pass the stopping line as soon as possible and as fast as possible while respecting legal regulations such as red lights and speed limits. Information about upcoming switches of the traffic-light's state would be beneficial for this. Simple visual systems, which display the remaining time of the current phase and can be considered to be cooperative, are employed in some parts of the world.

In this thesis, we present a novel driver-assistance system, which aims to automatically perform maneuvers in a car, reducing the time between the traffic-light's switch to green and the passage of the car. Simultaneously, the performed velocity when passing the stopping line is maximized without violating legal regulations. The core of this application are online calculations of acceleration-trajectories that are performed automatically. To this end, we make use of optimal control problems and suitable solving methods. A crucial part is wireless communication between the car and the traffic-light which shares necessary information about the intersection area as well as current and future signal states. More specifically, the car does not influence the traffic-light's behavior. Communication between these two agents is restricted to one direction. Beyond the pure design of the application, it is already running in a car and has been successfully tested on a test field and in real-world traffic. As it is a priori not clear how drivers react to a novel assistance system that controls the car's acceleration rather dynamically, we provide references to studies concerning the acceptance of the strategies performed by our application. Moreover, we present another novel assistance system which is purely for information purposes. According to the discussion above, both assistance systems are considered to be cooperative and based on an individual point of view.

Besides this approach, we are also interested in considering traffic from a global point of view. We are given a network of roads and a set of cars with their respective route they take through the network. Moreover, the point in time each car enters the network is given as well as the velocity at this point in time. We want to determine the behavior of all cars and traffic-lights in the network of roads in such a manner as to induce a best possible traffic flow. To this end, we consider *mixed integer linear programs (MILPs)* and elaborated methods for solving them. As the derived model turns out to be of rather high complexity, we introduce different methods to facilitate the solving process, including a *cutting plane method*, *column* *generation*, and a tailored *branch-and-bound algorithm*. For a resulting iterative solving method, results on the maximum number of iterations are provided. After all, the calculation of the optimal behavior of all cars and traffic-lights turns out to be time-consuming and is hence not suitable for real-world applications. In consequence, we design and implement an algorithm that manages the passages of cars over an intersection while considering realistic driving maneuvers. In contrast to the optimization of the global traffic flow, each car aims to optimize its passage individually. The algorithm is also proposed to be implemented in a real-world application based on cooperation between the cars and the traffic-light. A major advantage is that also usual cars, which do not run this application, can be considered. Again, the method is also regarded in terms of a bound on the maximum number of optimization problems to be solved.

Finally, we investigate the different approaches in terms of their respective influence on nowadays real-world traffic with the aid of systematic numerical test series. To this end, a *microscopic traffic simulation* software is used, where the different algorithms are implemented and the resulting simulated traffic flow is evaluated. Therefore, an elaborated *car-following model* is parameterized with the aid of detailed visual recordings in a way that it mirrors the behavior of real-world cars near an intersection in a satisfying manner. The discussion includes the comparison of globally optimized traffic flow and individually optimized behavior of single cars. Besides traffic flow and emissions, the performances of the different solving methods are also evaluated. The main result is that all methods achieve enhancements in traffic flow. It is worth mentioning at this point that traffic flow induced by the individual MILP-based optimization algorithm is just slightly worse than traffic flow induced by global optimization. Whereas the solving times for global optimization are much higher. Compared to nowadays real-world traffic, reductions in waiting time of up to 28 % are achieved by the driver-assistance system, which is already running in a car. Both MILP-based approaches induce reductions of up to 99 %. In parallel, savings in fuel consumption of up to 19 % for the driver-assistance system and 54 % for both MILP-based approaches are obtained. Besides its influence on traffic flow, the presented assistance system is also regarded in terms of the deviation of calculated trajectories and actually performed ones by the car during test drives.

Outline of the Thesis

Before we introduce and discuss different methods of optimizing traffic at traffic-light controlled intersections, we provide an overview of theory and methods in the field of mathematical optimization in Section 2, which are employed in the course of this thesis. The first part of this section considers *optimal control* providing concepts and methods to solve dynamic optimization problems which arise in many practical problem settings. *Direct shooting methods* and *model predictive control* are of special interest. The second part of Section 2 deals with finite-dimensional optimization problems, considering those, which contain integer variables. We revisit different methods for solving mixed integer programs or complex optimization problems in general, e. g., cutting plane methods, column generation, branch-and-bound algorithm, and common methods to model *logical implications*.

In Section 3, we present the two novel driver-assistance systems based on wireless communication between traffic-light and cars. To this end, we make use of the definitions and solving methods stated in the previous section. Besides the pure concept, also technical issues and requirements are discussed.

In Section 4, we investigate optimized traffic flow from a global point of view, which considers not only the motion of all cars in the network but also the traffic-light's signal states. Therefore, a novel mixed integer optimization problem is formulated, whose solutions can be calculated offline and afterwards be performed automatically by the cars and traffic-lights. Besides a possible, but as of now hard to achieve, practical implementation, the obtained solutions can serve as benchmarks for other methods or algorithms which improve or somehow influence traffic flow. This includes the application developed in Section 3. In addition to deriving the model and discussing further possibilities to represent the problem, we investigate the model's complexity by using concepts of scheduling theory. Finally, we present different solving methods which exploit the problem's structure, e. g., by adding cutting planes and performing a column-generation approach. For the resulting iterative solving method results proving a termination in finitely many steps and providing the maximum number of iterations are derived. Additionally, a tailored branch-and-bound algorithm is introduced. Heuristics which can be applied optionally are also discussed.

For a real-world application fast feedback times and recalculations due to unforeseen events are crucial. Additionally, traffic participants that do not run the respective system must be taken into account. Thus, we present in Section 5 a novel algorithm and propose a cooperative assistance system which does not only schedule a car's passage over an intersection by calculating acceleration-trajectories, but also negotiates the exact time of passing the intersection with the traffic-light. Thus – in contrast to the application of Section 3 – the communication between these two agents has to be bidirectional. Besides the design of the algorithm and a discussion on a possible real-world system, theoretical results on the maximum number of performed iterations are provided.

Section 6 deals with numerical investigations of real-world traffic and traffic induced by the different methods developed in the course of this thesis. To this end, efficient realizations of the solving methods for the MILP developed in Section 4 are implemented. This includes the cutting plane and column generation algorithms, as well as the branch-and-bound algorithm. Also, the driver-assistance system which is already running in a car, and the proposed application of Section 5 are implemented and investigated numerically. Besides the effects on traffic flow, solving times and other issues concerning the solving process of the MILP-based approaches are considered.

2 Selected Methods in Optimization

In this section, we discuss the theoretical background of the methods and algorithms that are presented in the following sections and serve for improving or optimizing traffic flow. First, in Section 3, we discuss a driver-assistance system, which has been implemented and successfully tested in a car. In this discussion, we use concepts and solution methods in the context of optimal control, cf. Section 2.1. In Section 4, we develop and implement a mixed integer linear program that models optimal traffic flow at traffic-light controlled intersections from a central point of view. This way, we obtain solutions for the movements of all cars and the behavior of the traffic-lights which provide a best possible traffic flow for the whole system. Furthermore, an MILP-based algorithm is presented in Section 5 which regards each car individually. Both concepts make use of methods discussed in Section 2.2.

For the purpose of this discussion, we distinguish between infinite-dimensional and finite-dimensional optimization problems. For both types, solving methods are presented and in the course of this thesis adapted for our purposes.

First, we define a classical optimization problem, which asks for finding an optimum object in a set of objects. This set of objects contains all *feasible solutions* and is therefore called *feasible set*. The term *optimum object* refers to a given *objective function* whose domain is a superset of the feasible set and which is to be minimized or maximized. A generic optimization problem can be written as

$$\begin{array}{ll} \text{min / max} & f(\boldsymbol{x}) \\ \text{subject to:} & \boldsymbol{x} \in \boldsymbol{X}, \end{array}$$

where $f : \boldsymbol{X} \to \mathbb{R}$ is the objective function which is to be minimized or maximized and $\boldsymbol{X}$ denotes the feasible set. In this thesis, we make a major distinction between optimization problems whose feasible set is a subset of an infinite-dimensional set and those whose feasible set is a subset of a finite-dimensional set.

2.1 Optimal Control

In many practical applications, processes arise which can be tracked over a continuous time horizon $\mathcal{T} := [t_0, t_f] \subseteq \mathbb{R}$. A common way to represent the process is through its time dependent *differential state* $\boldsymbol{x} \in \mathcal{X} := \{\boldsymbol{x} : \mathcal{T} \to \mathbb{R}^{n_x}\}$. Additionally, we assume the process to be influenced by measurable and bounded *control functions* $\boldsymbol{u} \in \mathcal{U} := \{\boldsymbol{u} : \mathcal{T} \to \mathbb{R}^{n_u} \mid \boldsymbol{u} \text{ measurable and bounded}\}$. The dynamics of the process over time are commonly represented as a system of *ordinary differential equations (ODEs)*:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \quad \forall t \in \mathcal{T},$$

with $\boldsymbol{f} : \mathcal{T} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$. Considerations about finding a state trajectory $\boldsymbol{x}(\cdot)$ and control trajectory $\boldsymbol{u}(\cdot)$ which satisfy specific conditions and are optimal in a certain sense lead to the definition of an *optimal control problem*. In the remainder of this section, we follow presentations of problem formulations and solution methods similar to those found in [29, 51, 56, 89].

Definition 2.1 (Optimal Control Problem (OCP)). *An optimal control problem is an infinite-dimensional constrained optimization problem of the form:*

$$
\begin{aligned}
\min_{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)} \quad & \varphi(\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)) && && (2.1)\\
\text{subject to:} \quad & \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t,\boldsymbol{x}(t),\boldsymbol{u}(t)) && \forall t \in \mathcal{T}, && (2.2)\\
& 0 \leq \boldsymbol{p}(t,\boldsymbol{x}(t),\boldsymbol{u}(t)) && \forall t \in \mathcal{T}, && (2.3)\\
& 0 \leq \boldsymbol{r}(t_k,\boldsymbol{x}(t_k)) && \forall \{t_k\}_{k\in I} \subseteq \mathcal{T}. && (2.4)
\end{aligned}
$$

The aim is to determine state and control trajectories $\boldsymbol{x}(\cdot)$ *and* $\boldsymbol{u}(\cdot)$ *for the process dynamics* (2.2) *minimizing the objective function* $\varphi : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$. *Additionally, these trajectories have to respect path constraints* $\boldsymbol{p} : \mathcal{T} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_p}$ *and point constraints* $\boldsymbol{r} : \mathcal{T} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_r}$ *at discrete time points* $t_k \in \mathcal{T}$ *with a finite index set* $I = \{1,\dots,n_I\}$.

In order to ensure existence and uniqueness of the ODE's solution, we assume $\boldsymbol{f} : \mathcal{T} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ to be piecewise Lipschitz continuous in Definition 2.1. Therefore, we refer to the well-known Picard-Lindelöf theorem. The path constraints (2.3) may contain restrictions on admissible values of the state and control trajectories and boundaries for the trajectories. The point constraints (2.4) allow to model constraints on the states on a finite set of points $\{t_k\}_{k\in I} \subseteq \mathcal{T}$. Here, intial states, which are crucial for the existence and uniqueness of the solution, as well as terminal states may be invoked. The OCP is clearly infinite-dimensional as the control trajectory $\boldsymbol{u}(\cdot)$ and state trajectory $\boldsymbol{x}(\cdot)$ are the unknowns which are to be determined. Regarding the objective function, we present two variants:

Objective Functions The OCP's objective function $\varphi(\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot))$ is a general function that usually consists of a *Lagrange type*, which is an integral contribution, and a *Mayer type*, which is a contribution in the horizon's end-point. If the objective consists of both types, it is of *Bolza type*:

$$\varphi(\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)) = \int_{t_0}^{t_f} \boldsymbol{l}(t,\boldsymbol{x}(t),\boldsymbol{u}(t))\, dt + \boldsymbol{m}(t_f,\boldsymbol{x}(t_f)).$$

Problems where the deviation of the states and controls from a desired trajectory is to be minimized, e. g., the application presented in Section 3, often require a *least-squares objective*, which is of the general form:

$$\varphi(\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)) = \int_{t_0}^{t_f} \|\boldsymbol{l}(t,\boldsymbol{x}(t),\boldsymbol{u}(t))\|_2^2\, dt + \|\boldsymbol{m}(t_f,\boldsymbol{x}(t_f))\|_2^2.$$

2.1.1 Solution Methods for Optimal Control Problems

In this section, we give an overview of different solution methods for optimal control problems. These differ in the type of discretization, the precision of the obtained solutions, and suitability for implementations. Furthermore, *indirect methods* and *direct methods* are distinguished. While for indirect methods the optimization is performed in an infinite-dimensional function space, direct methods apply a transformation to a finite-dimensional space before the optimization. We will present a brief outline of both types. As the application presented in Section 3 is solved via direct multiple shooting, which is a direct method, we will mainly focus on this type. Indirect methods are briefly discussed for the sake of completeness.

2.1.1.1 Indirect Methods Indirect methods go back to the work of Pontryagin, cf. [82], and are based on the so-called *first-optimize-then-discretize* scheme. The optimization takes place in an infinite-dimensional function space and the necessary conditions of optimality are used to transform the OCP into a so-called *multi-point boundary value problem (MPBVP)* using the *maximum principle*. The MPBVP is then solved numerically by appropriate numerical methods, cf. [80].

The main advantage of indirect methods is a high accuracy regarding the optimal solution, as the problem itself is solved analytically. However, this is also the major disadvantage. For formulating the optimality conditions for each problem mathematical insight is required. For large scale problems, which often arise in practical applications, the derivation can be very difficult. Moreover, even slight changes in initial states or the model itself, e. g., introducing an additional constraint, can make a repetition of these steps necessary. Therefore, indirect methods have not become a suitable tool for fast numerical solutions of optimal control problems.

2.1.1.2 Direct Single Shooting In contrast to indirect methods, direct solution methods are based on the so-called *first-discretize-then-optimize* scheme. In the first step, the OCP is discretized leading to a finite-dimensional optimization problem which can be solved by nonlinear programming techniques. In both direct single shooting and direct multiple shooting, the control functions $\boldsymbol{u}(\cdot)$ are discretized. Therefore, we choose $N+1$ fixed discretization points

$$t_0 < t_1 < \ldots < t_N := t_f$$

defining a not necessarily equidistant time grid on $\mathcal{T}$, which we call the *shooting grid*. For simplicity, we assume the shooting grid to coincide with the constraint grid of the point constraints used in Definition 2.1. However, the following argumentation can be extended to differing grids for the controls and point constraints. On each of the resulting intervals $[t_k, t_{k+1}]$, $0 \leq k \leq N-1$, the control functions $\boldsymbol{u} \in \mathcal{U}$ are piecewise approximated using finitely many control parameters $\boldsymbol{q} = (\boldsymbol{q}_0, \ldots, \boldsymbol{q}_N)$:

$$\boldsymbol{u}(t) \approx \boldsymbol{\nu}_k(t, \boldsymbol{q}_k), \quad \forall t \in [t_k, t_{k+1}],\ k \in \{0, \ldots, N-1\}.$$

Typically, the functions $\boldsymbol{\nu}_k$ are of linear or constant type, cf. [56, 89]. In the constant case and assuming $n_u = n_q$, this leads to an approximation of the controls via

$$\boldsymbol{u}(t) \approx \boldsymbol{q}_k, \quad \forall t \in [t_k, t_{k+1}],\ k \in \{0, \ldots, N-1\}.$$

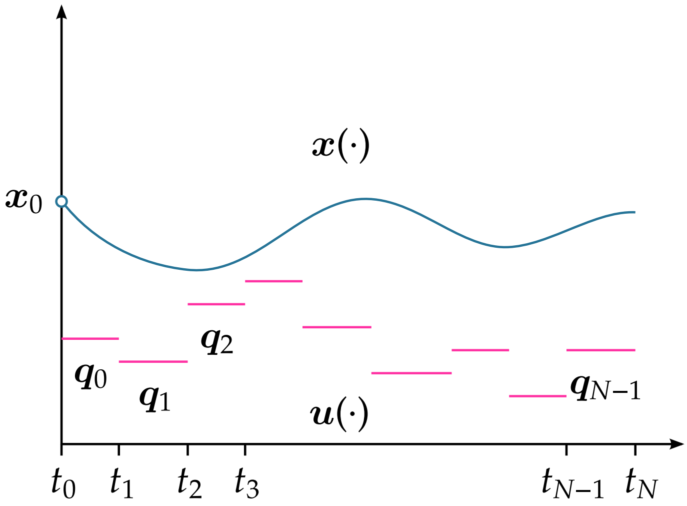


Figure 2.1: Visualization of the direct single shooting method with discretized controls and state trajectory, which is obtained by integration.

For completeness, the control at the final step is set as

$$q_N := q_{N-1}.$$

The states $x(\cdot)$ are regarded as dependent variables on $[t_0, t_f]$. Numerical integration is used to obtain the state as function $x\,(t; q)$ of the finitely many control parameters. In each iteration of the solving process, an ODE has to be solved. The OCP can then be viewed as an NLP in the $n_x + Nn_q$ unknowns x_0 and q which can be solved to local optimality using a finite-dimensional NLP-solver, e. g., by using *sequential quadratic programming (SQP)*, cf. [56, 89]. Path constraints are commonly discretized and enforced on the discretization grid only. Note that the point constraints are already present in a discrete version. It might happen that the discretized path constraints are violated in the interior of the grid intervals. If this effect is not neglectable, which is the case in most practical applications, one could enforce the constraints on a even finer subgrid. Figure 2.1 illustrates the discretization scheme of the direct single shooting method.

An advantage of the approach is that it can be implemented rather straightforwardly if suitable solvers are available. Additionally, the number of unknowns in the resulting NLP is relatively small. However, direct single shooting comes with two drawbacks. While knowledge about the controls can be brought in, no knowledge about the process itself, i. e., $x(\cdot)$ except the initial values, can be used. Moreover, if the initial guess for the initial state x_0 is too far away from the optimal solution, a singularity may exist and no solution of the IVP is available. In practice, a close enough initial guess to prevent this may be hard to obtain. Even if a solution exists, it may not be computed numerically due to propagation of errors over the course of the integration. Depending strongly on the non-linearity of the process, the error propagation may lead to a singularity even if the initial values were quite good.

2.1.1.3 Direct Multiple Shooting Direct multiple shooting originates in ideas of Bock and Plitt, cf. [10], and is also a direct method. In contrast to direct single shooting, the states are not regarded as dependent variables, but also discretized on the shooting grid $\{t_k\}$ using new variable vectors $\boldsymbol{s}_k \in \mathbb{R}^{n_x}$. These *shooting variables* serve as initial values on the resulting N independent IVPs on the intervals $[t_k, t_{k+1}]$:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{q}_k) \quad \forall t \in [t_k, t_{k+1}],\ k \in \{0, \ldots, N-1\} \\ \boldsymbol{x}(t_k) &= \boldsymbol{s}_k. \end{aligned} \tag{2.5}$$

Note that $\boldsymbol{s}_N$ is not an initial value for an IVP, but used to check terminal costs and constraints. Again, the path constraints are discretized and treated as in direct single shooting leading to

$$0 \leq \boldsymbol{p}(t_k, \boldsymbol{s}_k, \boldsymbol{q}_k) \quad \forall k \in \{0, \ldots, N\}$$

instead of (2.3). At this point, the systems (2.5) are not necessarily continuous in the grid points. Hence, we introduce additional *matching constraints*

$$\boldsymbol{s}_{k+1} = \boldsymbol{x}_k(t_{k+1}; \boldsymbol{s}_k, \boldsymbol{q}_k) \quad \forall k \in \{0, \ldots, N-1\}.$$

Here, the notation $\boldsymbol{x}_k(t_{k+1}; \boldsymbol{s}_k, \boldsymbol{q}_k)$ denotes the value $\boldsymbol{x}(t_{k+1})$ which is obtained as the solution of (2.5) on the interval $[t_k, t_{k+1}]$ with initial values $\boldsymbol{x}(t_k) = \boldsymbol{s}_k$ and applying the controls $\boldsymbol{u}(t) = \boldsymbol{q}_k$ on $[t_k, t_{k+1}]$. The resulting NLP in $(N+1)n_x + Nn_q$ unknowns finally reads as

$$\min_{\boldsymbol{s}, \boldsymbol{q}} \quad \sum_{k \in \{0, \ldots, N\}} l(t_k, \boldsymbol{s}_k, \boldsymbol{q}_k) \tag{2.6}$$

$$\text{subject to:} \quad \boldsymbol{s}_{k+1} = \boldsymbol{x}_k(t_{k+1}; \boldsymbol{s}_k, \boldsymbol{q}_k) \quad \forall k \in \{0, \ldots, N-1\}, \tag{2.7}$$

$$0 \leq \boldsymbol{p}(t_k, \boldsymbol{s}_k, \boldsymbol{q}_k) \quad \forall k \in \{0, \ldots, N\}, \tag{2.8}$$

$$0 \leq \boldsymbol{r}(t_k, \boldsymbol{s}_k) \quad \forall k \in \{0, \ldots, N\}. \tag{2.9}$$

Note that we assumed $I \subseteq \{0, \ldots, N\}$ for notational simplicity leading to a different notation of the originally discrete point constraints in (2.9). Also, the matching constraints (2.7) might not be satisfied during the iterations of the nonlinear programming algorithm used to solve the NLP, but are satisfied when convergence has been achieved. The NLP can be solved to local optimality with tailored iterative methods, e. g., the aforementioned SQP-methods. The objective (2.6) is formulated here in a discretized form $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}$. Figure 2.2 visualizes the concept of the direct single shooting method.

An advantage of the direct multiple shooting in contrast to the single shooting is that a priori knowledge about the states can be brought in via the initial values $\boldsymbol{s}_k$, leading to a faster convergence of the system. Additionally, this method shows a higher stability, as the time horizons the IVPs are solved on are much

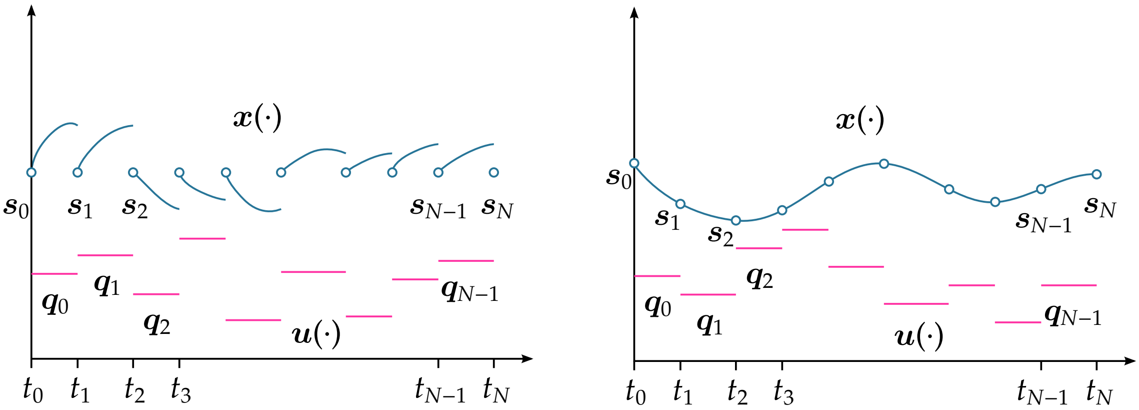


Figure 2.2: Visualization of the direct multiple shooting method with discretized controls and state trajectories, which are obtained by piecewise integration. On the left, a non-converged state is depicted with violated matching constraints. On the right-hand-side, the matching constraints are fulfilled and the solution is a solution of the original problem.

smaller as in the single shooting. System architecture permitting, the different IVPs may be solved simultaneously using state-of-the-art solvers. On the other hand, the resulting NLP is larger as more variables, namely $\boldsymbol{s}_k$, enter the problem compared to single shooting. With the aid of the so-called *condensing step* many of these variables can be eliminated. We kindly refer the reader to [10] for detailed information about this dimension reduction technique, which can be improved for sparse systems according to [29, 68].

As the direct multiple shooting approach offers the aforementioned beneficial criteria and is available through efficient implementations, it is the method of choice for many practical applications.

2.1.1.4 Direct Collocation Collocation methods go back to [100] and have been refined, e. g., in [9, 47, 88, 102]. Again, both controls and states are discretized on a time grid with N intervals. As we have seen in the direct multiple shooting, the discretized states on the grid points read as $\boldsymbol{x}(t_k) = s_k$. One-step or multi-step integration methods, called *collocation scheme*, are applied in the intervals $[t_k, t_{k+1}]$ to connect the states on the grid. Thus, the ODE system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)), \quad \forall t \in \mathcal{T}$$

is replaced by finitely many equality constraints, e. g., by the explicit Euler approximation

$$\frac{s_{k+1} - s_k}{t_{k+1} - t_k} = \boldsymbol{f}(t_k, s_k, q_k) \quad \forall k \in \{0, \ldots, N-1\}.$$

Point constraints and path constraints are included in a similar way as in the shooting methods. The resulting NLP consists of $(Nm + 1)n_x + Nmn_u$ unknowns, where m denotes the number of vectors of the collocation scheme in a single interval $[t_k, t_{k+1}]$. In the presented Euler case, it simply holds that $m = 1$. The Lagrange term of the objective is replaced by a summation term on the grid points. The NLP can be solved with state-of-the-art NLP-solvers, e. g., interior point methods.

As in direct multiple shooting, collocation also allows us to use knowledge about the process behavior in the initialization of the optimization problem. Moreover, perturbations do not spread over the whole time horizon, as small violations of the matching constraints over the course of the NLP dampen them out. Although the problem can get very large, depending on the specific collocation scheme, it is rather sparse and efficient solution methods exist, e. g., [108]. However, a drawback of collocation is the difficulty to include adaptivity of the ODE solution process. This means that highly nonlinear or stiff systems require a very fine collocation scheme in regions we are not aware of beforehand. Introducing such a very fine grid on the whole horizon leads to a very large NLP in the number of variables. There are methods which try to overcome this issue by starting with a coarse grid and refining it during the solution process, cf. [8].

In the current setup, we can solve NLPs on a fixed time horizon. In many practical applications, e. g., an acceleration controller in a car, the observable system states can change very rapidly. For instance, because of unpredictable changes in the road surface or other cars which might suddenly appear. Moreover, it is very difficult to realize the calculated controls in practice in a sufficiently exact way. This usually leads to big differences in the calculated system states and the performed ones in the real world. Normally, the effect grows the further we move on in the time horizon. We therefore discuss *model predictive control (MPC)* as an iterative method to face these issues in the next section.

2.1.2 Model Predictive Control

In practice, one often wants to control a process over a relatively long time horizon. Many applications aim to induce system states which should be as close as possible to desired reference states. Therefore, the objective of the OCP which has to be solved is of a least squares type:

$$\min_{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)} \int_{t_0}^{t_f} \|\boldsymbol{x}(t) - \bar{\boldsymbol{x}}(t)\|_{\boldsymbol{W}}^2 + \|\boldsymbol{u}(t) - \bar{\boldsymbol{u}}(t)\|_{\boldsymbol{Q}}^2 \,\mathrm{d}t + \|\boldsymbol{x}(t_f) - \bar{\boldsymbol{x}}(t_f)\|_{\boldsymbol{P}}^2$$

Here, $\bar{\boldsymbol{x}}(\cdot)$ and $\bar{\boldsymbol{u}}(\cdot)$ denote the reference values of the states and controls. The expression $\|\boldsymbol{a}\|_{\boldsymbol{A}} := \sqrt{\boldsymbol{a}^T \boldsymbol{A} \boldsymbol{a}}$ denotes the norm induced by symmetric weighting matrices $\boldsymbol{W} \in \mathbb{R}^{n_x \times n_x}, \boldsymbol{Q} \in \mathbb{R}^{n_u \times n_u}$, and $\boldsymbol{P} \in \mathbb{R}^{n_x \times n_x}$. These are chosen to be positive semidefinite and allow to realize a prioritization of particular states and controls in terms of achieved distance between actual and desired values. Often, the reference trajectories are chosen to be equilibria of the system, but it is also possible to consider dynamic trajectories.

As already mentioned above, it is often necessary in practical applications to be able to handle unforeseen changes in the system's environment and to react to imprecise realizations of the calculated controls. Moreover, the measured system states which were used to initialize the OCP can be fraught with inaccuracies. Another reason that makes it necessary to update the calculated controls and states is the ODE model itself. As in applications, natural or very complex technical

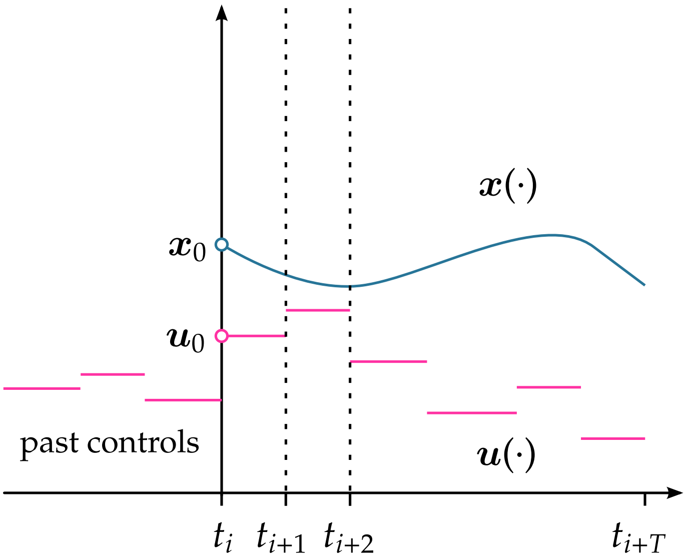


Figure 2.3: Visualization of the MPC-scheme. At discrete time points $t_0, \ldots, t_i, \ldots$, which do not need to coincide with the discretization points of the controls, an OCP is solved, e. g., by one of the methods described above. The values for the initial states $\boldsymbol{x}_0$ and current values of the controls $\boldsymbol{u}_0$ for each OCP are obtained by measurements. Here, the controls are piecewise approximated by constant values.

systems are regarded, the underlying ODE of the OCP is a more or less good theoretical description of the process. Thus, even with a highly accurate realization of the calculated controls and unbiased measurement of the initial states, the actual system states may differ from the calculated ones.

This leads us to the idea of solving not only a single instance of the OCP, but rather a series of OCPs on a moving time horizon. More specifically, at every time instant t, an optimal control problem with initial value $\boldsymbol{x}_0(t)$ is solved, e. g., with one of the direct methods mentioned above. The obtained control $\boldsymbol{u}(t)$ is fed back into the system. In practice, these steps of measuring the system states, solving the OCP for the particular time step, and feeding the control into the system is performed at discrete sampling times $t_0, t_1, \ldots$. This is mainly due to the fact that the computation of the optimal controls and feeding them into the system cannot be done instantaneously. Also, measuring the actual system states, which serve as initial values of the actual problem, takes a certain amount of time. Figure 2.3 illustrates the concept of the MPC algorithm as iterative process on the time horizon.

When applying an MPC-based algorithm in practice, usually many deviations from this idealized description occur. First, not the exact optimal control problem is solved, but rather an approximation, e. g., by one of the methods presented in Section 2.1. Beyond this, a high delay between measuring the system states and applying the controls, e. g., due to high solving times, may result in unacceptable deviations between the desired and actual system states. Also, it may not always be possible to measure all required states at every sampling time. Additionally, these measurements may not be sufficiently exact. In the literature, a variety of methods exists which deal with these issues. As it is not the purpose of this thesis to develop solution methods for the MPC-scheme, we kindly refer the reader to [18, 56] for further information on *real-time iterations*, and to [29, 85] for an insight in *moving-horizon estimation*. In Section 3, we present a driver-assistance system, which is already working in a car and making use of MPC and direct multiple shooting for solving the OCPs.

2.2 Mixed Integer Programming

In contrast to the problems in the last section, we now consider optimization problems which are finite-dimensional. First, we define the problem class of *mixed integer nonlinear programs*.

Definition 2.2 (Mixed Integer Nonlinear Program (MINLP)). *A mixed integer nonlinear program is a finite-dimensional optimization problem in the unknowns* $\boldsymbol{x} \in \mathbb{R}^{n_x}$ *and* $\boldsymbol{y} \in \mathbb{R}^{n_y}$ *with an additional requirement for* $\boldsymbol{y}$ *to be integral,*

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{y}} \quad & f(\boldsymbol{x},\boldsymbol{y}) \\ \text{subject to:} \quad & \boldsymbol{g}(\boldsymbol{x},\boldsymbol{y}) \leq 0, \\ & \boldsymbol{x} \in \mathbb{R}^{n_x}, \\ & \boldsymbol{y} \in \mathbb{Z}^{n_y}, \end{aligned} \tag{2.10}$$

with an objective function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$ *and a function* $\boldsymbol{g} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_g}$ *that implies constraints on the set of permissible solutions. Both are considered to be twice continuously differentiable. In case that no integral vector* $\boldsymbol{y}$ *is present, we call it a nonlinear program (NLP).*

Note that the objective function in the definition above is to be maximized as this will be the case in the mixed integer models we are discussing in the course of this thesis. Transformations between maximizing and minimzing an objective function are simply realized via multiplication with -1. MINLPs are known to be $\mathcal{NP}$-hard, cf. [32]. This means, if $\mathcal{NP} \neq \mathcal{P}$, then there exist instances of problem (2.10), which cannot be solved in polynomial time with respect to the size of the problem formulation. Of particular interest in this thesis are problems where f and $\boldsymbol{g}$ are of linear type. The optimization problem (2.10) is then called *Mixed Integer Linear Program*. We give a formal definition as follows.

Definition 2.3 (Mixed Integer Linear Program (MILP)). *A mixed integer linear program is an optimization problem in the unknowns* $\boldsymbol{x} \in \mathbb{R}^{n_x}$ *and* $\boldsymbol{y} \in \mathbb{R}^{n_y}$ *with an additional requirement for* $\boldsymbol{y}$ *to be integer,*

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{y}} \quad & \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{d}^T\boldsymbol{y} \\ \text{subject to:} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{y} \leq \boldsymbol{b}, \\ & \boldsymbol{x} \in \mathbb{R}^{n_x}, \\ & \boldsymbol{y} \in \mathbb{Z}^{n_y}. \end{aligned} \tag{2.11}$$

The matrices $\boldsymbol{A} \in \mathbb{R}^{m_1} \times \mathbb{R}^{n_x}$ *and* $\boldsymbol{D} \in \mathbb{R}^{m_2} \times \mathbb{R}^{n_y}$ *as well as the vector* $\boldsymbol{b} \in \mathbb{R}^{m_1+m_2}$*, with* $m_1, m_2 \in \mathbb{N}$*, define linear constraints on the set of permissible solutions. Additionally, the linear objective function can be expressed via vectors* $\boldsymbol{c} \in \mathbb{R}^{n_x}$ *and* $\boldsymbol{d} \in \mathbb{R}^{n_y}$*. In case that no*

Algorithm 2.1: Pseudocode of cutting plane algorithms. A solution of the original problem is determined by solving a series of relaxed problems.

```
initialize with a relaxation of the original problem, i. e., by omitting certain
  constraints;
repeat
    solve current relaxation;
    if relaxation is infeasible then
        return infeasible;
    else
        obtain optimal solution of relaxed problem $x^*$;
        determine one (or more) constraints separating $x^*$ from the original
          problem's feasible set and add them to current problem;
    end
until $x^*$ is valid for original problem;
return $x^*$ as optimal solution of the original problem;
```

integral vector $\boldsymbol{y}$ *is present in the definition of the optimization problem, we call it a linear program (LP).*

A straightforward approach to solving an OCP is to apply a discretization method, e. g., direct collocation, leading to an NLP. In case that a piecewise constant discretization method is used and the objective function is also linear, a linear program is obtained. In both cases, adding further integer variables and constraints on these variables results in a mixed integer program.

While enormous progress has been made in the field of mixed integer linear programming, cf. [43, 111], it remains challenging to bring together concepts from linear integer programming and nonlinear optimization. For an overview of the topic and further references, we kindly refer the reader to, e. g., [7, 14, 67]. In the remainder of this section, we will give an overview of solution methods for mixed integer programming.

2.2.1 Cutting Planes

The concept of *cutting plane* algorithms relies on the premise that the optimal solution of an optimization problem lies on the boundary of the feasible set. While this is always the case for linear programs, in the nonlinear case this holds only if the feasible set and the objective function are both convex. Nevertheless, there are approaches to deal with nonlinearity, e. g., by constructing linear outer approximations, cf. [78]. A generic cutting plane algorithm starts with a relaxation of the original problem. This means that certain constraints on the feasible set – including demands on integrality of variables – are omitted. The feasible set is iteratively solved to optimality with respect to the objective function. The task is to determine constraints, which are valid for all points in the original feasible set but are violated by the optimal solution of the current relaxation. Afterwards, the constraints are added to the relaxation. This procedure, which is depicted in Algorithm 2.1, is performed until the feasible set in the neighborhood of the optimal solution is described well enough. The solving step itself is usually performed by a state-of-the-art MI(N)LP solver. Especially for feasible sets, which are described by many inequalities, the advantage of a cutting plane algorithm is that the feasible sets of the relaxed problems, which are iteratively solved, are of much lesser complexity than the original one. This hopefully results in lower solution times for solving a series of less complex optimization problems compared to the time it takes to solve a single, more complex one. Obviously, the overall runtime of the cutting plane algorithm strongly depends on the time for computing valid cutting planes. Figure 2.4 illustrates this concept for a linear program.

An important application for cutting plane methods are integer programs. Here, a continuous relaxation in the integral variables is solved iteratively in order to obtain a valid integral solution. Much effort has been spent to efficiently determine strong cuts that cut off as much as possible from the relaxed feasible region. According to [44], the problem to determine a constraint which is valid for all integral points in the feasible region of the original problem but violated by a given point in the relaxed set is called *separation problem*. For an LP, a rather simple approach to derive such inequalities is as follows. Given a valid inequality

$$\sum_{j=1}^{n_y} a_j y_j \leq b,$$

where $y_j \geq 0$ are integer variables, we introduce a so-called *Gomory-Chvátal cut*

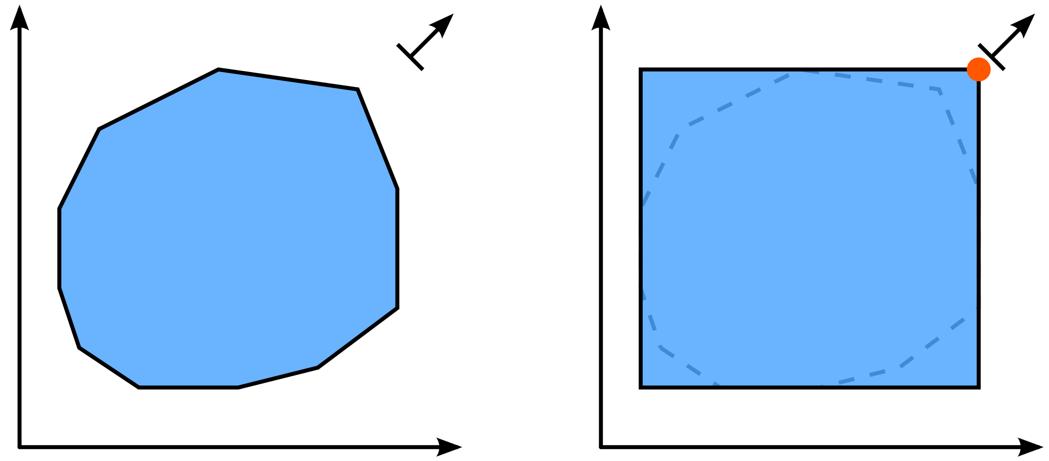
$$\sum_{j=1}^{n_y} \lfloor a_j \rfloor y_j \leq \lfloor b \rfloor.$$

This concept goes back to the works of Gomory [40] and Chvátal [15] and can be extended to include continuous variables as well.

A further example for cutting plane methods are *disjunctive cuts*, which were first described by Balas in [5] and are commonly used in state-of-the-art MILP-solvers. Disjunctive cuts are based on a disjunction in the feasible set imposed by integer variables. In case of a single disjunction, they are also referred to as *split cuts*, cf. [16]. Basically, a split cut for a single integer variable is a valid inequality for all points in the relaxations (of this variable) of the two disjunctive sets. It relies on the fact that the convex hull of these disjunctive relaxations includes all integer points of the feasible set. Of course, one is interested in finding strong cuts for the convex hull. This concept can also be extended to any kind of disjunction in feasible sets. For recent developments concerning disjunctive cuts and further references, we kindly refer the reader to [24]. Information on *perspective cuts* which rely on replacing an original convex function in the considered MINLP with its so-called *perspective function* can be found in [27, 39, 51]. There are still further kinds of cuts, some of which are implemented in state-of-the-art (MI)LP-solvers.

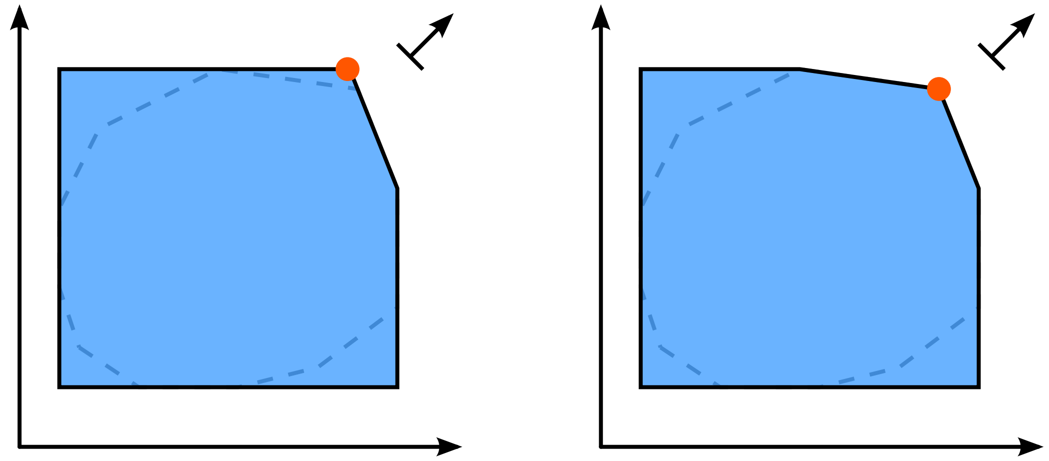
2.2.2 Column Generation

Another important approach for handling large-scale optimization problems is *column generation*. As it might be computationally demanding to solve the considered optimization problem as a whole and many variables are equal to



(a) Feasible set of the original problem and objective function.

(b) Feasible set of a relaxation of the original problem. The optimal solution with respect to the objective function is determined.



(c) Current optimal solution is not valid for original problem. Thus, a separating constraint is added to the relaxation. An optimal solution for the resulting relaxation is determined.

(d) Current optimal solution is again not valid for original problem. Another separating constraint is added to the relaxation. The optimal solution for the resulting relaxation is determined. As no constraints of the original problem are violated, the solution is also optimal for the original problem.

Figure 2.4: Visualization of the cutting plane method for a two-dimensional linear program. Solutions of a a series of relaxations of the original problem are calculated until no constraints of the original problem are violated by the current solution.

zero in the optimal solution anyway, one starts with a small subset of variables of the original problem. This results in the *restricted master problem (RMP)*. Further variables are iteratively added to the RMP. One possibility to identify the variables which are to be added is to solve the so-called *pricing problem*. It is basically a separation problem for the dual RMP, which identifies the most promising variables to be included in the next iteration by calculating maximal reduced costs (in case that the objective function is to be maximized). More specifically, the variables with the highest, strictly positive reduced costs enter the primal RMP. If no solution with strictly positive reduced costs exists, the current solution of the primal RMP is also optimal for the original problem. For detailed information on column generation, cf. [64, 65].

2.2.3 Branch-and-Bound

The *branch-and-bound* method originates for the MILP-case in [17, 60], but has later been extended to the nonlinear case, e. g., in [12, 70, 71]. A comprehensive overview of the branch-and-bound framework and different applications in the linear and nonlinear case can be found in [62]. We explain the algorithm for MI(N)LPs and the special case of the integer vector $\boldsymbol{y}$ being restricted to binary values, i. e., $\boldsymbol{y} \in \{0, 1\}^{n_y}$. Basically, a series of problems is solved following a tree structure. The root node consists of the original problem with all binary variables relaxed. Every node of the tree represents an LP or NLP. In each node of the tree, more binary variables are fixed than in its parent node. Thus, a valid upper bound on the objective value for all nodes in the whole subtree is given in each node by the objective value of the optimal solution. Finally, the leave nodes of the tree provide an enumeration in the integer variables. Certainly, one does not want to solve all possible optimization problems of the tree. To this end, the upper bounds in each node are used to cut off whole subtrees.

Considering a binary problem, the relaxed problem with $\boldsymbol{y} \in [0, 1]^{n_y}$ is solved first. Then, it is decided which variable is to be fixed to either bound, e. g., y_i. This step is called the *branching step*. The two resulting subproblems with $y_i = 0$ and $y_i = 1$, respectively, are added to the list of active problems. This step is repeated until the list of active problems is empty. Each node provides an upper bound on the objective value of all nodes in its subtree via its own objective value. Updating this bound after the optimization problem in a node has been solved is called the *bounding step*. Additionally, the best known objective value of an integer solution defines a global lower bound on the original problem's objective. The process of introducing two child nodes for a node is only skipped if:

- The problem of the current node is infeasible. As all subproblems will also be infeasible, the whole subtree can be pruned.
- The solution of the current node is integral in $\boldsymbol{y}$. In this case, the global lower bound can be compared with the objective and possibly provide an update.
- The objective value of the current node is lower than the current global lower bound. As it is a valid upper bound on the objective values in the subtree, the subtree can be pruned.

The method's main principle is illustrated in Figure 2.5. In fact, most MILP and MINLP solvers contain a branch-and-bound framework. However, the

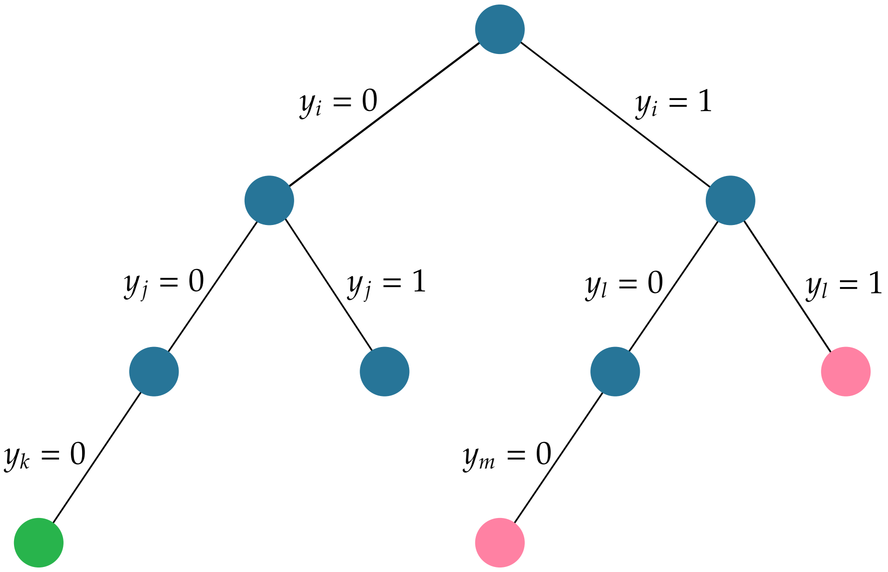


Figure 2.5: Visualization of a branch-and-bound tree. Each node depicts a single (N)LP. The solution of the blue nodes is fractional in at least one integral variable. The red nodes are pruned due to infeasibility or an objective value which is lower than the global lower bound. The green node indicates a valid solution for the (N)LP which provides a global lower bound. The lines between the nodes mark the branching step for a certain variable.

performance strongly depends on the actual implementation of the algorithm. We briefly outline different concepts and choices that have to be made. An important issue is the strategy of deciding in which order the nodes are to be processed. General methods are *best-first-search* and *depth-first-search*. While in the former in each iteration the node providing the currently highest upper bound is solved, in the latter the aim is to find a feasible solution as quickly as possible. Best-first-search often results in large trees, as many fractional solutions arise. A frequently used strategy combines both methods and is called *diving-method*. Here, depth-first-search is applied until an integral solution is found. Afterwards, the best open subproblem is determined and a new depth-first-search is started on this subproblem.

Furthermore, it is important to have good strategies for choosing the particular variable which is to be branched on in the current step. Usually, multiple different variables are fractional in the current relaxed solution. One possibility is to select the particular variable for branching with the maximum distance from an integral value. In contrast to this so-called *maximum fractional branching*, another strategy is the *strong branching*. Here, it is the idea to test which of the fractional candidates provides the best bound before actually branching on any of them, cf. [1]. A comprehensive overview of different branching-rules and further discussions on this topic can be found in [61, 74].

Finally, much effort is usually made to determine the upper bound in the different nodes heuristically. The purpose is to reduce the size of the tree by pruning nodes, which lead to either infeasible or non-optimal solutions, as early as possible. Insight in the special structure of the problem under investigation is often crucial here in order to obtain upper bounds which are as low as possible.

Cutting planes, cf. Section 2.2.1, can easily be incorporated in a branch-and-bound method leading to a *branch-and-cut* algorithm. To this end, in each node, the relaxed solution can be cut off by a cutting plane approach until a certain threshold is reached, before the branching step on the remaining problem is applied. In fact many state-of-the-art solvers implement a branch-and-cut mechanism. Additionally, also column generation methods, cf. 2.2.2, can be applied to a branch-and-bound algorithm, either as *branch-and-price* method or together with a cutting plane approach leading to a *branch-cut-and-price* method. Analogously to the branch-and-cut method, a column generation algorithm can be performed in each node before branching.

2.2.4 Modeling Logical Implications

In many practical problem settings, situations arise where constraints can be either enabled or disabled depending on certain logical conditions. Thus, we discuss possibilities to model optimization problems dealing with logical implications. First, we define the problem structure for the purpose of our considerations as:

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{z}} \quad & f(\boldsymbol{x},\boldsymbol{z}) && && (2.12)\\ \text{subject to:} \quad & \boldsymbol{g}(\boldsymbol{x},\boldsymbol{z}) \leq 0, \\ & [z_k = 1] \Rightarrow [\boldsymbol{x} \in \boldsymbol{S}_k] && \forall k \in \{1,\dots,n_z\}, && (2.13)\\ & \boldsymbol{x} \in \mathbb{R}^{n_x}, \\ & \boldsymbol{z} \in \{0,1\}^{n_z}, \end{aligned}$$

with

$$\boldsymbol{S}_k = \{\boldsymbol{x} \in \mathbb{R}^{n_x} \mid h_k(\boldsymbol{x}) \leq 0\}$$

In addition to the objective function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$ and constraints expressed by $\boldsymbol{g} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \to \mathbb{R}^{n_g}$, there are n_k logical constraints. Each of them involves a binary *indicator* variable z_k. The value this variable attains indicates whether $\boldsymbol{x}$ has to belong to the set $\boldsymbol{S}_k$ or not. In other words, if z_k is equal to 1 then the constraints $h_k(\boldsymbol{x}) \leq 0$ defining set $\boldsymbol{S}_k$ are active, cf. [11].

There are many examples of problem classes containing such logical implications. For example in flow shop problems, which belong to the class of scheduling problems, statements such as "if job A is scheduled before B, then condition C holds" arise. In the context of the discussions in this thesis, another example would be: "if car A is on the intersection area in time step t, then car B is not allowed to be on the intersection area in time step t.".

A well-known approach for expressing logical implications is the *big-M* method. Let the binary variable z_k indicate whether the constraint $h_k \leq 0$ is active or not. Then we can introduce the constraint

$$h_k(\boldsymbol{x}) \leq M_k(1 - z_k), \tag{2.14}$$

where M_k denotes a large positive constant. One can easily see that if $z_k = 1$ holds, the constraint $h_k(\boldsymbol{x}) \leq 0$ is imposed, whereas, if $z_k = 0$ is valid, the constraint (2.14) becomes redundant as $h_k(\boldsymbol{x}) \leq M_k$ holds for all $\boldsymbol{x}$. This argument is valid under the premise that $M_k \geq \sup_{\boldsymbol{x} \in \mathcal{F}}$, where $\mathcal{F}$ is the feasible set for $\boldsymbol{x}$. A major difficulty of the big-M method is that the values M_k might be hard to be determined. In case that $\mathcal{F}$ itself is unbounded, it is even impossible to calculate a valid big-M. Even if an M_k exists that fulfills the requirements, one should bear in mind that numerical issues might appear when choosing a big-M which is close to certain bounds used by the solver software, e. g., feasibility bounds or integrality bounds. Also, the solving process itself strongly depends on the particular choice of the value for M_k. More precisely, in the continuous relaxation of (2.14), which is used by current MI(N)LP solvers, a value of z_k close to 0 leads to a deactivation of the constraint $h_k(\boldsymbol{x}) \leq 0$. If the value of M_k is chosen to be very large, the relaxation gets weaker. On the other hand, big-M formulations are rather straightforward to implement, do not increase the problem size, and do not destroy linear, or convex properties of the original problem. Under the premise that we are able to determine rather tight values for M_k, big-M formulations can be the method of choice.

Another elaborate possibility to model logical implications in optimization is *disjunctive programming* and goes back to the work of Balas [5]. A generalization is the so-called *generalized disjunctive programming (GDP)* paradigm. The GDP formulation involves boolean and continuous variables that are specified in algebraic constraints, disjunctions, and logic propositions. For a detailed overview and illustrative examples of GDP, we kindly refer the reader to [41, 42, 84]. An application of GDP for mixed integer optimal control problems can be found in [51]. However, in this thesis, we will make use of big-M formulations as they are suitable for our purposes and implemented straightforwardly.

3 Developing a Cruise Control System

A major goal of this thesis is to establish, implement, and evaluate a novel driver-assistance system. Parts of this section concerning the concept of the system, implementation of the acceleration controller, discussions on technical requirements, the included HMI, and a brief report on test drives have already been published in [96]. The aim of this assistance system is to cross a traffic-light controlled intersection autonomously in an urban area. The developed application controls a vehicle's acceleration while respecting right of way regulations given by the traffic-light in the first place. A crucial part is the wireless exchange of information with the traffic-light. Thus, the application is considered to be cooperative while regarding each car individually.

In this section, we give an insight in the system's functionality. As the implementation of the assistance system is not only realized in a traffic-simulation software but also in a car, we will have a more detailed insight in technical requirements and challenges for an implementation in a car. Impact on traffic flow, which is measured by using a microscopic traffic simulation software, is evaluated later on in Section 6.4. We highlight an additional application providing a visual overview of information of the traffic-light and other traffic participants in the area of a traffic-light controlled intersection.

3.1 Overview

Traffic-light controlled intersections are bottlenecks for urban traffic flow. Unfortunately, there are limited possibilities for increasing a traffic-light's *capacity*, or, in other words, increasing the amount of vehicles which can pass over a certain time horizon. One could think of extending the junction area physically, e. g., by building more lanes.

Another strategy is the improvement of a traffic-light's logical rules, i. e., in which particular way it allows vehicles, which are driving on the different lanes, to pass the junction area. In fact, a lot of research exists that tries to cope with this problem of improving the distribution of red and green phases. There are algorithms, which treat the problem as a multi-agent problem, that take the traffic-light and oncoming cars into account. Frequently, the traffic-lights are called *self-organizing* in this case, cf. [35, 75]. In [49] and [98] fuzzy logic is applied to improve the performance of an isolated traffic-light. Multiple research approaches are based on evolutionary or genetic algorithms, e. g., [90, 105]. Finally, there are publications which make use of mathematical programming to achieve an increase in traffic flow at intersections by controlling the behavior of traffic-lights, cf. [94, 95].

A third possible way to improve traffic flow at intersections is to adapt the individual behavior of each vehicle (or a certain amount of vehicles). In fact, a high potential lies in controlling not only the moment when a car enters an intersection (or more precisely, when it passes the stopping line) but also the way the car accelerates and decelerates when approaching it. The time for passing an intersection for a car is minimal if the stopping line is passed with maximum velocity at the time the light switches to green. Note that for simplicity, we only distinguish between the signal states *green* and *red*. The former one also includes the amber phase as cars are still allowed to pass the intersection. The latter one additionally contains the red-amber phase. In contrast to the work in this thesis, there are approaches which only aim to minimize the time gap between the switch from red to green and the transit of the intersection. For example, in [23] and [52], systems for a green-light-optimized speed-advisory (GLOSA) are presented. The authors introduce algorithms whose outcome is a static velocity which is suitable for passing the intersection without having to stop because of a red light. Apart from the fact that only static velocities are calculated, these methods do not consider preceding vehicles. Moreover, we do not only develop a system which provides the driver with information about a proposed velocity, as in [30]. We additionally develop an acceleration controller which autonomously performs the calculated accelerations. Another related algorithm is presented in [3], called predictive cruise control. Here, methods of model predictive control are used to calculate trajectories for single and multiple cars in order to pass a traffic-light controlled intersection automatically. Although it is included in the objective to drive near the driver's preferred velocity, the presented results reveal that the cars drive with less than the preferred velocity when passing the intersection. As explained above, a highest possible velocity is crucial for a better outcome in terms of minimizing the time it takes to pass the intersection area. Also, an analysis of solutions obtained in Section 4, which mirror an overall optimized traffic flow, supports this statement. Finally, the authors in [79] introduce an algorithm, including an optimal control problem, in order to determine speed-trajectories for a car which is approaching a traffic-light while the light's signal-states are fixed. In this publication and the provided simulations therein, only a single car is considered, although the authors briefly propose an extension of the algorithm to multi-vehicle scenarios.

3.2 Concept of an Assistance System

In the BMWi-funded project UR:BAN, cf. [58], partners from industry, research institutes, and municipal authorities developed driver-assistance systems as well as traffic management systems for traffic in an urban setting. The project involved systems for enhancing perception of situations that occur in traffic and research fields which take the interaction between a human user and a machine into account. Besides these and many more areas, a central research topic was the investigation of wireless communication technologies and their applications in traffic-related systems.

As explained above, situations concerning an intersection are of particular interest. When passing an intersection, many different traffic participants have to be taken into account. This is why encouraging communication and cooperation between those appears to unlock some potential and hence was studied in the UR:BAN-project. It seems easier for an artificial system to recognize an intersection area, for example visually, when a traffic-light is located in front of the intersection. Additionally, the occurring regulations of right of way do not directly depend on other traffic participants since the traffic-light, as a central regulation unit, governs which cars are allowed to pass. These relatively simple rules are easier to implement in a system than those rules that depend on the presence of another car in a certain position. Finally, another advantage of focusing on traffic-light intersections arises: there is already some computational infrastructure available, which can be used for implementing algorithms and communication devices. All the aforementioned arguments are reasons why we develop a driver-assistance system which controls a car's acceleration and deceleration when approaching a traffic-light regulated intersection.

The basic concept of the system is as follows: when approaching a traffic-light, a communication between the car and the traffic-light is established. In general, different types of wireless communication are possible here. We make use of the C2X-technology, which is practically a wireless LAN using a communication protocol that can exclusively be used for automotive applications. The relevant communication technology will be looked at more closely in Section 3.4.1. Once the communication has been established, the traffic-light provides the car with some information. In particular, it shares its position and the geometry of the intersection area including number and location of lanes and stopping lines using the *global positioning system (GPS)*. The traffic-light also submits data about its current light statuses for each lane as well as a corresponding prediction. More specifically, it provides information for each lane about the time to the next change of its light status and the time to the second-next change. For more information about the traffic-light and its behavior in communication, cf. Section 3.4.2. The two types of location data (location of intersection and geometry of intersection area) can now be used by the assistance system in the car to locate itself relatively to the traffic-light. This requires the car to have a positioning system itself, cf. Section 3.4.3. We now know how far the car is away from the traffic-light, or, more precisely from the stopping line, and which lane it is driving on. The latter information is used for retrieving the current status of the traffic-light. Note that there is no visual recognition at all. However, we could think of combining the identification of the traffic-light's current status via wireless communication and visual recognition. In this case, the visual system could serve as security fallback if the wireless communication was interrupted or defective.

Up to this point, we do not benefit from the communication between the traffic-light and the car. Data about positions and traffic-light status could also be retrieved via up-to-date maps and visual recognition, respectively. The major advantage of a direct exchange of information with the traffic-light is the provided prediction of the signal status. We can make use of this information and determine a certain strategy, which we plan to perform during a fixed time horizon (called *planning horizon*). We will refer to such a strategy as *regime*. A regime could for example be the intention to stop in front of the traffic-light or to pass it without any adaption of speed. We will have a look at all the regimes later. The point is that only the prediction of the traffic-light's upcoming signal changes makes us capable of planning ahead for several seconds. After gathering all the information mentioned above and determining a regime, we calculate trajectories for the car's acceleration in the planning horizon. Finally, we pass the calculated acceleration for the next time step to the engine via a control unit, which is called *Autobox*.

Remember that our ultimate goal is to improve traffic flow and to increase the number of cars which pass the traffic-light during a green phase. To this end, we minimize the time gap between the traffic-light's switch from red to green and the automated crossing of the stopping line. Additionally, we want the car to be as close as possible to its maximum velocity on the stopping line. Both concepts lead to a minimization of the time it takes the car to pass the intersection (also called *passing time*). It is easy to understand that an individual minimization of the passing time leads to a maximization of the number of vehicles which can pass the intersection during a fixed green phase. The idea is that also non-equipped cars benefit from the system, as one of their preceding cars needs less time to pass the intersection. For a detailed evaluation of the effects on traffic flow, cf. Section 6.

The developed system is technically embedded in the car's *adaptive cruise-control (ACC)*. The ACC is a state-of-the-art technology extending the car's cruise control, that allows the driver to set a preferred velocity which the car performs automatically. The ACC integrates data of preceding vehicles or obstacles, which are recognized via radar sensors or laser sensors. Afterwards, the system adapts the car's velocity so that a defined time gap or distance to the preceding car or obstacle is satisfied. For further information, cf. [87]. This setup allows us to retrieve the driver's preferred velocity and provides technical infrastructure and fallbacks. We will discuss these issues shortly.

Before the calculation of a desired acceleration starts, a central module, called *regime-controller*, exclusively enables a single regime based on the gathered information about the environment and provides the necessary data. This information has to be available in the car as soon as a fixed distance to the traffic-light is undershot. The so-called *starting-distance* is currently set to 200 m, and we consecutively demand the communication range to be at least 200 m. Due to the regime-based design of the assistance system, we will from now on refer to it as *regime-ACC (RACC)*. The authors of [38] present a similar approach for traffic on a highway based on different maneuvers.

Free-Transit-Regime In case that the regime-controller asserts that the traffic-light's stopping-line will be reached during a green phase if the car continues at its current velocity, the free-transit regime is enabled. Here, the system simply performs the driver's preferred velocity.

Deferred-Transit-Regime In case of a predicted arrival at the stopping line during a red phase, the regime-controller activates the deferred-transit-regime. Core of this regime is an MPC-controller, cf. Section 2.1.2, which calculates trajectories on the planning horizon, cf. Section 3.3. Basically, we want the car to pass the stopping line with the driver's preferred velocity as soon as the light changes to green. More specifically, we aim for the minimum of the preferred velocity and 40 km/h. That is because of the legal regulations stated in [26], where regulations concerning traffic-lights are arranged. Additionally, it is possible to incorporate a certain security threshold between the changing time from red (in Germany red-yellow) to green and the automated passage. During the deferred-transit, a defined minimal velocity ($v^{min} > 0$) is not undershot. This avoids a disturbance of succeeding cars, and also a stop of our vehicle. In this regime the concept of minimizing the passing time, as explained above, is fulfilled. The car's velocity when passing the stopping line is maximized, and the time gap between the switch from red to green and the passage is minimized.

Pole-Stop-Regime If again the regime-controller predicts an arrival during a red phase and additionally the velocity would drop below v^{min}, the pole-stop-regime is enabled. The word *pole* indicates that this regime is only active if there is no preceding vehicle detected. Otherwise, the car-following-regime is activated. During the pole-stop-regime, the car performs a stopping maneuver in front of the traffic-light's stopping-line, while respecting a defined distance of several meters, e. g., 10 m. This so-called *stopping gap* can be parametrized statically, or could also be dependent on the geometry of the intersection the car is approaching. This adaption allows us to avoid blocking other lanes, which can be rather short. The information about a junction's geometry is submitted by the traffic-light, cf. Section 3.4.1; it is, however, currently not evaluated for adapting the stopping gap.

Pole-Start-Regime After having performed a pole-stop, the car starts automatically during the pole-start-regime. To this end, trajectories are calculated, again by model predictive control, which lead to an acceleration of the car and a transit of the stopping line after the traffic-light switched to green. As the car has stopped several meters in front of the stopping line, it can be accelerated already during the red phase. Remember that we have a prediction of the changing time to green. This leads to a higher velocity the car passes the stopping line with, in contrast to a stop directly in front of the stopping line. This effect obviously grows with a growing stopping gap until the minimum of the driver's preferred velocity and the legal passing velocity of 40 km/h can be reached. As discussed in Section 3.4.6, we have to find a tradeoff between an augmentation of the vehicle's passing velocity and the acceptance of the system by other traffic participants. Again, in this regime, we reduce the vehicle's passing time.

Car-Following-Regime If a preceding car is detected, and because of its current position, velocity, and acceleration it is predicted that the distance between our car and the preceding one would fall below a determined security distance while performing one of the regimes above, the car-following-regime is enabled. During this regime, the system follows the preceding car using the ACC-system.

Changes in the traffic situation, e. g., another car is switching to our lane in front of us, or a trajectory which is not performed well enough, will lead to the necessity of recalculating the trajectory. Additionally, the provided times of changes are fraught with inaccuracies. That is why we design our assistance system in a way that it allows us to do both plan ahead for a certain time horizon and be dynamic enough to deal with inaccuracies and changes of the traffic situation. To achieve this, our assistance system is based on a cyclic method. In fact, we repeatedly gain information about the traffic situation, including the traffic-light's statuses and predictions. Additionally, we get information about preceding cars by measurement via built-in radar sensors, cf. Section 3.4.3. We also incorporate data about our own car (current position, velocity and acceleration) to get all the data we need to determine a regime which we exclusively want to perform from now on. As mentioned above, the regime-controller decides which regime we are about to perform. In the next step of the algorithm, the controller of the exclusively activated regime calculates trajectories on the planning horizon, cf. Section 3.3. The determined acceleration is passed to the engine afterwards. This cyclic method is performed with a frequency of 10 Hz and visualized schematically in Figure 3.1.

Security Fallbacks Remember that we develop a pure assistance system. This means, the driver is in charge of driving the car and of supervising its behavior. Nevertheless, we incorporate some security fallbacks. First of all, the system is technically an extension of the car's ACC. So applying the brake pedal deactivates the ACC and simultaneously also our assistance system.

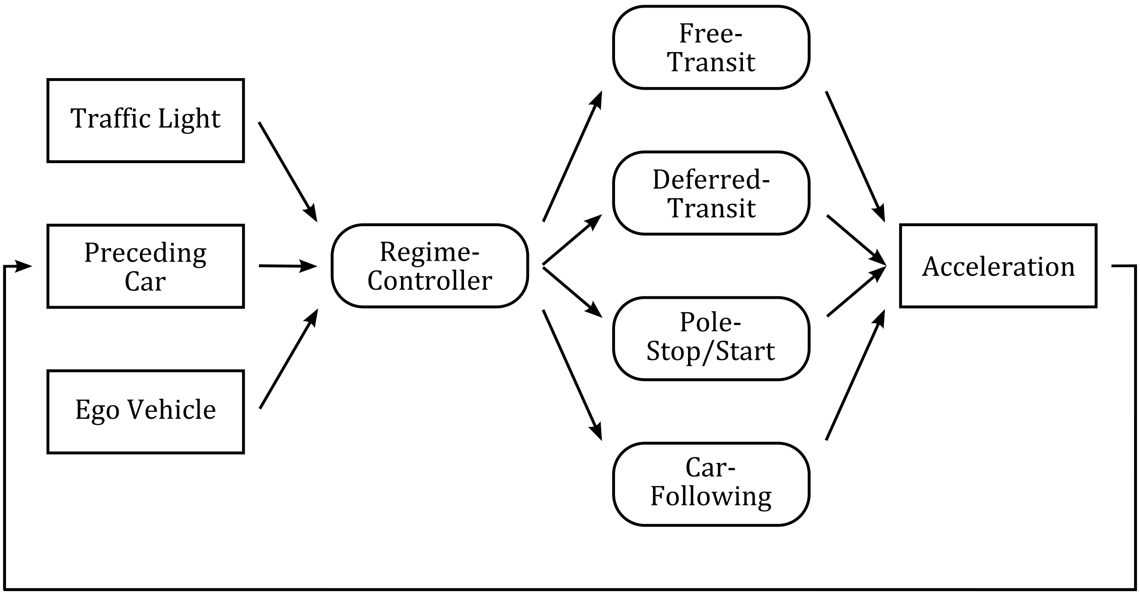


Figure 3.1: Schematic visualization of the cyclic process the assistance system is based on. The main steps are: gathering environmental information, determining a regime exclusively, calculating trajectories in a regime, and passing the calculated acceleration to the engine. This process is performed with a frequency of 10 Hz.

We also compare the acceleration a regime wants to perform with the desired acceleration of the ACC in every time step. The minimum of both values is sent to the engine afterwards. This way, we want to avoid collisions with preceding cars, for example if we mistakenly did not choose the car-following-regime despite a small gap between our car and a preceding car.

A few seconds before the pole-start-regime is activated, the driver has to confirm that it is safe to accelerate the car and to enter the intersection. This is necessary, as pedestrians or cyclists can still be on the junction area, while the radar sensor is not able to detect them.

Furthermore, the assistance system is only activated on lanes which allow either a straight transit, or on turning lanes where no right of way rules concerning oncoming traffic are present.

3.3 Controller Implementation

We now discuss the implementation of those regimes more specifically which include a calculation of trajectories for the whole planning horizon. Note that in the free-transit, car-following, and pole-stop-regime technically the car's ACC is used. Nothing special has to be implemented for the first two regimes. We simply activate the vehicle's ACC. For the latter one we introduce an artificial object, which is standing in front of the stopping line and manipulate the ACC, with the result of stopping the car within the distance of the stopping gap.

During the deferred-transit and pole-start-regime, an acceleration trajectory on the planning horizon $\mathcal{T} := [t_0, t_f]$ is calculated. First, we introduce the following ODE which models the longitudinal motion of the car:

Table 3.1: Physical quantities used for modeling resistances

Symbol	Unit	Value	Description
C_w	-	0.32	drag coefficient
S	m^2	2	size of the car's frontal area
ρ	kg/m^3	1.2	density of air
m	kg	1500	car's mass
μ	-	0.015	rolling resistance coefficient
g	m/s^2	9.81	gravitational acceleration

$$\begin{aligned} \dot{s}(t) &= v(t) && \forall t \in \mathcal{T}, && (3.1)\\ \dot{v}(t) &= a(t) - \frac{C_w S \rho}{2m} v(t)^2 - \mu g && \forall t \in \mathcal{T}, && (3.2)\\ \dot{a}(t) &= j(t) && \forall t \in \mathcal{T}. && (3.3) \end{aligned}$$

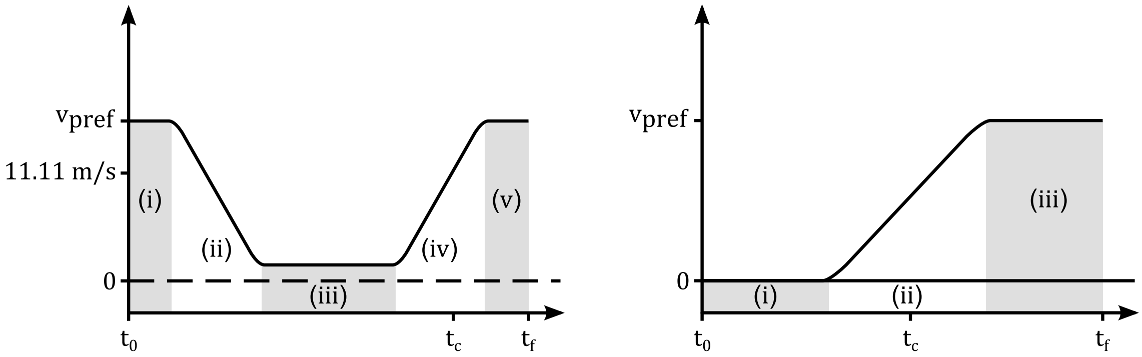
Here, the function $v(t)$ reflects the car's velocity, $a(t)$ its acceleration, and $j(t)$ its jerk. Additionally, there are some physical quantities in Equation (3.2) modeling air resistance and rolling resistance according to [55], cf. Table 3.1. We set up an OCP, cf. Section 2.1, of the form:

$$\min_{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)} \quad \int_{t_0}^{t_f} \|\boldsymbol{x}(t) - \bar{\boldsymbol{x}}(t)\|_{\boldsymbol{W}}^2 + \|\boldsymbol{u}(t) - \bar{\boldsymbol{u}}(t)\|_{\boldsymbol{Q}}^2 \, \mathrm{d}t \tag{3.4}$$

$$\begin{aligned} \text{subject to:} \quad \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) && \forall t \in \mathcal{T}, && (3.5)\\ 0 &\le \boldsymbol{p}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) && \forall t \in \mathcal{T}, && (3.6)\\ 0 &\le \boldsymbol{r}(\boldsymbol{x}(t_k)) && \{t_k\}_{k \in I} \subseteq \mathcal{T}. && (3.7) \end{aligned}$$

The objective (3.4) minimizes a least-squares term on the planning horizon. Quadratic matrices $\boldsymbol{W} \in \mathbb{R}^{n_x \times n_x}$ and $\boldsymbol{Q} \in \mathbb{R}^{n_u \times n_u}$ are introduced for weighting the deviation between the desired values $\bar{\boldsymbol{x}}(t)$ and $\bar{\boldsymbol{u}}(t)$ and values of $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ in the solution, cf. Section 2.1.2. The motion according to Equations (3.1)–(3.3) is included in (3.5), where the jerk $j(t)$ is the system's control unit. We also introduce path constraints $\boldsymbol{p}(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$ in (3.6). These are used to model bounds on the states and controls for physical and comfort reasons. Point constraints (3.7) provide the possibility to restrict values of the states on discrete time points t_k with discrete index set I. Typical constraints of both types are:

$$\begin{aligned} v^{min} \le v(t) &\le v^{max} && \forall t \in \mathcal{T}, && (3.8)\\ a^{min} \le a(t) &\le a^{max} && \forall t \in \mathcal{T}, && (3.9)\\ j^{min} \le j(t) &\le j^{max} && \forall t \in \mathcal{T}, && (3.10)\\ s(0) &= 0, && && (3.11)\\ v(t_c) &\le min(v_{pref}, 11.11). && && (3.12) \end{aligned}$$



(a) The velocities during the five phases of the deferred-transit-regime's trajectory.

(b) The velocities during the three phases of the pole-start-regime's trajectory.

Figure 3.2: Schematic illustration of the different phases during the trajectory-based regimes. The diagrams show time on the horizontal axis and velocity on the vertical axis.

The value of v^{max} is the present speed limit (usually 13.89 m/s in German cities). If the changing time to green t_c is within the planning horizon, the Constraint (3.12) is added to the system. As explained above, the velocity is then limited to the minimum of the driver's preferred velocity v_{pref} and the legal speed limit of 11.11 m/s. The two regimes deferred-transit and pole-start differ in the desired values $\bar{\boldsymbol{x}}(t)$ and $\bar{\boldsymbol{u}}(t)$, as well as in the values of the weighting matrices $\boldsymbol{W}$ and $\boldsymbol{Q}$. Both of them can be further adapted in order to realize different profiles, e. g., economic or rather dynamic trajectories. We also make use of them to split the resulting trajectory for the deferred-transit roughly into five phases:

(i) maintain-velocity-phase,

(ii) deceleration-phase,

(iii) constant-slow-phase,

(iv) acceleration-phase,

(v) end-velocity-phase.

After maintaining the velocity for a certain amount of time, we decelerate the car until a slower velocity is reached and constantly performed during the second phase. Subsequently, the vehicle is accelerated until the desired velocity for transit is reached and further accelerated if $v_{pref} > 11.11$ m/s. At the end of the trajectory, the car should maintain the driver's preferred velocity. The trajectory performed during the pole-start-regime can analogously be split into three phases:

(i) idleness-phase,

(ii) acceleration-phase,

(iii) end-velocity-phase.

Figure 3.2 schematically shows the different phases of the trajectories during the deferred-transit and pole-start-regime.

For the implementation of the presented assistance system, we use the ACADO-toolkit, cf. [50]. ACADO is an open source software for modeling, simulation, and control of dynamic processes. The included code-generation tool makes it possible to generate tailored algorithms that realize an MPC-scheme using a direct solution method, cf. Section 2.1. The generated code allows us to adapt the weighting matrices $\boldsymbol{W}$ and $\boldsymbol{Q}$ as well as the finitely many desired values $\bar{\boldsymbol{x}}(t_k)$ and $\bar{\boldsymbol{u}}(t_k)$ for $\{t_k\}_{k\in I_N}$ in order to define the objective function appropriately.

3.4 Technical Requirements and Implementation

To implement the RACC in a car, some adjustments and extensions in hardware and software of the car have to be made. Besides these extensions, we only use technology which is installed in current cars. Furthermore, the traffic-light has to be capable of communicating and running necessary calculations. We give an insight in what the challenges and practicable solutions for getting the application to work properly are.

3.4.1 C2X-Technology

As mentioned above, communication with the traffic-light is crucial for planning trajectories. There are different possibilities to do so. First, one could think of purely identifying the traffic-light's current state visually. Although this approach would hardly require any additional devices to be installed in the infrastructure, a resulting application in a car would only be able to react to the light's current state. Nevertheless, there are some works on transmitting information using high speed LED transmitters, cf. [106]. Despite the fact that such technology allows us to establish the necessary rather complex communication between the traffic-light and a car, it cannot compensate the drawback that we would need to guarantee an obstacle-free field of view. Especially in urban areas this would be hard to achieve.

This limitation of visual communication leads us to the idea of wireless communication via radio networks, which is also called Car-to-Car (C2C) and Car-to-X (C2X), respectively. In recent years there have been multiple research projects making use of C2C and C2X (also called: V2V and V2X). Some examples of developed applications are: an application for displaying information of traffic-lights and giving advice on the velocity to be performed by the driver, cf. [30], an application which provides information about construction areas along with related changes in speed limits and possibly closed lanes, cf. [104], or a system that informs about emergency vehicles, e. g., position and possible conflicts, mostly for security reasons, cf. [58]. All the aforementioned projects make use of communication based on wireless LAN. It comes with the advantages of being easily deployed, being based on a mature technology, providing low latency, and being capable of natively supporting C2C and C2X communication in ad hoc mode, cf. [2]. Regarding more technical aspects, there exist many different standards and specifications concerning WLAN. Among these, the suitable standard for vehicular applications is the IEEE 802.11p-standard, cf. [4]. In practice, different types of messages are exchanged periodically between the agents. To make use of a well tested and approved standardization, we choose the resulting specifications of the project simTD [104]. Four different types of messages are used for our purposes, which are all of XML-type:

- *Cooperative awareness message (CAM)*: provides information of the presence, the position, and the basic status of each agent.
- *Decentralized environmental notification message (DENM)*: provides information about specific driving environment events or traffic events to other agents.
- *Intersection*: provides topological information about an intersection via GPS-data, e. g., number and position of lanes and stopping lines.
- *Signal phase and timings (SPaT)*: provides the current status of a signalized intersection along with the expected time to change.

These messages provide necessary information for the RACC's functionality. Still, considering further systems, which may also invoke a bidirectional communication between the agents, the data provided by the car-related messages (CAM and DENM) might not suffice. We will have a more detailed look at this discussion in Section 5.2.1.

Concerning the concrete realization of the RACC, there are also other kinds of technology which can be suitable for exchanging data periodically over distances of several hundred meters with low latency. Another technology used for communication between agents is *long term evolution (LTE)*. It provides high data rate and can benefit from a large coverage area. The attainable delay satisfies most of the vehicular network application requirements, cf. [76], as long as a direct communication between traffic-light and car is established. Current technical solutions often realize a communication based on cellular network via a central back-end server. The resulting indirect communication is often fraud with high latency. Thus, we focus on 802.11p as it is perfectly suitable for our application and we are mainly interested in developing algorithms which can be easily adapted for different kinds of communication-techniques.

3.4.2 Cooperative Traffic-Light

Additional technical devices for the traffic-light have to be installed to make it capable of providing the necessary information. These are according to [28]:

- A *communication control unit (CCU)*, for wireless communication via C2X, cf. Section 3.4.1. It is additionally installed in the car.
- A *application unit (AU)* is needed to process applications requiring a higher computing capacity in the field.
- A GPS receiver has to be installed for both time and positioning reference.

Note that the calculated switching times, which are crucial for our assistance system, can be inaccurate. This is due to the fact that the algorithm, which runs on the AU, learns the switching times over a certain period of time and afterwards provides estimated times until the next and second-next change, respectively. These are usually accurate for traffic-lights with a fixed switching scheme. But there are numerous traffic-lights with dynamic switching schemes, which can be based on the current traffic situation or incorporate priorizations for public transport or police vehicles. In this case, the calculated switching times are fraught with inaccuracies. Nevertheless, even for dynamic switching schemes it is possible to calculate correct switching times about five seconds in advance. Once the signal status for a lane changed to amber, there is usually the same timing procedure (at least in Germany, cf. [26]) for the lanes, which are about to be set to green: the amber-phase should last three or five seconds. The subsequent phase, during which all lanes are set to red is fixed and depends on the geometry of the intersection area. Afterwards, the phase of red-amber normally lasts another second before this particular lane gets the green signal. Still, this information has to be broadcasted with low latency. As mentioned above, this can not always be guaranteed when using a communication technique via central back-end servers. In this case, algorithms might be applied which are predicting future changes for a longer time horizon in advance. Unfortunately, they are likely to be not very exact – especially for traffic-lights with dynamic switching schemes.

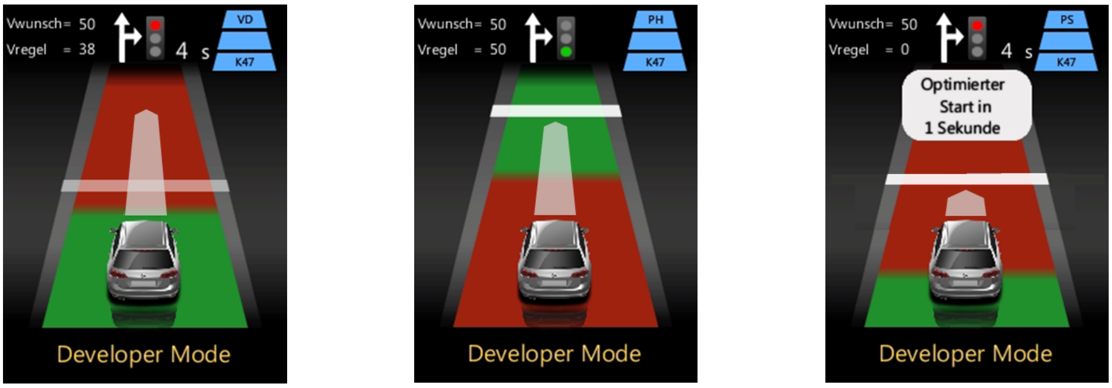
3.4.3 Adaptions for the Car

We will now focus on additional technical and software requirements, which have to be met in the car in order to run the RACC. All the computational devices presented in this chapter are connected via a *controller area network (CAN)*, cf. [63], which is established in the car.

3.4.3.1 Positioning System For the purposes of the application, it is necessary to guarantee a very fine position detection of the car on the road. In particular, we have to know which lane the car is driving on. Otherwise we could not be sure if the signal group we are matching ourself to is the correct one. As it is already included in many cars, we use a GPS-device for position detection. In [77] the authors demonstrate that error ranges in GPS positioning between 2 and 15 meters occur in urban areas. To achieve a correctness that is satisfying for our application, we make use of correction data based on mobile-communication technology via an additional modem. This makes us capable of identifying a car's position within an error range of less than a meter. In combination with a map-matching algorithm, which is not further explained in this thesis, we can determine the correct lane in most situations.

3.4.3.2 Communication Devices The wireless communication with the traffic-light via C2X is managed by the CCU. This is a computational device which handles outgoing and incoming C2X-messages and provides the data via CAN. Additionally, the car's antenna is physically upgraded to allow C2X and C2C-communication.

3.4.3.3 Radar Sensor When approaching a traffic-light and automatically passing the intersection, it is crucial to detect preceding vehicles. This can be achieved via a radar sensor, which is necessary for the car's ACC-system and thus already installed in the car. The sensor shares information about acceleration and velocity of a detected object as well as the current distance to it via CAN. This data is analyzed in every time step by the regime-controller. Special care has to be taken if an object is not moving. The sensor detects only those objects with a high reliability, which are in motion or have been in motion in the course of the tracking process. For our assistance system, it is crucial to detect not only moving vehicles, but also those which are standing in front of the traffic-light. Therefore, we enabled



(a) Currently red light, but planned arrival at green.

(b) Currently green light, but planned arrival at red.

(c) Currently red light, pole-start in 1 second. The switch to green will be in 4 seconds.

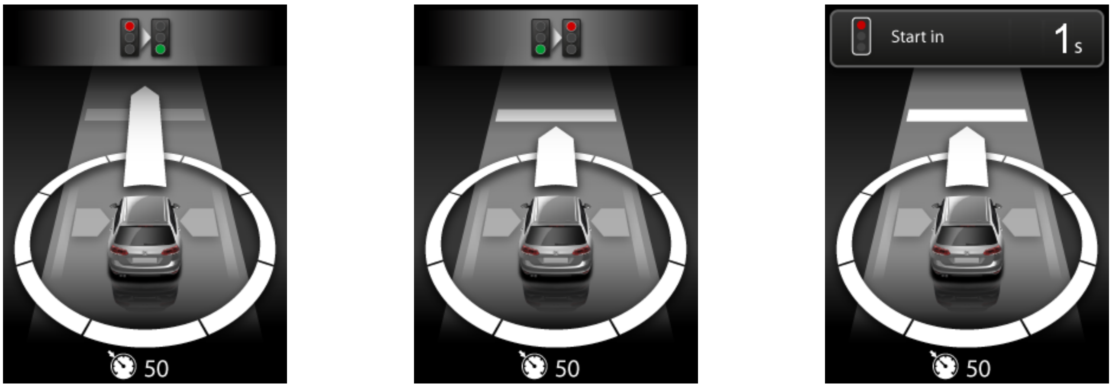
Figure 3.3: Developer HMI showing three different situations when approaching a traffic-light. The red and green carpet moving from the bottom to the top visualizes the car's position relatively to the signal phases.

the sensor's functionality to detect all objects, including non-moving ones. The resulting detection rate is not always satisfying. But as the main focus of this thesis is to implement the car's behavior, we do not discuss this issue any further.

3.4.3.4 Software Setting for Acceleration Controller Besides the processing units mentioned above, the RACC needs to be implemented in the car. Thus, another computational device, which runs a Windows operating system is installed and connected to the CAN. The assistance system's functionality, including the ACADO-based acceleration controller, is encoded in C and C++-files. These are included in the *Automotive Data and Time-Triggered Framework (ADTF)* , cf.[22]. Hence, the RACC is available in the car.

3.4.4 Human-Machine Interface

Besides a gain of efficiency in terms of increasing traffic flow, we also attempt to design the application as convenient as possible for the driver. Consequentially, one aspect is to inform the driver about the maneuver the car is about to perform. Additionally, before the performance of the pole-start-regime, the driver has to confirm that it is safe to accelerate the car and enter the intersection. For these purposes, a *Human-Machine Interface (HMI)*, shown in Figure 3.3, is developed, which provides information about the current signal state of the traffic-light, allowed directions of the lane the car is matched on, time in seconds until the next change to green, preferred velocity, and currently performed velocity. Additionally, we introduce a red and green carpet underneath the car moving from the bottom to the top. It visualizes the car's position with respect to the red and green signal phases of the traffic-light to come. Thus, it indicates if the car will arrive at signal state red or green, assuming that it keeps on driving with the current velocity. The position of the stopping line in the display mirrors its distance to the car. It is therefore moving from the top to the bottom. An arrow in front of the displayed car points at the stopping line if the stopping-regime is performed, or at a leading



(a) Currently red light, but planned arrival at green.

(b) Currently green light, but planned arrival at red.

(c) Currently red light, planned pole-start in 1 second.

Figure 3.4: User HMI showing three different situations when approaching a traffic-light. Information about the RACC is incorporated in an already existing HMI providing information about multiple applications.

car if the following-regime is active. The three blue bars in the top right corner indicate the quality of received C2X-messages and if a regime has already been chosen. In Figure 3.3c), the reader can see the information that the car is about to start after performing a pole-stop. At this central position also the prompt occurs which asks the driver to confirm that the situation allows a safe starting maneuver after a full stop. The quote *Developer Mode* indicates that this HMI is highly experimental.

In cooperation with professional HMI-developers, another kind of HMI is established. Figure 3.4 shows examples which correspond to the above ones in the developer HMI. Here, information about the RACC is incorporated in the existing HMI showing information about multiple systems and applications. Thus, in the resulting layout, some aspects from the developer HMI, i. e., the green and red carpet as well as information about the currently performed velocity, allowed directions, and signal quality are omitted. Instead of the carpet, two traffic-lights indicate the current signal state and the predicted state at scheduled time of arrival. Again, information about the time to green if a pole-stop has been performed is given, and the prompt, which asks for confirmation to start the car automatically, occurs.

3.4.5 Test Drives

Up to this point, we have had an insight in the functionality and technical requirements for the RACC. We now discuss setup and experiences which we gained during intense test drives on both a test field closed for public traffic and in real-world traffic.

Initial test drives for the system were performed on a test field. For this purpose, we set up a portable traffic-light including necessary computing and communication devices as well as fictional data about adjacent lanes and stopping lines. First, we focused on technical issues, such as range and latency in communication between traffic-light and car. Also the quality of the car's refined positioning system was part of the experiments. Without going into too much detail, we can

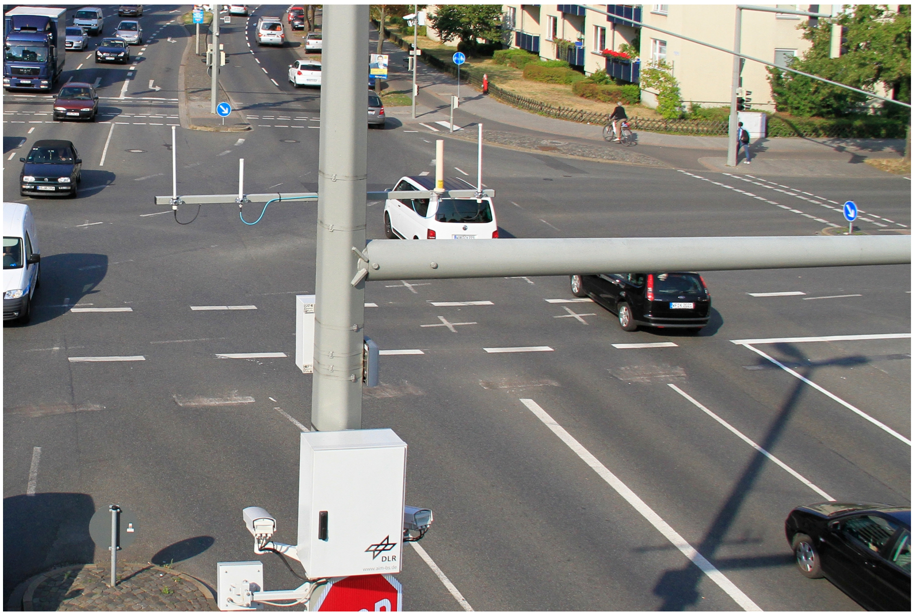


Figure 3.5: Some devices which are installed at multiple intersections in the city of Braunschweig. The image shows antennas for communication, processing units (gray boxes), and stereo cameras for monitoring traffic, cf. Section 6.3.

state that our requirements for these quantities could be met. Afterwards, the regime-based functionality was deeply examined: selecting the proper regime for a certain situation, calculating trajectories, using the car's ACC, switching between different regimes in certain situations, and passing the traffic-light's stopping-line shortly after switching to green with desired velocity.

Subsequently, we extended our experiments to real-world traffic. Part of the project UR:BAN was to equip traffic-lights in several cities with devices which allow to communicate and cooperate with them, cf. Section 3.1. Thus, the so-called *Anwendungsplattform Intelligente Mobilität (AIM)*, cf. [93], in the city of Braunschweig was involved in the project. This is a test field in real-world traffic which consists of multiple traffic-light controlled intersections. They all are equipped with computational infrastructure, including those mentioned in Section 3.4.2. In Figure 3.5, some of these devices at one of the AIM-intersections can be seen. Repeating the experiments of the test field, we achieved similar results. Only the communication range was slightly shorter in the urban setting (possibly due to structural disturbances or other present wireless transmitters), but with 300 meters still large enough. Also the RACC's performance was still satisfying.

Beyond these issues, it was part of the test drives to get an impression of the performance and practicability of the QP-solver qpDUNES, which is presented in [29]. As the OCPs occurring in the RACC are rather sparse, it seems promising to make use of a QP-solver exploiting this structure, which is qpDUNES. Besides the very low solving times compared to ACADO's standard QP-solver qpOASES, qpDUNES reacted very sensitive to unforeseen changes in the measured states leading to unreasonable solutions or even unsolvabilities. Thus, the final RACC-implementation is based on qpOASES.

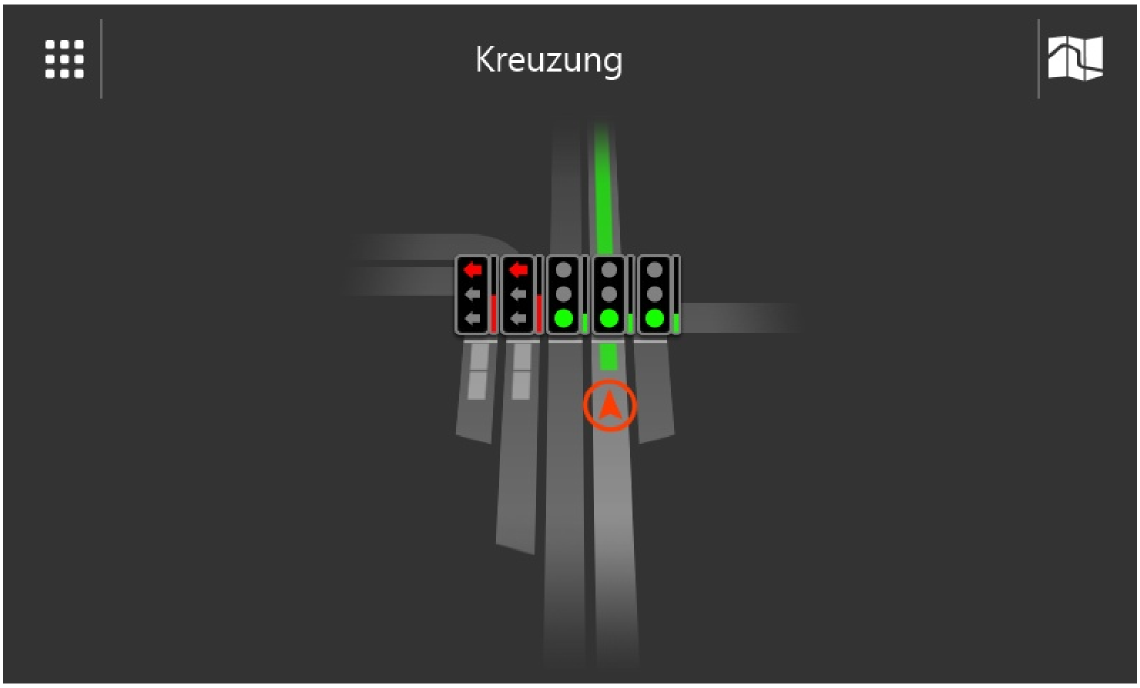


Figure 3.6: Example of the information the MIB-HMI shows, when the car approaches a traffic-light. Besides the intersection's geometry and current states of the traffic-light, predicted times until the next switches are visualized by receding bars. Grey shadows on the lanes indicate queues of vehicles in front of the stopping line. The red arrow marks the car's position on a lane.

3.4.6 Acceptance of the System

It is not clear in advance how other traffic participants will react to a car which behaves as explained above. Especially the presented deceleration processes could possibly be surprising or even frustrating for others, for example, when an equipped car is approaching a green light which will shortly turn red. The application may begin a stopping maneuver, but other drivers behind us may want to keep driving at their desired velocity, as they are not informed about the change to come. One could also imagine other cars to overtake an equipped car if it has stopped relatively far away from the stopping line. It is also not clear if other cars, which are following an equipped car, will benefit from the additional information.

During our test drives in the city of Braunschweig, we did not experience frustration of other drivers or an overtaking maneuver in front of the stopping line. Additionally, there is some research by the *German Aerospace Center*, which is a partner in the UR:BAN-project, concerning the acceptance of assistance systems by other drivers. In particular, in [83], the authors find out that drivers in non-equipped cars do actually benefit from leading cars which run a traffic-light assistance system. More specifically, the first non-equipped follower behind an equipped car is mainly influenced and adapts his behavior in a way that he also profits from the application. We also experienced this behavior during our test drives. In [92], parameters which measure the degree of approval of the system, are introduced. Early analysis of experiments with test subjects in a driving simulator reveals that there is a high level of acceptance regarding the introduced maneuvers.

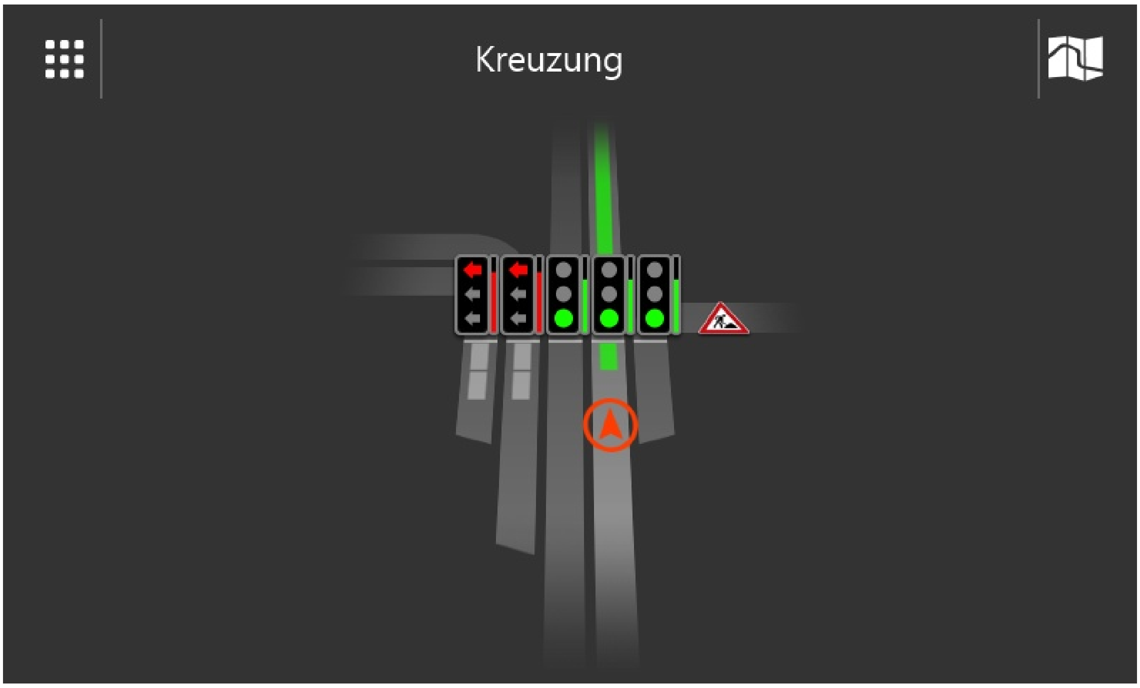


Figure 3.7: Example of the information the MIB-HMI shows, when the car approaches a traffic-light. Besides the information shown in Figure 3.6, the symbol for road works indicates road works on a lane.

In this section, we have seen the concept, functionality, and implementation of a driver-assistance system so far. Besides results concerning the practicability and acceptance of the system, we are interested in effects on real-world traffic-flow. For the setting and results of experiments using a microscopic traffic simulation software, cf. Section 6.4. In Section 6.4.1, the quality of the implemented MPC-process is considered. In the remainder of this section, we present another – purely informing – driver-assistance system developed in the UR:BAN-project.

3.5 Developing an Information System

In addition to the driver-assistance system which automatically performs certain maneuvers, a purely visual system has been developed in the project UR:BAN. It is called *MIB-HMI*, due to the fact that this HMI is displayed on the so-called MIB-screen located in the center of the car's dashboard. This system gathers all the information which is available at a traffic-light controlled intersection. Beyond the well described data about the traffic-light's position, the exact geometry of the junction, and dynamic information about the light's signal state and upcoming changes, there can be further information about other traffic participants available.

In the project UR:BAN, different systems have been developed, which are based on communication between a traffic-light and other traffic participants, e. g., cars, cyclists, emergency vehicles, and traffic cones. The latter ones can, for example, indicate a temporary closure of a lane. For further information on those systems, cf. [109]. All of them have in common that the traffic-light is aware of the presence and position of the listed traffic participants because of wireless communication. The installed AU, cf. Section 3.4.2, can mirror this information, or, more specifically, spread the received messages. We want the MIB-HMI to provide information to the driver, allowing him or her to make better and earlier decisions,

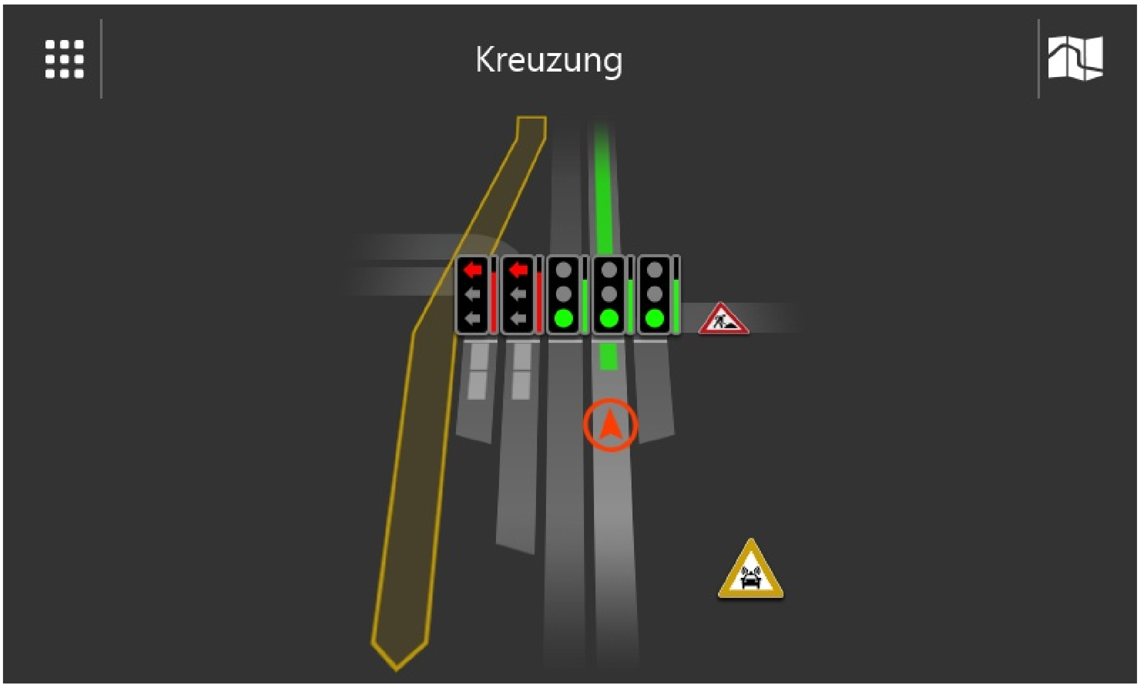


Figure 3.8: Example of the information the MIB-HMI shows when the car approaches a traffic-light. Besides the information shown in Figure 3.7, the presence and possible route of an emergency vehicle is illustrated.

which typically occur at intersections, e. g., which lane is the best choice to drive on. Figure 3.6 shows the HMI's content when approaching an intersection. Here, the geometry of the intersection, the traffic-light's current states, and the predicted times until the next switches are visualized. Additionally, the gray shadows on some lanes indicate queues of vehicles in front of the stopping line, based on both direct measurements and estimates by the traffic-light. The red arrow marks the current position of the car. For this purpose, the car's positioning data are used to determine the lane the car is driving on, and to indicate the distance to the stopping line. The green area on the lane the car is driving on implies that the cars in front of the stopping line will pass the traffic-light during the current green phase. In Figure 3.7, additional information is provided. The driver is informed about road works, which are indicated by a triangle including the according symbol. Moreover, in Figure 3.8, an approaching emergency vehicle is indicated. As it also broadcasts messages including information about its turn signal, we can mark the route it will probably take.

The concept of enabling cooperation among different traffic participants and infrastructure provides a variety of concepts and resulting systems. A majority of them will eventually lead to an enhancement of traffic flow and certainly security issues. If we carry the ideas of this section forward, we end up with cooperation of cars and traffic-lights in both directions. This means, the traffic-light processes data about position, velocity, and other information of oncoming cars. Resulting systems, which incorporate these information, can lead to a further enhancement in terms of traffic flow by adapting the light's signal states highly dynamically. In Section 5, we present an algorithm which regulates the transit of multiple cars based on communication between the cars and the traffic-light in both directions.

Another extension of the RACC could be to react not only to already performed movements of other cars, which are detected via radar sensors, but also intended trajectories could be shared among different traffic participants and incorporated in the individual strategies. Especially dead times that arise until a movement is detected could be reduced and resulting trajectories could be more comfortable for the driver.

When we consider autonomously driving cars, we should be aware that cooperation will not merely be optional but necessary. Imagine a non-negligible amount of autonomous cars which respect all traffic regulations properly. Apart from a potential increase in security, traffic flow could get worse in some places, cf. [66]. Thus, cooperation between cars, as well as cars and infrastructure is a highly dynamic and growing field. In the next section, we will follow the idea of cooperation. Consequentially, we focus on globally optimized traffic flow.

4 Optimizing Traffic Flow

Following the argumentation of the previous section, we now focus on globally optimized traffic flow. The term *globally* does not refer to obtaining a global optimal solution. Globally means that traffic as whole – from a global point of view, i. e., by regarding traffic-lights and all cars togehter – shall be optimized. In Section 3.1, we have seen some examples for applications which aim to improve traffic flow at intersections by considering each car individually. In the literature, a large number of papers analyze problems in the context of optimizing traffic at traffic-light controlled intersections. Often *traffic-lights* and *cars* are viewed separately. Big efforts have been made and many algorithms have been studied to influence the behavior of traffic-lights in order to increase traffic flow. For instance, in [33, 34] mixed integer linear programming techniques are applied to this problem. Other publications focus on self-adapting traffic-lights [110] and evolutionary algorithms [91]. Another common approach is to improve a car's individual behavior when approaching a traffic-light while keeping the signal phases fixed. We have seen examples from the literature as well as a detailed description of an assistance system, which is developed in this thesis, in the previous section.

We now study traffic at traffic-light intersections as a whole. To this end, we first discuss common approaches based on different kinds of modeling traffic and resulting solving methods. Subsequently, we develop a MILP in Section 4.2 including variables for the states of both the cars and the traffic-lights on a fixed time horizon. Simultaneously, realistic motion dynamics are incorporated. In contrast to other publications, e. g., [112], which also contain controls for cars and traffic-lights, we obtain solutions for a globally optimized traffic flow. These offline-calculated solutions can serve as a benchmark for other approaches which try to improve traffic flow at intersections and can provide heuristics, e. g., for light-cycle optimizing algorithms. Later on, in Section 6, we compare the quality of traffic flow achieved by the RACC with the globally optimal traffic flow. As the behavior of traffic-lights in terms of allowed switching schemes is legally restricted, we incorporate the majority of these regulations (in German traffic) into the MILP in Section 4.2.5.

Due to the high complexity and runtimes of the presented MILP, we develop solving strategies in Section 4.5 to reduce computing times. These are also evaluated numerically in Section 6. Nevertheless, this approach is not suitable for a real-time application due to high solving times, which are necessary for reasonable amounts of cars and time horizons. The main purpose of the model is to provide solutions which can serve as a benchmark for other methods or algorithms.

Parts of this section are currently under review for a publication together with Mirko Hahn and Sebastian Sager. Mirko Hahn and Sebastian Sager contributed to ideas leading to the different optimization models in this section. The joint work with them also lead to the presented solving methods. Mirko Hahn highly contributed to an implementation of the models and solving methods which are under numerical investigation in Section 6. Parts of the therein presented results are also included in the publication.

4.1 Motivation and Setting

A detailed overview and discussion of methods for modeling traffic is given in [6]. We will highlight two major types of models in the following.

4.1.1 Macroscopic Traffic Models

A common approach to model traffic flow are *macroscopic traffic models*, which go back to [72] and [86]. These models basically describe traffic on an abstract level and do not consider each car individually. In many papers which aim to improve traffic flow in a certain sense, traffic is modeled as macroscopic flow. Here, the movement of cars is often described by a two-dimensional partial differential equation for the traffic density $\rho(x,t)$, where $t \geq 0$ denotes time, and $x \in \mathbb{R}^+$ denotes the position. In [45, 46], the authors consider the problem of computing optimal traffic-light programs for intersections in an urban setting using traffic densities. Also in [73] the presented solving strategies for optimizing urban traffic including intersections are based on a macroscopic traffic model.

4.1.2 Microscopic Traffic Models

In contrast to macroscopic models, *microscopic traffic models* invoke ordinary differential equations for each vehicle. Here, the individual position and velocity, as variables depending on time, describe the state of the system. Reducing the actions of a single vehicle to longitudinal movement on a single lane, the basic structure of a microscopic model is as follows:

$$\begin{cases} \frac{ds_i}{dt} = v_i, \\ \\ \frac{dv_i}{dt} = a_i(s_1, \dots, s_N, v_1, \dots, v_N), \end{cases}$$

where $i \in \{1, \dots, N\}$ and a_i is the current acceleration of the i-th vehicle depending on the position $\boldsymbol{s}$ and velocity $\boldsymbol{v}$ of all vehicles. Due to the resulting large systems of ordinary differential equations, many models reduce the acceleration of the i-th vehicle to a reaction depending only on a subset of the other cars. Often only the directly preceding car is taken into account. Hence, these models are also called *car-following models*. Common examples from the literature are the models of Krauß and the IDM by Treiber and Kesting, cf. [54, 59, 99]. A major advantage of microscopic models compared to macroscopic models is that they regard the motion of each vehicle individually, which allows us to invoke restrictions on physical quantities, such as acceleration and jerk. This rather realistic description of traffic flow results in a higher complexity of the model.

In this thesis, we introduce different approaches for improving traffic flow at urban intersections. Besides the pure design of these methods, we also want to compare them in terms of quality of the resulting traffic flow and other criteria. Therefore, we have to make sure that they are comparable among each other. As one of the approaches is a driver-assistance system, cf. Section 3, we want the other solving methods to have similar properties. Thus, using a macroscopic traffic model for optimizing traffic flow, which is the main issue of this section, would be rather inappropriate. In the following paragraph, we develop a mixed integer linear program which incorporates the longitudinal motion model of the RACC and is therefore based on a microscopic point of view.

4.2 Developing a Mixed Integer Linear Program

In this section, we develop an MILP which describes the traffic flow of all cars on a simple urban road network within a fixed time interval $\mathcal{T} := [0, T_N]$. In addition to the behavior of all cars at any timestep, we are interested in the required signal states of the network's traffic-lights.

The basic structure of our scenario, for which we explain the model, is as follows: straight roads intersect in a single intersection. Each road consists of two lanes running in opposite directions. In each lane, a traffic-light tl from the set of traffic-lights TL is installed in front of the intersection. However, the derived model allows to represent networks with multiple intersections and more than two roads. For simplicity, we stick to the stated simple example when introducing the model. As we want to determine the optimal movement of the cars along the road offline, only the time each car enters the network, called *arrival-time* $\bar{t}_c \in \mathcal{T}$, and its velocity at this time $\bar{v}_c > 0$ are fixed. All cars enter the network in the origin of their respective lane. In order to facilitate comparisons of traveled distances of different cars, e. g., to prevent collisions of cars on the same lane, each lane starts at distance zero. Thus, the initial value of each car's traveled distance is also zero. Traffic-lights are installed at fixed positions which are determined by their respective distance to the lane's origin. The basic layout for a network consisting of two lanes which intersect in a single intersection is visualized in Figure 4.1. In particular, we need the model to contain the following characteristics:

- a car's longitudinal movement,
- realistic bounds on the car's motion,
- logic of traffic-lights, which basically permits and prohibits cars to be on the intersection in certain timesteps,
- optionally, legal regulations on the behavior of the traffic-lights,

The goal is to determine the optimal traffic flow from a global point of view. More specifically, we are not only interested in solutions which consider the behavior of each car and traffic-light as it would be the case if everything was controlled from a global control unit. We also want to determine a global optimum (in contrast to local optima) as the obtained solution should serve as a benchmark for other methods which increase traffic flow.

4.2.1 Cars and Motion Model

For the purposes of the model, a car c in the set of all cars C is a moving occupant of a stretch of road. At any given time, it moves on a road using a specific lane. A lane may be used by multiple cars. Two cars using the same lane may not simultaneously occupy the same space within that lane. We refer to our efforts to prevent this as *collision-prevention*. Note that we only consider longitudinal movement and that overtaking maneuvers are forbidden.

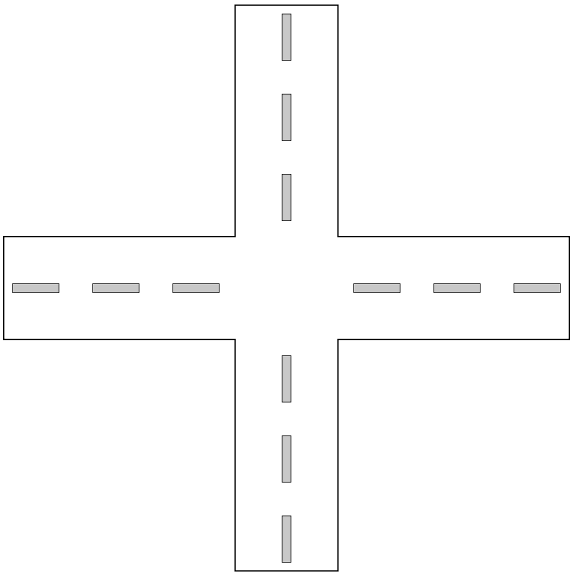


Figure 4.1: Simple intersection consisting of two intersecting roads. Each road consists of two lanes running in opposite directions.

The movement of any given car is governed by the following laws of motion. Note that we focus on linear constraints as friction is negligible in an urban setting which means the trade-off between realism and low runtimes is acceptable:

$$\begin{aligned}
\dot{s}(t) &= v(t) && \forall t \in \mathcal{T}, \\
\dot{v}(t) &= a(t) && \forall t \in \mathcal{T}, \\
\dot{a}(t) &= j(t) && \forall t \in \mathcal{T}, \\
s(t) &= 0 && \forall t \in [0, \bar{t}_c], \\
v(\bar{t}) &= \bar{v} && \forall t \in [0, \bar{t}_c], \\
a(\bar{t}) &= 0 && \forall t \in [0, \bar{t}_c], \\
v(t) &\in \left[v^{min}, v^{max}\right] && \forall t \in \mathcal{T}, \\
a(t) &\in [a_{min}, a_{max}] && \forall t \in \mathcal{T}, \\
j(t) &\in [j_{min}, j_{max}] && \forall t \in \mathcal{T}.
\end{aligned} \tag{4.1}$$

In the ODE-system, $s(t)$ is the traveled distance of the car, which is the position of the car's front on the lane. The initial distance is set to 0 for each car and increases according to the car's movement on the lane. Its velocity is encoded in $v(t)$, $a(t)$ is its acceleration, and $j(t)$ its jerk. The bounds $0 \leq v^{min} < v^{max}$, $a^{min} < 0 < a^{max}$, and $j^{min} < 0 < j^{max}$ are due to physical and comfort restrictions and set for each car individually. Note that defining the IVP as above, i. e., on the whole time horizon with initial values fixed until the car enters the network, and regarding a shorter time horizon beginning at $\bar{t}_c$ for each car is equivalent. The linear ODE-system is discretized using a direct collocation method, cf. Section 2.1.1.4, with an equidistant discretization $T := \{0, \ldots, N\}$ of $\mathcal{T}$. In fact, the explicit Euler method on T with step length $dt := T_N/N$ is an appropriate choice for the rather simple motion model. This yields the following system of equations. Note that the variables for the jerk $j(t)$ are omitted as they are only bounded from above and below. These bounds are enforced directly by bounding the difference between the accelerations in succeeding time steps:

$$\begin{aligned}
s_{c,t+1} &= s_{c,t} + v_{c,t} \cdot dt && \forall c \in C,\ t \in T \setminus \{N\}, && (4.2)\\
v_{c,t+1} &= v_{c,t} + a_{c,t} \cdot dt && \forall c \in C,\ t \in T \setminus \{N\}, && (4.3)\\
s_t &= 0 && \forall c \in C,\ t \in \{0, \dots, \bar{t}_c\}, && (4.4)\\
v_t &= \bar{v}_c && \forall c \in C,\ t \in \{0, \dots, \bar{t}_c\}, && (4.5)\\
a_t &= 0 && \forall c \in C,\ t \in \{0, \dots, \bar{t}_c\}, && (4.6)\\
v_c^{min} &\le v_{c,t} && \forall c \in C,\ t \in T, && (4.7)\\
v_{c,t} &\le v_c^{max} && \forall c \in C,\ t \in T, && (4.8)\\
a_c^{min} &\le a_{c,t} && \forall c \in C,\ t \in T, && (4.9)\\
a_{c,t} &\le a_c^{max} && \forall c \in C,\ t \in T, && (4.10)
\end{aligned}$$

as well as the inequalities:

$$j_c^{min} \le \frac{1}{dt} \cdot (a_{c,t+1} - a_{c,t}) \le j_c^{max} \quad \forall c \in C,\ t \in T \setminus \{N\}. \tag{4.11}$$

For simplicity, we demand $\bar{t}_c \in T$ for all cars. After modeling the movement of individual cars, we now focus on preventing collisions between succeeding cars driving on the same lane. To this end, we introduce the set of predecessors C_c^{pred} for each car c and constraints of the form:

$$s_{c,t} \le s_{d,t} - l_d - g_c \quad \forall c \in C,\ d \in C_c^{pred},\ t \in T, \tag{4.12}$$

where l_d is the length of car d and g_c is the (optional) safety gap maintained by c. As we are considering networks with single lanes without overtaking maneuvers and turning maneuvers, one has $|C_c^{pred}| \le 1$ for each car $c \in C$ and we write $pred(c)$ for the unique predecessor of a car c. Still, we need a way to prevent collisions on the intersection between cars driving on different lanes.

4.2.2 Triggers

While collision-prevention between cars in a single lane is essential, the main goal is to model traffic at intersections. Collision-prevention between different lanes is a crucial component of this. A basic insight is that such interaction always yields constraints that are only relevant for cars that enter specific sections of the road. For instance, an intersection between two lanes is essentially a small section on either lane that cannot be driven on freely. Generally speaking, we introduce the concept of *trigger zones*. Simply put, a trigger zone is a subsection of a lane that is special in that there is a set of constraints that apply to the car's behavior only if the car is located within that subsection.

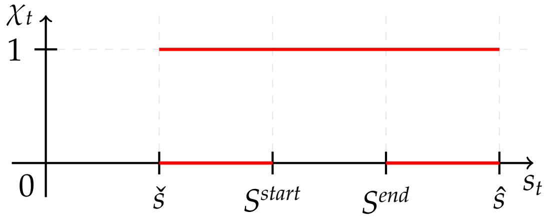


Figure 4.2: Admissible values for the distance variable s_t of a single car if the trigger variables of a single trigger zone are $\chi_t = 0$ and $\chi_t = 1$.

In Section 4.2.3, we describe how to use trigger zones to model traffic-lights. Furthermore, triggers can be used to a wide range of effects. For instance, they may also be used to impose local velocity constraints.

4.2.2.1 Modeling Triggers In order to enable and disable constraints based on the location of the car without leaving the framework of linear programming, we use big-M formulations, cf. Section 2.12. We assume that we have binary variables $\chi_{tz,t}$ for $t \in T$ and a trigger zone tz from the set of all trigger zones in the network TZ that are guaranteed to be 1 if a car is located within the trigger zone and can be chosen to be 0 if no car is located within the trigger zone at timestep t. As we mainly discuss the mechanism for a fixed single trigger zone, we simply write χ_t instead of $\chi_{tz,t}$. We now consider a generic linear inequality constraint of the form

$$\boldsymbol{a}^T \boldsymbol{x} \leq b.$$

By rephrasing this as big-M formulation

$$\boldsymbol{a}^T \boldsymbol{x} - M \cdot (1 - \chi_t) \leq b,$$

where $M \geq \max\left(\boldsymbol{a}^T \boldsymbol{x} - b\right)$, we can ensure that the constraint becomes redundant if $\chi_t = 0$. We call the binary vector $\boldsymbol{\chi}$ containing all χ_t for the whole time horizon *trigger indicator*. The main question then becomes how to acquire such a trigger indicator.

4.2.2.2 Variants for Trigger Indicators There are multiple ways of modeling trigger zones with the ultimate goal of turning a given vector of binary variables $\boldsymbol{\chi}$ into a trigger indicator. For the purpose of this discussion, we assume that there is a known lower bound $\check{s}_c$ and a known upper bound $\hat{s}_c$ on a car's distance variables $s_{c,t}$. Furthermore, we consider a trigger indicator with associated trigger zone $\left[S^{start}_{tl}, S^{end}_{tl}\right]$. For the sake of simplicity, we assume that $\check{s}_c < S^{start}_{tl} < S^{end}_{tl} < \hat{s}_c$. As we only regard a single car for our considerations in this section, we omit the indices for its individual variables. We do the same for indices of a single trigger zone.

First, consider Figure 4.2. It shows the values the distance variable of a single car should be able to attain for $\chi_t = 0$ and $\chi_t = 1$ of a trigger zone. Note that for $\chi_t = 0$, the permissible distance values form a non-convex set. This means that there is no way to find a convex set of pairs (s_t, χ_t) that contains all of the marked

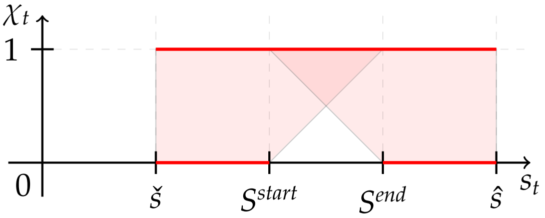


Figure 4.3: Two convex sets of pairs (s_t, χ_t). A binary variable indicates in which set the solution is located.

points but does not also contain points with $\chi_t = 0$ and $s_t \in \left(S^{start}, S^{end}\right)$. It follows that there is also no polyhedron that meets these criteria. Since we are interested in calculating global optima, it is our aim to design convex feasible sets. Note that *global* refers here to the kind of optimum (in contrast to local optimum). Our main goal is to find a linear programming formulation as these are additionally rather comfortable to solve via established solver software. In order to achieve such a formulation for the trigger mechanism, we must add at least one variable. The variants presented in this section represent different ways of adding the necessary variables. Additionally, we try to model the trigger mechanism in a way that does not require a separate trigger variable χ_t for every pair of car and trigger zone. Therefore, some of the presented variants allow trigger variables for a single traffic-light to be shared among cars driving on a certain lane.

Type I: Outer-Convexification Triggers The first way one could go about adding variables is depicted in Figure 4.3. Here, the feasible set is split into two convex sets. We introduce a second type of binary variables χ'_t for $t \in T$ that indicate which of the two "halves" our solution is located in and refer to these variables as the *hull choice variables*. One can choose different variants of this approach, e. g., symmetry is optional. One could generalize the approach by choosing two parameters $\lambda_0, \lambda_1 \geq 0$ with $\lambda_0 + \lambda_1 \geq 1$ which indicate how far the hulls reach into the trigger zone. The hulls would then be defined as the convex hulls of the (s_t, χ_t) points

$$(\check{s}, 0)\,,\ \left(S^{start}, 0\right),\ \left((1-\lambda_0)\cdot S^{start} + \lambda_0 \cdot S^{end}, 1\right), (\check{s}, 1)$$

for $\chi'_t = 0$ and

$$(\hat{s}, 0)\,,\ \left(S^{end}, 0\right),\ \left((1-\lambda_1)\cdot S^{end} + \lambda_1 \cdot S^{start}, 1\right), (\hat{s}, 1)$$

for $\chi'_t = 1$, respectively. Figure 4.3 depicts these hulls for $\lambda_0 = \lambda_1 = 1$. We can now model a trigger zone via the constraints:

$$((1-\lambda_1)\cdot S^{end} + \lambda_1 \cdot S^{start} - \check{s}) \cdot \chi'_t - s_t \leq -\check{s} \qquad \forall t \in T, \quad (4.13)$$

$$\lambda_1 \cdot (S^{start} - S^{end}) \cdot \chi_t + (S^{end} - \check{s}) \cdot \chi'_t - s_t \leq -\check{s} \qquad \forall t \in T, \quad (4.14)$$

$$((1-\lambda_0)\cdot S^{start} + \lambda_0 \cdot S^{end} - \hat{s}) \cdot \chi'_t + s_t \leq (1-\lambda_0)\cdot S^{start} + \lambda_0 \cdot S^{end} \qquad \forall t \in T, \quad (4.15)$$

$$\lambda_0 \cdot (S^{start} - S^{end}) \cdot \chi_t + (S^{start} - \hat{s}) \cdot \chi'_t + s_t \leq S^{start} \qquad \forall t \in T. \quad (4.16)$$

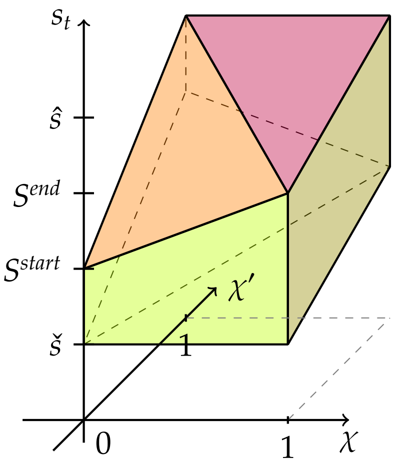


Figure 4.4: Polyhedron resulting from the constraints, which define an outer-convexification trigger.

The resulting three-dimensional polytope for each $t \in T$ defined by Constraints (4.13)–(4.16) and $0 \leq \chi_t, \chi'_t \leq 1$ is visualized in Figure 4.4. Again, $\lambda_0 = \lambda_1 = 1$ holds.

Type II: Enter-Leave Triggers In the enter-leave trigger formulation, we abandon the single indicator variable in favor of two binary indicator variables χ_t^{in} and χ_t^{out}. We want to establish the following logical implications:

$$\begin{aligned} \chi_t^{in} &= 0 \Rightarrow s_t \leq S^{start}, \\ \chi_t^{out} &= 0 \Rightarrow s_t \geq S^{end}. \end{aligned}$$

If these implications hold, it follows that

$$s_t \in \left(S^{start}, S^{end}\right) \Rightarrow \chi_t^{in} = 1 \wedge \chi_t^{out} = 1.$$

If we can also guarantee that χ_t^{in} and χ_t^{out} can be set to 0 if $s_t \leq S^{start}$ and $s_t \geq S^{end}$, respectively, they can be used to acquire a meaningful trigger indicator via

$$\chi_t^{in} + \chi_t^{out} - \chi_t \leq 1 \quad \forall t \in T. \tag{4.17}$$

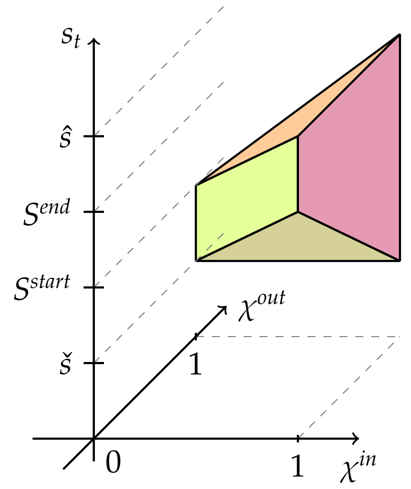


Figure 4.5: Polyhedron resulting from the constraints, which define an enter-leave trigger.

We ensure that when $s_t \notin \left(S^{start}, S^{end}\right)$, either χ_t^{in} or χ_t^{out} can always be assigned the value 0. In this variant of modeling a required trigger zone, we end up with the additional inequalities:

$$\chi_t^{in} + \chi_t^{out} \geq 1 \qquad \forall t \in T, \tag{4.18}$$

$$\left(S^{end} - \check{s}\right) \cdot \chi_t^{out} + s_t \geq S^{end} \qquad \forall t \in T, \tag{4.19}$$

$$\left(S^{start} - \hat{s}\right) \cdot \chi_t^{in} + s_t \leq S^{start} \qquad \forall t \in T. \tag{4.20}$$

Note that for $\chi_t^{out} = \chi_t^{in} = 1$, Equation (4.17) guarantees that $\chi_t = 1$. It is therefore sufficient to demand that $\chi_t \in [0, 1]$. χ_t need not be explicitly marked as binary variables. In Figure 4.5, the polyhedron for each $t \in T$ defined by Constraints (4.18)–(4.20) and $\chi_t^{in}, \chi_t^{out} \leq 1$ is depicted. Table 4.1 lists the possible configurations of the χ-type variables depending on the car's position.

An advantage of both formulations is that the trigger variables χ_t can be shared among all cars driving on the same lane. Nevertheless, we have to introduce the additional binary variables χ_t', and $\chi_t^{in}, \chi_t^{out}$, respectively for every pair of car and trigger zone.

Type III: Three-Way-Split Triggers For the sake of completeness, we briefly address a third formulation for modeling a car's position relative to a trigger zone. In three-way split triggers, we partition the car's distance-axis into three sections

Table 4.1: Possible values for the indicator variables of an enter-leave trigger depending on the car's current position s_t

s_t	χ_t^{in}	χ_t^{out}	χ_t
$[\check{s}, S^{start})$	$\{0, 1\}$	1	$[\chi_t^{in}, 1]$
$(S^{end}, \hat{s}]$	1	$\{0, 1\}$	$[\chi_t^{out}, 1]$
$[S^{start}, S^{end}]$	1	1	1

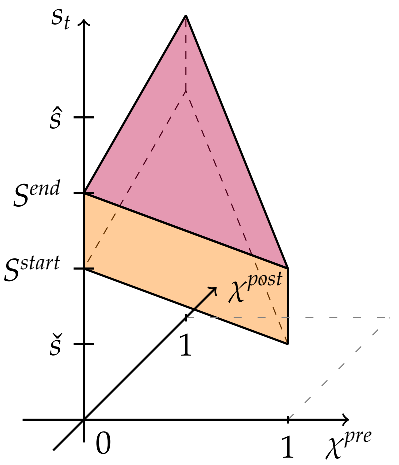


Figure 4.6: Polyhedron resulting from the constraints, which define a three-way-split trigger.

and introduce two additional binary indicators χ_t^{pre} and χ_t^{post} to indicate together with χ_t the car's presence before, after and within the trigger zone, respectively. These variables are linked using a linear equality constraint

$$\chi_t + \chi_t^{pre} + \chi_t^{post} = 1 \quad \forall t \in T. \tag{4.21}$$

Using Equation (4.21), we can induce the logic of a trigger-indicator by forcing χ_t^{pre} and χ_t^{post} to be 0 if $s_t \in \left(S^{start}, S^{end}\right)$ and setting them to 1 if $s_t \leq S^{start}$ or $s_t \geq S^{end}$, respectively. Constraints that fulfill the purposes are:

$$\begin{aligned} \chi_t^{pre} + \chi_t^{post} &\leq 1 && \forall t \in T, && (4.22)\\ \left(S^{end} - S^{start}\right) \cdot \chi_t^{pre} + \left(S^{end} - \hat{s}\right) \cdot \chi_t^{post} + s_t &\leq S^{end} && \forall t \in T, && (4.23)\\ \left(S^{start} - \check{s}\right) \cdot \chi_t^{pre} + \left(S^{start} - S^{end}\right) \cdot \chi_t^{post} + s_t &\geq S^{start} && \forall t \in Tl. && (4.24) \end{aligned}$$

Note that inequality (4.22) is already implied by equation (4.21). Again, the variable χ_t can be chosen as $\chi_t \in [0, 1]$. Here, equation (4.21) guarantees that the trigger indicator χ_t has to be either equal to one or equal to zero. The polyhedron for every $t \in T$ resulting from the Constraints (4.21)–(4.24) and $0 \leq \chi_t^{pre}, \chi_t^{post}$ is visualized in Figure 4.6. The values that the χ-type variables are allowed to attain, based on the car's current position, are listed in Table 4.2.

Table 4.2: Possible values for the indicator variables of an three-way-split trigger depending on the car's current position s_t

s_t	χ_t^{pre}	χ_t^{post}	χ_t
$[\check{s}, S^{start})$	1	0	0
$(S^{end}, \hat{s}]$	0	1	0
$[S^{start}, S^{end}]$	0	0	1

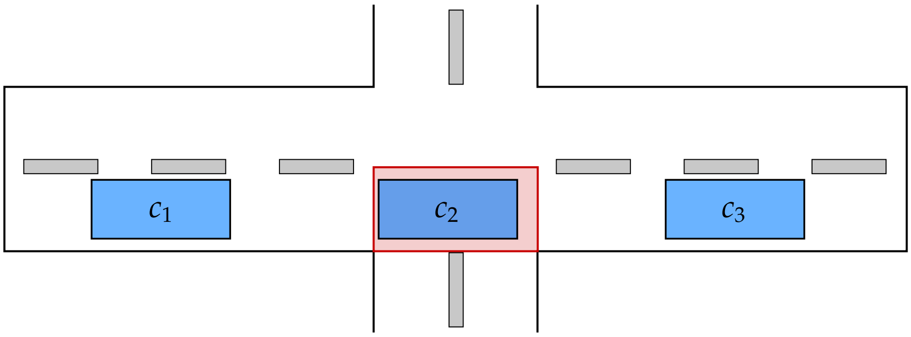


Figure 4.7: Three cars $c_1, c_2, c_3 \in C$ driving on a lane in the network. The configuration $(\chi_t^{in}, \chi_t^{out}, \chi_t)$ reads as $(\{0,1\}, 1, \{\chi_t^{in}, 1\})$ for c_1, $(1,1,1)$ for c_2, $(1, \{0,1\}, \{\chi_t^{out}, 1\})$ for c_3. As mentioned above, the variables χ_t need not be chosen binary due to Equation (4.17). This leads to the attainable values $(\{0,1\}, 1, [\chi_t^{in}, 1])$ for c_1, $(1,1,1)$ for c_2, $(1, \{0,1\}, [\chi_t^{out}, 1])$ for c_3.

An advantage of the three-way split formulation lies in the fact that it provides a guarantee that χ_t acts as proper trigger indicator. While the other two formulations allow for χ_t to jump to non-zero values (including 1) outside of the trigger zone as long as this has no negative impact on the objective function value, the three-way split formulation does not allow this.

However, this property can also be seen as the formulation's weakness. In the other formulations, trigger indicators can be shared among multiple cars, more specifically, among cars heading towards the same trigger zone. In three-way split triggers, this would not be possible as one car might force the indicator to assume the value 0 whereas another might force it to assume the value 1, leading to infeasibility. Three-way split triggers always require indicators to be car specific.

Numerical investigation of the different trigger formulations showed that type II yields the best runtimes in most cases. Also compared to formulations of the outer-convexification triggers with $\lambda_0 = 0$ and $\lambda_1 = 1$. For these values, no overlap of the two halves exists, cf. Figure 4.3. This is why we focus on enter-leave triggers. Figure 4.7 shows the three main configurations the trigger variable χ_t attains depending on the values of χ_t^{in} and χ_t^{out} for the three different cars.

4.2.3 Traffic-Lights

So far, we have addressed general ways to cause certain constraints to become relevant based on the current location of a car. In this section, we discuss ways of using the trigger mechanism to model the effect of a traffic-light. Traffic-lights are essentially reduced to sections of road that can only be driven on under certain circumstances. Driving on one of these sections will then prevent other sections from being driven on.

If we recall the concept of a trigger variable, we have a mechanism that allows us to retrieve the presence of a car on a trigger zone. Following this idea, we introduce a trigger zone for each traffic-light. The area where the lanes overlap on the intersection contains the trigger zones. In the course of this thesis, we refer to this area as *intersection area*. Figure 4.8 schematically shows an intersection with four lanes on two intersecting roads. The car c with its current position s_t and the trigger zone for the traffic-light the car is approaching are also visible. Additionally, trigger variables $\chi_{tl,t}$ for each of the traffic-lights tl of the intersection and timestep

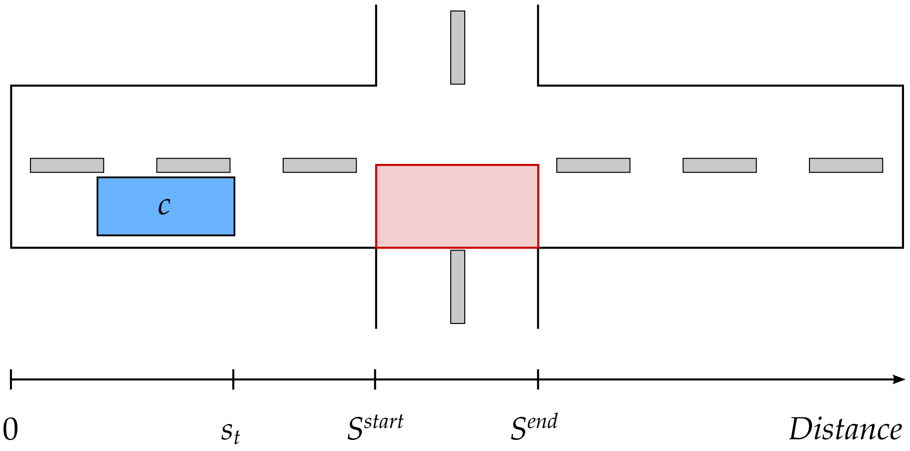


Figure 4.8: Car c with its current position s_t driving on a lane towards an intersection. The corresponding trigger zone for this lane starts at S^{start} and ends at S^{end}.

$t \in T$ are introduced. We can interpret a traffic-light tl to have a green light at timestep t if the corresponding trigger variable $\chi_{tl,t}$ equals 1. If $\chi_{tl,t} = 0$ holds, the traffic-light is set to red. If possible, we omit additional indices tl for traffic-lights. They are only used if we want to distinguish between trigger variables of different traffic-lights.

We now regard the set of all traffic-lights TL. For this set, we define a decomposition into subsets

$$TL = \dot{\bigcup_{l=1}^{L}} TL_l,$$

where $L \in \mathbb{N}$ and the sets TL_l contain *conflicting* traffic-lights. Conflicting means that only a single traffic-light $tl \in TL_l$ is allowed to be set to green in each timestep. This is enforced by the constraint:

$$\sum_{tl \in TL_l} \chi_{tl,t} \leq 1. \qquad \forall t \in T,\ l \in \{1, \dots, L\} \tag{4.25}$$

In particular, if the network consists of a single intersection and all traffic-lights are conflicting, it holds that $L = 1$. Theoretically, it is also possible to make traffic-lights belonging to different intersections conflicting.

4.2.4 Objective Function

In the previous sections, we focused on modeling longitudinal movement of cars on straight roads and the inhibiting effects of traffic-lights that govern the traversal of cars on intersecting sections of road. As we are looking for an optimal traffic flow of multiple cars on a fixed time horizon from a global point of view, we have to determine an objective function for that.

Optimal traffic flow could mean that one wants to reduce the (possibly weighted) overall traveling time of all cars for reaching their destination from a given origin. In our scenario, all cars have the same origin and no particular destination. As the time horizon is also fixed, minimizing traveling time for a certain route is similar to maximizing the covered distance at the end of the horizon in our scenario. Remember that we want to optimize traffic flow as a whole. Therefore, we maximize the sum of the distances of all cars in the last timestep N:

$$\max \quad \sum_{c\in C} s_{c,N}. \tag{4.26}$$

Thinking about economical issues one could also try to minimize the overall emissions. As the presented model does not include exhaust rates or something similar, we can assess a car's squared acceleration. Note that we would lose the model's linearity:

$$\min \quad \sum_{\substack{c\in C,\\ t\in T}} a_{c,t}^2. \tag{4.27}$$

One could also think about objective functions that take fairness into account. An objective function as in (4.26) could lead to a traffic flow where some cars have to wait for a relatively long time before they can pass an intersection (e. g., when the number of incoming cars on intersecting roads is very unbalanced) for the sake of a high overall traffic flow. Nevertheless, we focus on (4.26) as objective function since we are mainly interested in traffic flow which is optimal regarding the whole system.

For convenience, we list all constraints describing the passage of multiple vehicles over an intersection below. We refer to the resulting MILP as *global-MILP*. Remember that we choose to model the trigger mechanism via the trigger type II. In contrast to prior notation, we do not omit the indices for the cars and traffic-lights. Note that also more complex networks than the rather simple one which served for introducing the model can be represented. For notational simplicity, we introduce the set of traffic-lights which a car c passes during its traversal of the network as TL_c.

$$
\begin{aligned}
\max_{a,v,s,\chi^{in},\chi^{out},\chi} \quad & \sum_{c \in C} s_{c,N} && && (4.28)\\
\text{s.t.:} \quad & s_{c,t+1} = s_{c,t} + v_{c,t} \cdot dt && \forall c \in C,\ t \in T \setminus \{N\}, && (4.29)\\
& v_{c,t+1} = v_{c,t} + a_{c,t} \cdot dt && \forall c \in C,\ t \in T \setminus \{N\}, && (4.30)\\
& \frac{1}{dt} \cdot \left(a_{c,t+1} - a_{c,t}\right) \geq j_c^{min} && \forall c \in C,\ t \in T \setminus \{N\}, && (4.31)\\
& \frac{1}{dt} \cdot \left(a_{c,t+1} - a_{c,t}\right) \leq j_c^{max} && \forall c \in C,\ t \in T \setminus \{N\}, && (4.32)\\
& s_{c,t} = 0 && \forall c \in C,\ t \in \{0, \dots, \bar{t}_c\}, && (4.33)\\
& v_{c,t} = \bar{v}_c && \forall c \in C,\ t \in \{0, \dots, \bar{t}_c\}, && (4.34)\\
& a_{c,t} = 0 && \forall c \in C,\ t \in \{0, \dots, \bar{t}_c\}, && (4.35)\\
& v_c^{min} \leq v_{c,t} && \forall c \in C,\ t \in T, && (4.36)\\
& v_{c,t} \leq v_c^{max} && \forall c \in C,\ t \in T, && (4.37)\\
& a_c^{min} \leq a_{c,t} && \forall c \in C,\ t \in T, && (4.38)\\
& a_{c,t} \leq a_c^{max} && \forall c \in C,\ t \in T, && (4.39)\\
& s_{pred(c),t} - s_{c,t} \geq l_{pred(c)} + g_c && \forall c \in C,\ t \in T, && (4.40)\\
& \chi_{c,tl,t}^{in} + \chi_{c,tl,t}^{out} - \chi_{tl,t} \leq 1 && \forall c \in C,\ tl \in TL_c,\ t \in T, && (4.41)\\
& \chi_{c,tl,t}^{in} + \chi_{c,tl,t}^{out} \geq 1 && \forall c \in C,\ tl \in TL_c,\ t \in T, && (4.42)\\
& \left(S_{tl}^{end} - \check{s}_c\right) \cdot \chi_{c,tl,t}^{out} + s_{c,t} \geq S_{tl}^{end} && \forall c \in C,\ tl \in TL_c,\ t \in T, && (4.43)\\
& \left(S_{tl}^{start} - \hat{s}_c\right) \cdot \chi_{c,tl,t}^{in} + s_{c,t} \leq S_{tl}^{start} && \forall c \in C,\ tl \in TL_c,\ t \in T, && (4.44)\\
& \sum_{tl \in TL_l} \chi_{tl,t} \leq 1 && \forall t \in T,\ l \in \{1, \dots, L\}. && (4.45)
\end{aligned}
$$

Following the discussion of this section, we have met all requirements stated in the beginning. In Section 6, we investigate the global-MILP numerically by solving experimental data and comparing runtimes and complexity of the model with the formulations and solving strategies we develop in the course of this section.

4.2.5 Additional Traffic-Light Regulations

While lane block effects are easily implemented and somewhat resemble real traffic-lights, when confronted with dense traffic on multiple sides, they almost certainly switch between sides very rapidly. Though this effect may yield optimal outcomes, it is of limited use due to the fact that traffic-lights exhibiting such behavior cannot be used by regular human drivers and therefore cannot be placed on actual roads unless traffic is fully automated.

In reality, traffic-light programs are subject to additional constraints. They switch at frequencies lower than the frequency of simulation timesteps and must maintain their state for fixed amounts of time. To obtain results that are closer to what would be achievable in reality, lane block effects could be furnished with a kind of program consisting of the following additional parameters, which are closely related to the legal regulations stated in [26]:

- The *pulse* is the interval between two possible changes in a traffic-light's state.
- The *green period* is the time during which cars may enter the intersection. It is characterized by a lower bound expressed in multiples of the pulse interval.
- The *red period* is the period during which cars may not enter the intersection. It is characterized by a lower bound expressed in multiples of the pulse interval.
- The *cycle time* is the time between the beginning of a green period and its successive green period. It is characterized by a lower bound expressed in multiples of the pulse interval.
- The *evacuation time* is a fixed time interval expressed in multiples of the pulse interval. It is enforced at the beginning of the red phase in which crossing cars may still not enter the intersection and is intended to provide time for cars to leave the intersection. While technically part of the red phase, cars may be located inside the intersection during this interval. This means that, as far as lane block effects are concerned, it counts as part of the green phase. The red and green periods are adjusted accordingly if evacuation times are enforced.

By introducing these lane block programs and properly enforcing them, the solutions obtained by the solver can be applied much better to real situations. However, significant changes to the way the model is formulated are required.

4.2.5.1 Pulse Interval The difficulty in implementing pulse intervals lies in whether or not one limits oneself to all traffic-lights in the network being synchronized. The simplest way to maintain a constant state in a lane block for a fixed period of time is to use the same χ_t for multiple timesteps. While this is simple and (in the case of outer-convexification triggers) may substantially reduce the number of binaries, it is not possible to find solutions where traffic-light pulses along the same road are offset by a fraction of the pulse interval to allow for smoother traffic flow. This may impact the quality of the solution.

Another way to model pulse intervals would be to introduce an additional integer offset variable o for every trigger indicator. Let $P \in \mathbb{N}$ be the length of the pulse interval in timesteps and let $o \in \{0, \ldots, P-1\}$. For a timestep $t \in \{2, \ldots, N\}$, we can enforce the pulse interval by inserting the following additional constraints:

$$|\chi_t - \chi_{t-1}| \leq -\frac{1}{P} \cdot \left(t - P \cdot \left\lfloor \frac{t}{P} \right\rfloor + o\right) + 2, \tag{4.46}$$

$$|\chi_t - \chi_{t-1}| \leq \frac{1}{P} \cdot \left(t - P \cdot \left\lfloor \frac{t}{P} \right\rfloor + o\right). \tag{4.47}$$

Figure 4.9 illustrates the mechanism by which these constraints enforce the pulse interval. As we can see, for each period the two graphs intersect in such a way that an indicator change is only allowed when t is exactly at offset o from the beginning of a period. Note that this is only guaranteed to work if we demand that the χ is a binary vector. Otherwise, we cannot guarantee that the absolute value of the

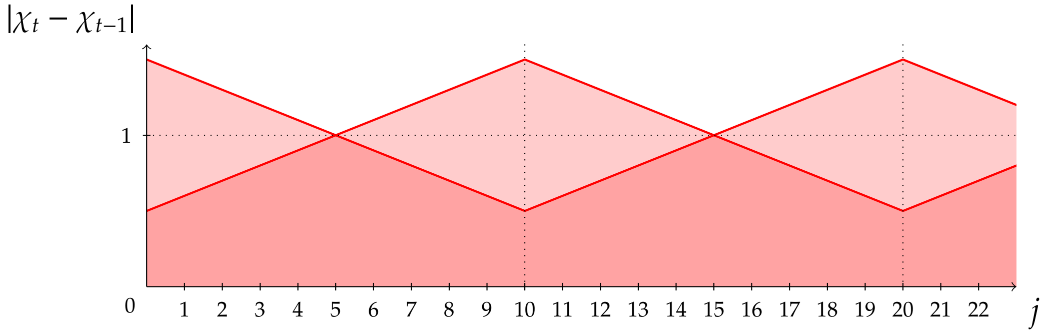


Figure 4.9: Variable offset pulse interval with ($P = 10, o = 5$)

difference $|\chi_t - \chi_{t-1}|$ equals 1 for changing values of χ_{t-1} and χ_t. This has an impact on the enter-leave trigger-formulation and the three-way split which usually do not require the trigger indicator itself to be binary.

4.2.5.2 Limiting the Green Period We now turn our attention towards imposing limits on the green period. For the purpose of this discussion, we assume that the green period has been properly adjusted to account for the fact that evacuation time counts towards the green period rather than the red period (as it would in reality). Let $\check{g} \in \mathbb{N}$ be the minimal number of timesteps for a green period. Our goal is to ensure that the traffic-light can only switch from green to red if it has not been red for at least $\check{g}$ steps. We achieve this by imposing constraints on $|\chi_t - \chi_{t-1}|$, which requires us to demand the integrality of χ_t. To ensure a minimal duration of the green phase, we impose an upper bound on the downward change of χ_t which exceeds 1 only if the green phase is long enough.

$$\chi_{t-1} - \chi_t \leq \frac{1}{\check{g}} \sum_{k=1}^{\check{g}} \chi_{t-k} \tag{4.48}$$

Note that if pulse intervals are implemented through a reduction of the number of indicator variables, we can choose $\check{g}$ to be an appropriate number of pulse intervals. This constraint also needs to be properly adjusted for $t \leq \check{g}$.

4.2.5.3 Limiting the Red Period Limits on the red period are obtained from constraints limiting the green period by replacing χ_t with $1 - \chi_t$ on the right hand side and replacing $\chi_{t-1} - \chi_t$ with $\chi_t - \chi_{t-1}$ on the left hand side of constraints (4.48):

$$\chi_t - \chi_{t-1} \leq 1 - \frac{1}{\check{r}} \sum_{k=1}^{\check{r}} \chi_{t-k}. \tag{4.49}$$

Here, $\check{r} \in \mathbb{N}$ is the minimum number of timesteps (or pulse intervals) within a red period.

4.2.5.4 Enforcing the cycle time The cycle time ensures that there is no oscillating behavior of the traffic-light's states. In real-world traffic-light programs, running on traffic-light intersections in Germany, the time between two successive changes of the traffic-light state from red to green is bounded from below by 30 seconds, cf. [26]. To enable this restriction in the above model, we add the following constraints for a given lower bound of the cycle time $\check{c} \in \mathbb{N}$ and every timestep t:

$$\chi_t - \chi_{t-1} - 1 \leq \chi_{t-i-1} - \chi_{t-i} \quad \forall t \in T, i \in \{1, .., \check{c}\}. \tag{4.50}$$

Again, we have to adjust this for the timesteps $t \leq \check{c}$. In this case we drop the constraints for $i \geq t$.

4.2.5.5 Enforcing the evacuation time The evacuation time is included in real-world traffic-lights in order to clear the intersection. This means, before switching the signal state from red to green for a certain lane, all traffic-lights are set to red. For our simple network, we assume the presence of a fixed evacuation time before each switch from red to green.

Technically, the evacuation time is a time interval at the beginning of a red phase during which cars may still remain within the intersection. For the purpose of the implementation used in this thesis, the evacuation time is a time interval at the end of a green phase during which cars may no longer enter the intersection. We assume that the red and green period have already been adjusted accordingly. We require the evacuation time $e \in \mathbb{N}$ to be expressed in integer multiples of the model's timestep. For a given trigger indicator and point in time $t \in \{1, \dots, N\}$, let E_t be the index set of pulse intervals that intersect the time interval between step t and step $t + e$. Furthermore, let S^{start} and S^{end} be the lower and upper bounds of the trigger zone in question.

We assume the presence of a binary indicator χ'_t that accurately indicates entry into the relevant trigger zone. This means, $\chi'_t = 1$ holds iff the car is on the trigger zone. Note that among the trigger formulations presented in Section 4.2.2.2, only the outer-convexification formulation with $\lambda_0 = 0$ and $\lambda_1 = 1$ and the three-way split formulation provide such an indicator. The χ_t^{in} used in the enter-leave formulation can be used to the same effect if we prevent it from assuming the value 1 prior to the car entering the trigger zone by introducing constraints of the following form:

$$\chi_t^{in} \leq \frac{1}{S^{start} - \check{s}} \cdot (s_t - \check{s}) \quad \forall t \in T. \tag{4.51}$$

Given such a binary indicator, the time of entry into the trigger zone is characterized by $\chi'_t - \chi'_{t-1} = 1$. We can therefore prevent a car from entering the intersection during the evacuation time by imposing an upper bound on this difference:

$$\chi'_t - \chi'_{t-1} \leq \chi_i \qquad \forall t \in T, i \in E_t. \tag{4.52}$$

Note that for lane block programs with a minimum red period spanning more than e timesteps, this can be reduced to a single constraint:

$$\chi'_t - \chi'_{t-1} \leq \chi_i, \quad \forall t \in T \text{ and } i = \max E_t. \tag{4.53}$$

This is due to the fact that we can assume that the traffic-light cannot switch to red in the time between timesteps t and $t + e$. Alternatively, we can use the following constraints:

$$\chi'_t - \chi'_{t-1} \leq \frac{1}{|E_t|} \cdot \sum_{i \in E_t} \chi_i \quad \forall t \in T. \tag{4.54}$$

We incorporate these legal regulations for traffic-lights by extending the global-MILP via adding the Constraints (4.46)–(4.52). The resulting MILP is called *global-realistic-MILP*. Measured runtimes of the different formulations (4.52)–(4.54) revealed the best performance for type (4.52). This is why we will work with this kind of constraints. In Section 6, the influence of these regulations is investigated in terms of runtime and achieved objective values of the MILPs.

4.3 Complexity Analysis

We derive statements about the computational complexity of the global-MILP with the aid of results from *scheduling theory*. For this purpose, we give a short overview of basic properties of scheduling problems and notation issues according to [81]. For a good overview over the wide area of scheduling theory, we refer the reader also to [13].

A scheduling problem deals with the allocation of a finite number of resources to a finite number of tasks over a given time period while optimizing one or more objective functions. Often, the resources are *machines* in a particular production-related environment or processing units. The tasks are often referred to as *jobs* and are to be allocated to one or more machines for particular times. Special restrictions or properties of the process can be considered if present. The objective can be manifold. One possibility is the minimization of the completion time of the last task or the sum of the completion times of all tasks. The properties of a scheduling problem are described by a triplet $\alpha \mid \beta \mid \gamma$, where α describes the machine environment, β contains constraints and characteristics of the process and may contain no entry at all or multiple entries. Finally, γ provides the objective function which is to be considered. We present values of the triplet we will need in the course of this section. A possible machine environment is:

Single Machine (1) This is the simplest case of all possible machine environments, where a single machine processes all tasks occurring in the problem one after another.

Job Shop (J_m) In a job shop problem with m machines, each job is processed by multiple machines according to a predetermined individual route.

Possible values for β are:

Release Dates (r_j) The release date denotes the time the job arrives at the system, i. e., that the task j cannot start its processing before its release date r_j.

Processing Times (p_{ij}) The processing time p_{ij} describes the time it takes to process job j on machine i. If only a single machine is present, the index i is omitted. Usually, the processing times are only added to the problem-defining triplet if p_{ij} inherit certain properties, e. g., constant values for all jobs. In case of arbitrary processing times, p_{ij} are not added to the triplet.

Family Dependent Setup Times ($s_{j,k}$) In some scheduling problems, the jobs are grouped into mutually exclusive sets. Each set of jobs is referred to as *family*. If a sequence of jobs on a machine requires a switch from a job in family j to a job in family k, then a setup time of $s_{j,k}$ is incurred. In general, $s_{j,k}$ needs not to be equal to $s_{k,j}$. By definition $s_{k,k}$ equals 0.

Finally, we define a possible value for the objective function in γ which is to be minimized:

Total Completion Time ($\sum CT_j$) For each job j, we define its *completion time* CT_j as the point in time the processing of job j is completed. The total completion time means the sum of the completion times of all jobs in the problem.

In order to get in insight in the complexity of the global-MILP, we first asses that it consists of

$$4 \cdot |C| \cdot |T_N| \cdot |TL| + 9 \cdot |C| \cdot |T_N| + |T_N| \cdot |TL| + 2 \cdot |C|$$

constraints and

$$2 \cdot |C| \cdot |T_N| \cdot |TL| + 3 \cdot |C| \cdot |T_N| + |T_N| \cdot |TL|$$

binary and continuous variables. In Section 6.5, numerical investigation reveals that solving the global-MILP without applying one of the methods described below requires rather high solving times. This is not very surprising if we focus on the involved task of determining an optimal sequence for the cars to pass the intersection only. In fact, this problem can be regarded as a scheduling problem and includes aspects that make the solution of classical scheduling problems difficult. In order to get this insight, we define an optimization problem via:

Definition 4.1. *We are a given a network of intersecting lanes, and a number of cars $c \in C$, which enter the network at the beginning of a particular lane with an individual velocity v_c^{max} at a certain point in time $\bar{t}_c$. Note that the number of intersecting lanes is arbitrary. The aim is to determine an optimal sequence of all cars passing the central intersection. To this end, we define the point in time a car $c \in C$ fully passes the stopping line of the intersection as CT_c. Afterwards it is no longer considered. Formally, the length of the conflict area is set to zero, i. e., $S_{tl}^{Start} = S_{tl}^{End}$ for each traffic light tl. As objective function, the sum of all CT_c is to be minimized. Additionally, overtaking, lane-changes and collisions between succeeding cars are forbidden. Finally, the movement of each car is* *modeled solely by the link of its velocity – which is bounded from below by zero and from above by v_c^{max} – and its resulting position. The problem is set on a discretized and fixed time horizon according to the global-MILP. For the purposes of the following discussion, we call this optimization problem car-scheduling-problem (CSP).*

Note that the CSP is strongly related to the global-MILP for a single intersection. In particular, the ODE-Constraints (4.30)–(4.32) are omitted and the collision-avoidance for succeeding cars, cf. Constraint (4.40), is enforced in front of the traffic-light only. Security gaps between succeeding cars can be regarded as an extended length of the leading car. All lanes are pairwise conflicting in the CSP. The global-MILP's objective function value can be retrieved from the CSP's objective function value by forward simulation of each car's position until the end of the time horizon:

$$s_{c,N} = (T_N - CT_c) \cdot v_c^{max} + S_{tl}^{end} + l_c.$$

With the aid of results from scheduling theory, we can derive the following statement, which is consistent to [37]. There, the authors consider a single intersection and cars, whose passage over the intersection has to be scheduled minimizing the total completion time. The introduced algorithms therein are based on a graph structure and stated to be $\mathcal{NP}$-hard.

Theorem 4.2. *The CSP is $\mathcal{NP}$-hard.*

Proof. The statement is derived by showing that each scheduling problem of type $1 \mid r_j \mid \sum CT_j$, which is known to be $\mathcal{NP}$-hard, cf. [69], can be transformed into an instance of the CSP in polynomial time. To this end, for each job j of the $1 \mid r_j \mid \sum CT_j$-type scheduling problem, a lane in the CSP's road-network is introduced. All of the resulting lanes intersect in a single intersection and are pairwise conflicting. The length of the lanes is set to a fixed positive value, which is chosen equally for all lanes and equals S^{Start}.

Simultaneously, for each job j, a car c is added to the network. The car's properties are derived as described below. First, the time a car enters the network $\bar{t}_c$ is set to zero for each car. The maximum (and initial velocity) for each car is derived from the lane's length, which is without loss of generality set to 1 and the job's first possible processing time r_j:

$$v_c^{max} := \frac{1}{r_j}.$$

We can assume that $r_j > 0$ holds. Otherwise, we define $r'_j := r_j + \epsilon$ for all jobs and a fixed $\epsilon > 0$. Then, r'_j is used for the definition of v_c^{max}. Finally, we set a car's length l_c depending on the corresponding job's processing time p_j via:

$$l_c := p_j \cdot v_c^{max}.$$

Clearly, the single machine in the original scheduling problem can be identified with the CSP's central intersection. The values of the respective objective functions

$\sum CT_j$ (for the $1 \mid r_j \mid \sum CT_j$-type scheduling problem) and CT_c for the CSP are equal due to the CSP's definition. The necessary transformations can be done in polynomial time. Please note that this proof differs from the original version according to comments by the reviewers. □

Remark 4.3. *Theorem 4.2 provides a complexity result for the CSP only. However, we do not have a result for the more complex problem setting modeled by the global-MILP. Besides, it seems reasonable that adding those constraints, which model the longitudinal motion more accurately, and extending the considered problem to the stretch of road behind the intersection, do not facilitate the problem.*

Incorporating conflict zones with a length strictly greater than zero, could be achieved by considering family dependent setup times $s_{i,j}$, which add a non-negative value between the processing of two jobs belonging to different families.

Regarding a road-network, which consists of multiple intersections, we can interpret the CSP as a job-shop problem and derive a $\mathcal{NP}$-completeness result again with complexity hierarchies. For results that allow multiple lanes to be non-conflicting, i. e., for $L > 1$, one has to refer to scheduling problems with parallel machines.

4.4 Digression in Modeling Traffic-Lights

In this section, we have a look at different formulations which aim to tackle the traffic-light functionality. So far, we developed an approach that models the traffic-light mechanism properly but suffers from two major drawbacks:

- Additional binary variables for each triplet of car, traffic-light, and timestep are introduced, i. e., the $\chi_{c,tl,t}^{\{in,out\}}$ variables. These are necessary for recognizing if the car is on the intersection or not.
- The big-M formulations might lead to bad relaxations and therefore to a bad solving behavior, e. g., high runtimes.

A very simple method to get rid of the additional indicator variables $\chi_{c,tl,t}^{\{in,out\}}$ would be to measure the euclidean distance between the car's position and the center of the intersection area. In case that the distance is lower or equal than the width of the intersection area, the traffic-light must be green. We can express this mechanism via the following constraint for each car and traffic-light per timestep:

$$\left(s_{c,t} - \left(S_{tl}^{start} + \frac{S_{tl}^{end} - S_{tl}^{start} + l_c}{2}\right)\right)^2 \geq \left(\frac{S_{tl}^{end} - S_{tl}^{start} + l_c}{2}\right)^2 - M \cdot \chi_{tl,t}. \tag{4.55}$$

The positive constant M has to be chosen sufficiently large. Note that we incorporate the car's length into the constraint as it has to leave the intersection completely before the traffic-light can be set to red. The variable $\chi_{tl,t}$ serves as binary indicator for each traffic-light and timestep as it is the case in the global-MILP. In fact, the Constraints (4.55) would replace the Constraints (4.41)–(4.44) as well as the binary variables $\chi_{c,tl,t}^{\{in,out\}}$. Although the indicator variables $\chi_{tl,t}$ have to be modeled as binary variables, in contrast to the formulation in the global-MILP, the total number of binary variables is much less than in the global-MILP. Nevertheless, introducing (4.55) would transform our currently linear problem to a quadratic non-convex

one. Thus, locally optimal solutions are not necessarily globally optimal anymore contradicting our demand of calculating the best possible traffic flow.
Another approach to reduce the global-MILP's complexity would be to transform the currently time-dependent system states of the ODE (4.1)

$$\frac{\mathrm{d}\boldsymbol{x}}{dt}(t) = \boldsymbol{f}_t\left(t, \boldsymbol{x}(t), \boldsymbol{u}(t)\right) \tag{4.56}$$

into a position-dependent system. According to [53], this can be achieved by taking the inner derivative $\frac{\mathrm{d}t}{\mathrm{d}s}(s) = \frac{1}{v(s)}$. This yields

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}s}(s) = \boldsymbol{f}_s\left(s, \boldsymbol{x}(s) \cdot \frac{1}{\boldsymbol{v}(s)}, \boldsymbol{u}(s)\right). \tag{4.57}$$

Additionally, we introduce the elapsed time t as new state of the ODE system

$$\frac{\mathrm{d}t}{\mathrm{d}s}(s) = \frac{1}{v(s)} \tag{4.58}$$

Assuming that the ODE-system is again discretized using an appropriate collocation method, we obtain variables of type $t_{c,s}$ for each car $c \in C$ and discretized position $s \in S$, where S is an appropriately discretized distance horizon for the problem setting. These variables provide directly the timesteps when a car is on the intersection area. Thus, we do not need any binary indicator variables of type $\chi^{\{in,out\}}_{c,tl,t}$ and associated constraints, which in the global-MILP are (4.41)–(4.44). As it would be rather difficult to keep the global-MILP's concept of binary indicators for each traffic-light and timestep, which model the traffic-light's signal-state, we introduce a precedence-relation for each pair of conflicting cars $c,\ d \in C$. Assuming that c drives towards traffic-light tl_c and d approaches the conflicting traffic-light tl_d, we want one of the following constraints to hold

$$t_{c,S^{start}_{tl_c}} \geq t_{d,S^{end}_{tl_d}+l_d} \qquad \vee \qquad t_{d,S^{start}_{tl_d}} \geq t_{c,S^{end}_{tl_c}+l_c}, \tag{4.59}$$

which means that either c enters the intersection area of the conflicting traffic-lights after d left, or vice versa. This can again be achieved by big-M formulations, cf. Section 2.2.4. In our particular case, this reads as:

$$t_{d,S^{start}_{tl_d}} - t_{c,S^{end}_{tl_c}+l_c} \leq M \cdot \chi_{c,d,tl_c,tl_d} \qquad \forall (c,d) \in C \times C,\ (tl_c, tl_d) \in TL \times TL, \tag{4.60}$$

$$t_{c,S^{start}_{tl_c}} - t_{d,S^{end}_{tl_d}+l_d} \leq M \cdot (1 - \chi_{c,d,tl_c,tl_d}) \qquad \forall (c,d) \in C \times C,\ (tl_c, tl_d) \in TL \times TL, \tag{4.61}$$

where χ_{c,d,tl_c,tl_d} denotes a binary variable for each ordered pair of cars and related traffic-lights $tl_c,\ tl_d \in TL \times TL$. These variables indicate which car passes the intersection consisting of the traffic-lights tl_c, tl_d first. In fact, we only have to consider conflicting cars and conflicting traffic-lights in the Constraints (4.60) and (4.61). In summary, the transformation from a time-dependent system to a position-dependent system leads to an omission of the two binary indicator variables that indicate a car's position relatively to the intersection area of a traffic-light, including necessary constraints. The time-dependent indicator for each traffic-light is superseded by a binary indicator for each ordered pair of conflicting cars. Summing it up, at most $2 \cdot |C| \cdot |TL| \cdot |T|$ binary indicators and at most $|TL| \cdot |T|$ (not necessarily binary) indicators are omitted. In contrast, at most $|C|^2 \cdot |TL|^2$ binary precedence-indicators are introduced. If we consider networks where two different cars share only a single intersection with conflicting traffic-lights (which is the case for a single intersection), we can drop the indices tl_c, tl_d leading to only $|C|^2$ many additional binary variables. For relevant time horizons of several minutes with a fine discretization, e. g., 2500 time steps and 100 cars, this means a dramatic reduction of binary variables in the optimization problem. Moreover, the presence of only a single kind of binary variables facilitates the decision which variable to branch on in a possible branch-and-bound algorithm, cf. Section 2.2.3 and Section 4.5.2. However, the position-dependent formulation brings some disadvantages. First, as no time-dependent state of the traffic-lights is present, legal regulations, cf. Section 4.2.5, are difficult to implement. Further constraints possibly including additional indicator variables would have to be added. Moreover, the resulting program would not be convex anymore.

A third possibility for inducing the mechanism of a traffic-light is as follows: based on the concept of the third trigger-formulation, cf. 4.2.2.2, we introduce two binary variables for each triplet of car, timestep and traffic-light, i. e., $\chi^{pre}_{c,tl,t}$ and $\chi^{post}_{c,tl,t}$. It should hold that $\chi^{pre}_{c,tl,t}$ is equal to 1 if and only if the car is in front of the intersection area, while $\chi^{post}_{c,tl,t}$ is equal to 1 if and only if the car is behind the intersection area. This is enforced by the Constraints (4.22)–(4.24). With a summation term for each car and traffic-light, we can determine the time steps when a car enters the intersection area and when it leaves:

$$\sum_{t \in T} \chi^{pre}_{c,tl,t} = z^{enter}_{c,tl} \qquad \forall c \in C,\ tl \in TL \tag{4.62}$$

$$N - \sum_{t \in T} \chi^{post}_{c,tl,t} = z^{leave}_{c,tl} \qquad \forall c \in C,\ tl \in TL. \tag{4.63}$$

The binary indicator variables $\chi_{c,t}$ in the global-MILP are here dropped in favor of integer variables $z^{\{enter,leave\}}_{c,tl}$ for each pair of car and traffic-light. As we saw in the formulation above, we need a kind of precedence constraint for each pair of conflicting cars at conflicting traffic-lights:

$$z^{enter}_{d,tl_d} - z^{leave}_{c,tl_c} \leq M \cdot \chi_{c,d,tl_c,tl_d} \qquad \forall (c,d) \in C \times C,\ (tl_c, tl_d) \in TL \times TL, \tag{4.64}$$

$$z^{enter}_{c,tl_c} - z^{leave}_{d,tl_d} \leq M \cdot (1 - \chi_{c,d,tl_c,tl_d}) \qquad \forall (c,d) \in C \times C,\ (tl_c, tl_d) \in TL \times TL, \tag{4.65}$$

Note that we do not need to model the z-variables explicitly and include Constraints (4.62)–(4.63). The summation term can be plugged in directly into Constraints

(4.64)–(4.65). We call the MILP consisting of Constraints (4.29)–(4.40) (ODE-constraints and collision-prevention), (4.22)–(4.24) (three-way-split trigger), (4.64)–(4.65), and Objective Function (4.28) *timing-MILP*. In a nutshell, the timing-MILP uses the same amount of binary indicator variables as the trigger formulation in the global-MILP for retrieving a car's position. But instead of (not necessarily binary) indicator variables for each traffic-light and timestep which are at most $|TL| \cdot |T|$ many, binary precedence-indicating variables are introduced for each pair of conflicting cars and their respective traffic-lights. Summing them up, the timing-MILP inherits at most $|C|^2 \cdot |TL|^2$ binary variables of type χ_{c,d,tl_c,tl_d} for pairs of conflicting cars and conflicting traffic-lights. If we consider again networks where two different cars share only a single intersection with conflicting traffic-lights, we can again drop the indices tl_c, tl_d leading to $|C|^2$ many binary variables.

Additionally, the timing-MILP is, in contrast to the ones discussed above, linear. Still, we do not make use of it. Experiments on multiple test scenarios revealed a higher solving time than the global-MILP on the same instances. Reasons for this outcome could be that more binary variables are present than in the global-MILP, even if the total number of variables are lower for certain instances. Also, summation terms as in (4.62) often cause weak relaxations during the solving process.

4.5 Solving Strategies

According to the statements in Section 4.3 concerning complexity and problem size, the global-MILP might be rather intractable for reasonable problem settings. We will investigate this issue in Section 6.5 numerically. At this point, we present different methods to facilitate the solving process for the global(-realistic)-MILP with the aid of methods presented in Section 2.2.

4.5.1 Iterative Solving Algorithm

We seize some of the ideas mentioned above and present an algorithm which calculates exact solutions of the global-MILP and global-realistic-MILP. Basically, smaller MILPs are solved iteratively as long as violated constraints are identified and added to the model. Moreover, variables are left out in the beginning and successively included later.

First, we exploit the fact that a car that has passed the intersection and reached its maximum velocity as dictated by vehicle-specific limits, or the speed of a car directly in front of it with lower maximum velocity, and is no longer accelerating, will achieve its maximum contribution to the objective function (4.26) by maintaining its velocity for the remainder of the time interval. This means that cars that reach this *steady state* can be removed from the model early. Since this only changes the value of the objective function (4.26) by a constant, the solution is not affected.

Additionally, we see that any given car's behavior will not be affected by a traffic-light's trigger for most of the time interval. This applies specifically to cars that have already passed the traffic-light, cars that have yet to reach the traffic-light and cars that encounter no conflicting car while passing the corresponding trigger zone. For such cars, the introduction of trigger-related model structures can be avoided entirely. To omit these superfluous model elements, we follow an iterative approach in which the model is solved repeatedly and additional variables and constraints are added only to improve solution feasibility with respect to the

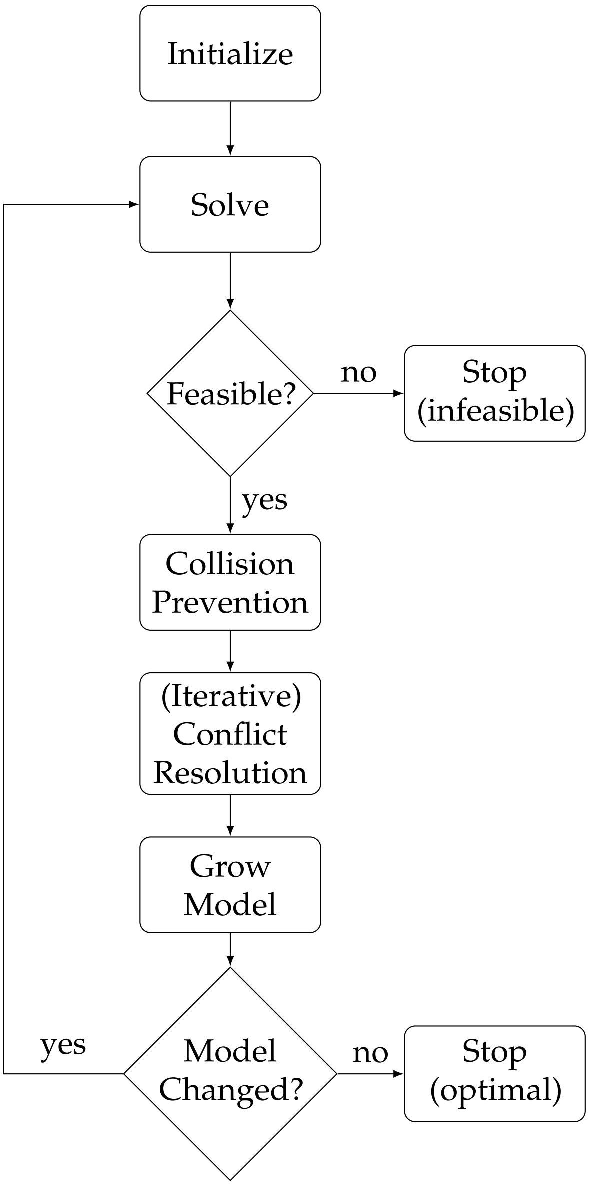


Figure 4.10: Flow chart of the iterative solving algorithm.

complete model after each iteration. Figure 4.10 depicts a flow chart detailing the steps of the iterative algorithm. In Section 6, we discuss the complexity reduction that can be achieved using this approach, as well as resulting effects on solving times.

Let us refine the ideas above and develop concrete methods. In the first step, the model is *initialized* by introducing Constraints (4.29)–(4.39) and associated state variables for an appropriate time frame for each car. The time frame is chosen by estimating the amount of time the car will need to reach the intersection, pass it and clear a given stretch of road behind the intersection, if left entirely unobstructed. No trigger-related variables or constraints are introduced during initialization. Hopefully, this method reduces the complexity of the MILP drastically.

The greedy-algorithm, which is discussed in Section 5, may be used to improve the estimates on each car's initial time frame and introduce some of the required trigger-related structures. To this end, the greedy-solution is used as an MIP start in the first iteration of the subsequent refinement loop.

Algorithm 4.1: Pseudocode of the iterative solving algorithm.

```
initialize MILP with Constraints (4.29)–(4.39) and associated variables for
  each car and those time steps which are at least necessary for the car to pass
  the intersection with maximum velocity;
while MILP ≠ global-MILP do
    solve MILP;
    if MILP infeasible then
        return problem infeasible;
    else
        for each car c in C do
            check for collisions with preceding car. Add Constraints (4.40) if
              check failed;
            check for conflicts with cars on the intersection by (iterative)
              conflict-resolution. Add Constraints (4.41)–(4.45) and associated
              variables if check failed;
            check if c crossed the intersection and reached a steady state. Add
              Constraints (4.29)–(4.39) for further time steps if check failed;
        end
    end
    if all checks passed then
        return solution of the global-MILP. To this end, extrapolate the
          motion of cars into the future;
    end
end
solve global-MILP;
return solution of the global-MILP or assess infeasibilty;
```

In the next step, the resulting problem is solved. Afterwards, it is analyzed whether the resulting solution is also feasible with respect to the global-MILP. Initially, collisions between succeeding cars driving on the same lane are identified. If these are detected, constraints of the form (4.40) are introduced as needed. We refer to this step as *collision-prevention*, which is in fact a cutting-plane method, cf. Section 2.2.1.

Afterwards, we check for conflicts, which occur on the intersection area of the traffic-lights. Using the MILPs terminology, we look for cars on conflicting lanes, which are on conflicting trigger zones for a timestep $t \in T$. The so-called *conflict-resolution* identifies these conflicts and adds the trigger-related constraints and variables of the conflicting vehicles for the time span in question. Particularly, Constraints (4.41)–(4.45) are added to the model along with necessary χ-type variables. This step consists of adding cutting planes and realizes a column-generation method, cf. Section 2.2.2. Note that no pricing problem is solved here to identify variables which are to be added to the problem.

Conflict-resolution will likely cause one of the conflicting cars to be delayed slightly to avoid the conflict. In dense traffic, one can easily imagine the passing car being closely followed by another car which would then come into conflict with the crossing car in the next iteration. In such situations, the conflict-resolution may require a high number of solver iterations. To avoid this becoming a problem, the *iterative conflict-resolution* is introduced. Thereby, we attempt to resolve conflicts that arise as a result of newly introduced trigger infrastructure without performing another full solver iteration. More precisely, a solver loop within the iterative conflict-resolution is started in which every car's trajectory is fixed, except for the vehicles that were in conflict and their followers. Conflicts within the internal solver loop are resolved using the conflict-resolution. This way, at least the conflicts of the conflicting cars are resolved in each step. When a solution is found which does not include any further conflicts, the internal solver loop is stopped and the crucial trigger-related variables and constraints of this solution are added to the model. Additionally, the last obtained solution can serve as MIP-start for the solve-step.

After (iterative) conflict-resolution, the algorithm checks whether all cars crossed the intersection and reached a steady state. If this is not the case for car c, the *model-growth* takes its last known state and uses a simple forward simulation with constant maximum jerk until $a_{c,t} = a_c^{max}$ and constant maximum acceleration until $v_{c,t} = v_c^{max}$. Thus, the existence of the car is extended by introducing additional Constraints (4.29)–(4.39) and associated state variables for further time steps as needed. This way, the model-growth heuristically determines when the car will have reached a steady state. Once this is the case for a car in an obtained solution, the car is removed from the model for the rest of the time horizon T.

These steps are repeated until no introduction of further variables or constraints are required. By extrapolating the steady state of all cars into the future, an optimal solution of the global-MILP can be obtained. Note that this is only a valid method if the maximum velocity of each car is not strictly greater than the maximum velocity of its predecessor, which makes the collision-avoidance-constraint (4.40) redundant once the steady state of each car is reached. In Algorithm 4.1, the iterative solving algorithm is given in pseudocode.

Considering realistic traffic-lights, cf. Section 4.2.5, with the required constraints and variables, we have to take some additional effects into account for the conflict-resolution step. It becomes necessary to introduce trigger-related constraints and variables if a car passes the intersection area even without being in conflict with another car. This behavior is essential to the proper enforcement of traffic-light programs as traffic-light rhythms can be affected by a car being on the intersection area without encountering a crossing vehicle. These constraints and variables are added during (iterative) conflict-resolution.

Theorem 4.4. *The iterative solving algorithm terminates in finitely many steps. In the worst case,* $3 \cdot |C| \cdot |T|$ *MILPs have to be solved.*

Proof. For counting the number of MILPs to be solved, we refer to Algorithm 4.1. In the worst case, only a single check for a single car fails after an MILP has been solved. This triggers another solution process after the MILP has been slightly adapted. As each kind of check can fail at most $|T|$ times for a car, the stated number of iterations follows. □

4.5.2 A Tailored Branch-and-Bound algorithm

As an alternative approach for improving the solving process of the global-MILP, we present a standalone branch-and-bound algorithm, which exploits the structure of the global-MILP. Its performance will be analyzed later, in Section 6.5.3. This branch-and-bound algorithm is designed for networks which involve a single intersection only. Hence, indices for the traffic-light are omitted in the discussion. Nonetheless, the routine can be adapted for multi-intersection networks. For a discussion on branch-and-bound algorithms, cf. Section 2.2.3.

In the algorithm which we develop in this section, we branch on the trigger-variables of type $\chi^{in}_{c,t}$ and $\chi^{out}_{c,t}$ for a car c and time-step t in increasing order of the time index. Simply put, for every time-step t and car c, four different nodes are generated, according to the possible values and combinations of the binary variables. In each node of the tree, a linear program is solved, while those binary trigger-variables are relaxed which have not been fixed yet. Moreover, these linear programs do not cover the whole time grid $T = \{0, \dots, N\}$, but only the first part of the grid $\{0, \dots, \tilde{t}\}$ for a particular $\tilde{t} \in T$. In a sense, we follow the movement of the cars over the course of time. Once an infeasibility is detected, the whole branch can be pruned.

Algorithm 4.2 depicts an outline of the branch-and-bound algorithm. For convenience, we define t^{inter}_c as the lowest time step in which a car c is able to reach the intersection without entering it. During the branch-and-bound algorithm, each node stores information about its current depth d in the tree, the currently considered car c, the current time step t, the upper bound u provided by the current solution, and the set of already fixed binary variables fix in a tuple of type (d, c, t, u, fix). Before we discuss the branching step in detail, we have a look at the bounding procedure.

Bounding Step As we solve LPs in each node which cover only a part of the whole time horizon, we have to make sure to retrieve upper bounds on the objective value as we are maximizing the objective function. Suppose that we calculated a solution x^* in the particular node with an objective value $o_{\tilde{t}}(x^*)$ of the smaller LP covering the horizon $[0, \dots, \tilde{t}]$. Generally, the objective function reads as:

$$o_i(x^*) := \sum_{c \in C} s_{c,i}$$

for any $i \in T$. Obviously, $o_N(\cdot)$ is the global-MILP's objective function. The upper bound $u(x^*)$ on $o_N(\cdot)$ in the whole branch is obtained via forward simulation of the movements of all cars with maximum velocity v^{max}_c:

$$u(x^*) := o_{\tilde{t}}(x^*) + \sum_{c \in C} (N - \tilde{t}) \cdot v^{max}_c \cdot dt.$$

We can see that $u(x^*)$ is a valid upper bound by assuming that a solution $\bar{x}_N$ for the whole MILP exists with an objective value $o_N(\bar{x}_N) > u(x^*)$. For the distances of the vehicles in time-step $\tilde{t}$ in $\bar{x}_N$ one has $o_{\tilde{t}}(\bar{x}_N) > o_{\tilde{t}}(x^*)$ due to the construction of $u(x^*)$. This is a contradiction to x^* being an optimal solution.

A more sophisticated approach for determining an upper bound considers the distance of each car to its predecessor, as collisions are prohibited in the MILP. To this end, we have to determine the extrapolated position $\hat{s}_{c,t}$ of each car for each

Algorithm 4.2: Pseudocode of the tailored branch-and-bound algorithm.

```
initialize tree $Q$ with the root node $\left(0, c, t_c^{inter}, \infty, \{\}\right)$ for a car $c$ with a minimal
  value for $t_c^{inter}$;
set global lower bound $l^* := -\infty$ ;
while $Q \neq \emptyset$ do
    select current node $a$ from $Q$ due to search strategy and remove it from $Q$;
    if $a.u < l^*$ then
        prune a;
        continue;
    end
    solve LP with $a.fix$ on the grid $\{0, \ldots, a.t\}$;
    if infeasible then
        return infeasible;
    end
    perform bounding step and update $a.u$;
    if $a.u < l^*$ then
        prune a;
        continue;
    end
    if $a.t = N$ and $\chi_{c,N}^{\{in,out\}} \in a.fix$ for all $c \in C$ then
        if $objective(a.x^*) > l^*$ then
            set current optimal solution $x^* \leftarrow a.x^*$;
            set global lower bound $l^* \leftarrow objective(a.x^*)$;
        end
    else
        if $a.t = N$ then
            create single child node $b$;
            set $b.d \leftarrow a.d + 1$;
            set $b.fix \leftarrow a.fix$;
            set $b.u \leftarrow a.u$;
            set $b.c$ to a car $c$ with $\chi_{c,N}^{\{in,out\}} \notin b.fix$ and minimum value of $t_c^{inter}$;
            set $b.t \leftarrow t_{b.c}^{inter}$;
            add $b$ to $Q$;
        else
            create four child nodes according to possible permutations of
              $\{0, 1\}^2$ which are the values of $\chi^{\{in,out\}}$;
            for each child $b$ do
                set $b.d \leftarrow a.d + 1$;
                set $b.u \leftarrow a.u$;
                set $b.c \leftarrow a.c$;
                set $b.t \leftarrow a.t + 1$;
                set $b.fix \leftarrow a.fix$ plus permutations of values in $\chi_{b.c,a.t+1}^{\{in,out\}}$;
                add $b$ to $Q$;
            end
        end
    end
end
return optimal solution $x^*$;
```

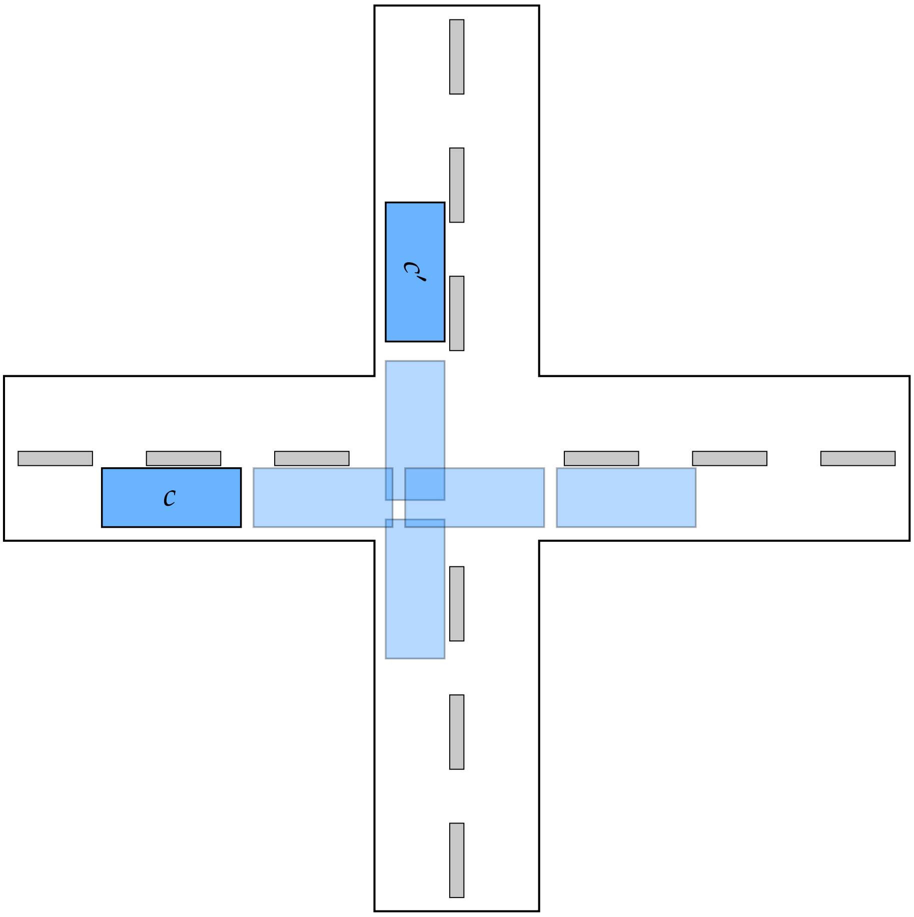


Figure 4.11: Current position and extrapolation of future positions of cars c and c'. The extrapolations (light blue rectangles) predict conflicts on the intersection in two time steps. Thus, the upper bound on the objective value is reduced by $2 \cdot min\left(v_c^{max}, v_{c'}^{max}\right) \cdot dt$.

time step. In this way, we obtain an extrapolated value which can be compared to the predecessor's position. The particular upper bound for each node reads as

$$u(x^*) := o_{\tilde{t}}(x^*) + \sum_{\substack{c \in C, \\ t \in \{\tilde{t}, \ldots, N\}}} \hat{s}_{c,t}.$$

The values for $\hat{s}_{c,t}$ are defined via

$$\begin{aligned} &\hat{s}_{c,\tilde{t}} := s_{c,\tilde{t}}, \\ &\hat{s}_{c,t} = min\left(\hat{s}_{c,t-1} + v_c^{max} \cdot dt,\ \hat{s}_{pred(c),t} - l_{pred(c)} - g_c\right) \quad \forall t \in \{\tilde{t}+1, \ldots, N\}. \end{aligned}$$

Bounding Heuristic An important effect on the quality of the upper bound is neglected in the presented branching step: no conflicts of cars on the intersection in future time steps are considered. That is, the position of all cars which are in front

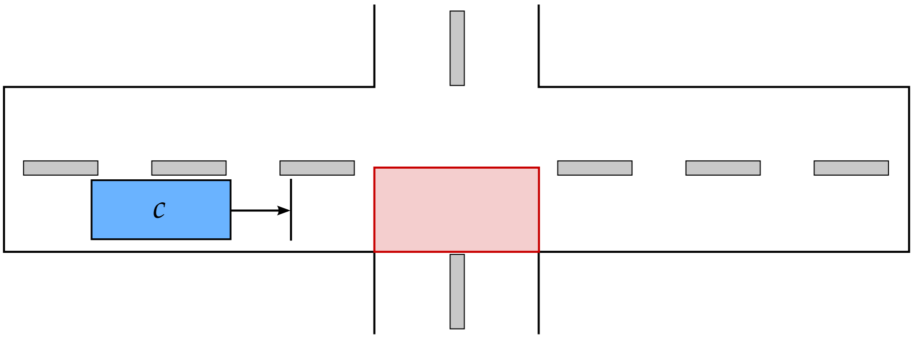


Figure 4.12: Car c on the network with position in time-step t. The vertical bar indicates the position the car can at most achieve in the next time-step. In this case, it cannot enter the intersection.

of the intersection in the current time step is simply extrapolated without taking possible conflicts on the intersection into account. A heuristical approach, which turned out to provide very good bounds in runtime experiments, incorporates conflicts as follows: we count the number of time steps $t > \tilde{t}$ in which multiple conflicting cars are on the intersection according to the extrapolated distances $\hat{s}_{c,t}$. Remember that we received these distances by extrapolating each car's distance with maximum velocity. Each of these *conflict-times* means that at least one of the conflicting cars cannot be on the intersection during this time step. This results in a lower value of the upper bound in this node. Following this consideration, we subtract $v_c^{max} \cdot dt$ from the upper bound $u(x^*)$ for each conflict-time, where c is the particular car with lowest maximum velocity participating in this conflict-time. In case that four cars (one on each lane) are detected to be in a conflict, two of them are not allowed to be on the intersection. Thus, we can increase the sum of all conflict-times by two. Figure 4.11 depicts this situation for two conflicting cars.

Branching Step Creating four nodes for each car and timestep would lead to a relatively large number of LPs which have to be solved in the resulting nodes. Therefore, we exploit the structure of the model and omit certain kinds of nodes as explained below. First, let us recall Table 4.1 which lists possible values for the trigger-variables depending on a car's current position. Thus, in the root node, we can fix the values of $\chi_{c,t}^{in}$ to 0 for each car c and time-step t in which the particular car is not able to reach the intersection even by driving with maximum velocity. Analogously, we fix $\chi_{c,t}^{out}$ to 1 for each car and time-step in which the particular car cannot manage to pass the intersection even by driving with maximum velocity. Additionally, we distinguish the following situations to identify negligible nodes for each car c:

(i) If c is in front of the intersection and not able to enter the intersection in the next time-step even by driving with maximum velocity, we can fix $\chi_{c,t}^{in}$ to 0 and $\chi_{c,t}^{out}$ to 1 for all time-steps t in which c is not able to enter the intersection, even by driving with maximum velocity. Remember that the current solution provides a maximum value for the car's covered distance in this time-step as we are restricted to networks with a single intersection. Figure 4.12 visualizes the described setting.

(ii) If c is in front of the intersection and able to enter in the next time-step by driving with maximum velocity, cf. Figure 4.13, we generate two nodes

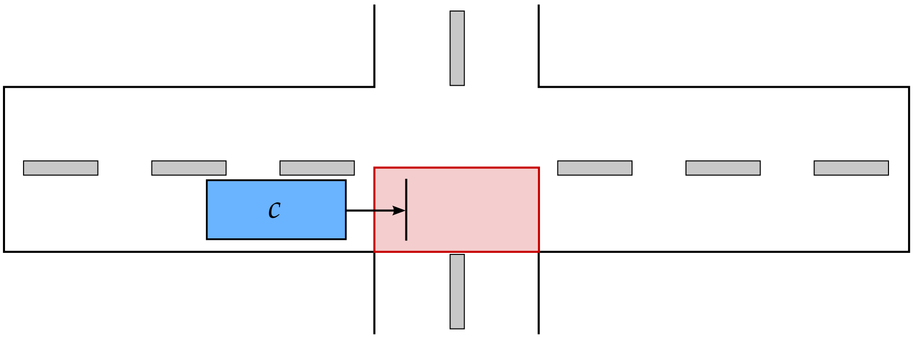


Figure 4.13: Car c on the network with position in time-step t. The vertical bar indicates the position the car can at most achieve in the next time-step. In this case, it is able to enter the intersection. We generate an additional node, where c does not enter the intersection if a conflicting car has the chance to enter before c will have left.

with $\chi^{in}_{c,t+1} = 1$, $\chi^{out}_{c,t+1} = 1$ and $\chi^{in}_{c,t+1} = 0$, $\chi^{out}_{c,t+1} = 1$. This can be interpreted as providing the possibility to enter the intersection in the following time-step or to stay in front of the intersection, e. g., to allow a blocking car to enter it. In case that c enters the intersection, we can bound the amount of time-steps it will be on the intersection from below by assuming that it drives with maximum velocity. Thus, we can generate nodes with $\chi^{in}_{c,t'} = 1$, $\chi^{out}_{c,t'} = 1$ for all $t' > t$ in which c cannot manage to leave the intersection.
In case that a blocking car c' is currently on the intersection, cf. Figure 4.14, we do not provide the possibility for c to enter. More specifically, c' has to be on the intersection and is not able to leave it in the next time-step by driving with maximum velocity. As we are maximizing the covered distance, c' will not slow down in the next timestep, once it is on the intersection. This means, we only generate the node with $\chi^{in}_{c,t+1} = 0$, $\chi^{out}_{c,t+1} = 1$ in this case. Note that we do not cut off valid solutions as we already generated nodes allowing c to enter the intersection before c' because this case already applied for c'. Analogously to the argumentation above, we can generate nodes for all $t' > t$ in which c' is not able to leave the intersection with $\chi^{in}_{c,t'} = 0$, $\chi^{out}_{c,t'} = 1$.

(iii) If c is currently on the intersection and is not able to leave it in the next time-step by driving with maximum velocity, we only generate nodes with $\chi^{in}_{c,t+1} = 1$, $\chi^{out}_{c,t+1} = 1$. We again generate similar nodes for all $t' > t$ in which c cannot manage to leave the intersection. Figure 4.15 visualizes the situation occurring here.

(iv) The last case occurs when c is either already behind the intersection or still on it, but able to leave it in the next time-step by driving with its current velocity. In this case, we generate nodes for all $t' \in [t+1, \ldots, N]$ with $\chi^{in}_{c,t'} = 1$, $\chi^{out}_{c,t'} = 0$. We do not cut off valid solutions here as once the car left the intersection, it is not able to enter it again. Also, it will not slow down, once it entered the intersection because of the distance, which is to be maximized. Thus, we at most cut off non-optimal solutions. In Figure 4.16, the described setting can be seen.

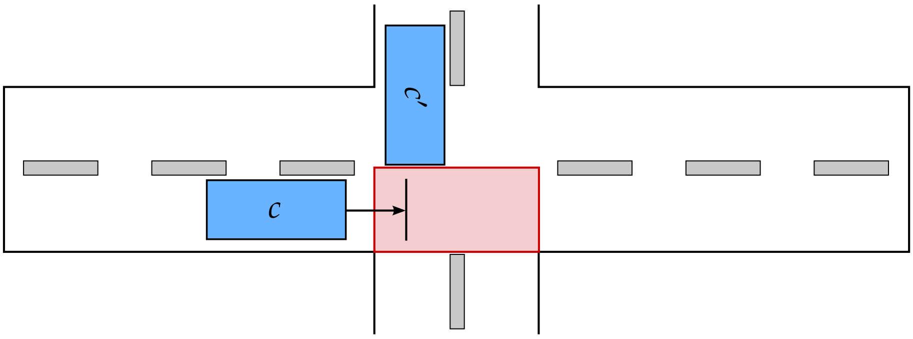


Figure 4.14: Car c on the network with position in time-step t. The vertical bar indicates the position the car can at most achieve in the next time-step. In this case, it is able to enter the intersection. A blocking car c' on the intersection prohibits c to enter.

Bounding Alternative As an alternative for solving an LP in each step which covers only a certain part of the time grid, one could also solve an LP for the whole time horizon. Such an LP would again be a relaxation of the global-MILP in the variables which are to be branched, with all branching-variables fixed up to the current time step. The bound in each node would simply be the objective value of the LP's optimal solution in this node. In Section 6.5.3, we compare both approaches in terms of different parameters. We focus on the quality of the bounds yielded by each method as well as the overall solving times and the solving times for each node.

Search Strategy The strategy for selecting the next node is realized via depth-first-search. This means, the node in Q with the highest depth d is chosen. If multiple nodes with the same depth exist, the node with the highest local upper bound u is chosen. As soon as a new integral solution is obtained, the node which provides the highest upper bound is selected. Thus, we realize a diving-method here, cf. Section 2.2.3

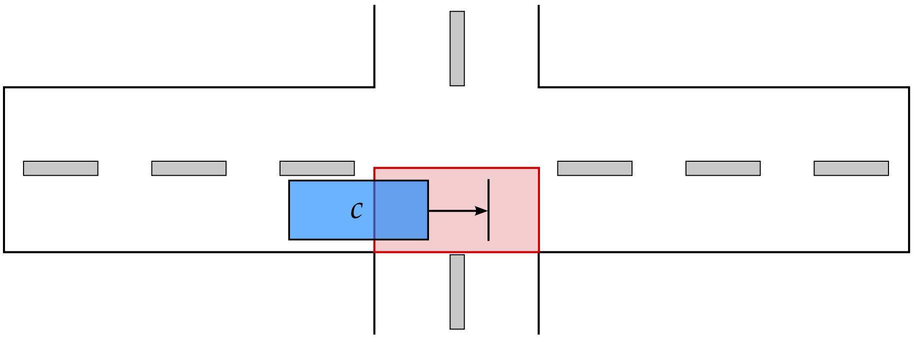


Figure 4.15: Car c on the network with position in time-step t. The vertical bar indicates the position the car can at most achieve in the next time-step. In this case, it is not able to leave the intersection.

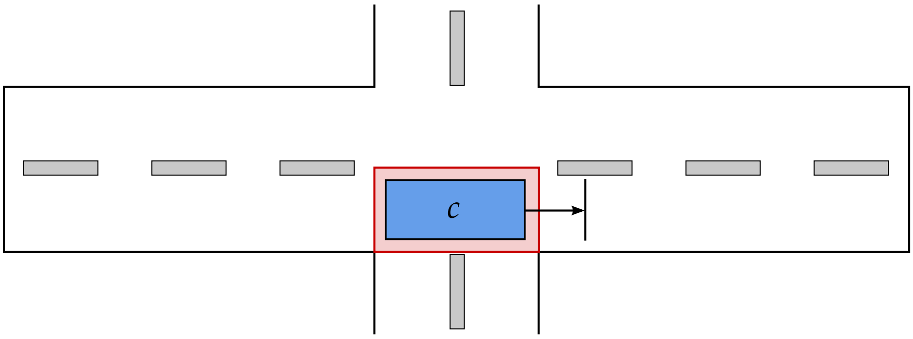


Figure 4.16: Car c on the network with position in time-step t. The vertical bar indicates the position the car can achieve in the next time-step with its current velocity. In this case, it either already left the intersection or is able to leave in the next time-step.

In the next section, we go back a step from an overall optimized traffic flow and present an algorithm which calculates trajectories for each car individually. The reduction in complexity that we achieve can hopefully be used to design a system for a real-world application.

As we will see in Section 6.5, the solving times for the global-MILP are quite high – at least with respect to practical purposes. Thus, an application which is based on the global-MILP is difficult to implement. Furthermore, the global-MILP determines an offline calculated behavior, which is rather sensitive considering a not exact performance of the calculated trajectories by the cars. This would result in collisions between succeeding cars and on the intersection. The experiences we made while testing the RACC on the road suggest that it is nearly impossible to perform these trajectories sufficiently well. Thus, it would be necessary to recalculate the global-MILP repeatedly, which is again inhibited by the high solving times. Finally, the model does not consider cars which are not connected to the system that calculates the solutions. Nevertheless, the solutions of the global-MILP can serve as a benchmark for the algorithm to be developed in the following section, and, of course, other systems and strategies which aim to improve traffic flow at traffic-light controlled intersections.

5 An Individual Approach to Optimizing Traffic Flow

In the previous section, we developed an MILP which includes an objective to achieve a globally optimal traffic flow by calculating states for cars and traffic-lights in a network of intersecting roads. The solutions of this MILP serve as benchmarks for applications which generate suboptimal but practically implementable solutions.

In this section, we discuss an algorithm which aims to optimize the behavior of every car $c \in C$ individually. Other cars are only considered to prevent collisions of succeeding cars on a lane and conflicting cars on the intersection. In the literature, a variety of publications consider GLOSA-methods, cf. Section 3. This approach calculates static velocities for each car to pass the intersection while preceding cars are neglected. Often, these velocities are displayed as advice to the driver and are not performed automatically. In [19], the authors present a system, which dynamically calculates times for each car to pass the intersection and non-static velocities in order to increase traffic flow. It is refined and extended by human drivers in [20, 21, 103]. The advantage of our method is that we calculate trajectories for every car, whereas the aforementioned system only switches between constant, maximum, and minimum acceleration in each step. This approach not only leads to a suboptimal behavior of the cars, the resulting trajectories should be rather uncomfortable for the passengers. Thus, we incorporate realistic motion dynamics and bounds on the physical quantities of the model. Another approach presented in [97], is based on time slots, which are dynamically assigned to vehicles and allow them to pass the intersection. Besides the fact that this approach is somehow related to the algorithm developed in the course of this section, we propose a method, which is more dynamic as no slots of fixed time exist. Again, our approach includes concrete acceleration-trajectories in contrast to the mentioned method. Finally, the authors of [107] present an algorithm similar to our approach for trains moving one after another on a single-track network. Thereby, two trains are forbidden to be in the same segment simultaneously. They also compare different solving methods, including MILPs, regarding runtimes and objectives, as we do in this thesis.

In this section, we first present the algorithm. Afterwards, possible adaptions and extensions are discussed. Finally, we propose a real-world system based on the algorithm.

5.1 Algorithm

In order to make the solutions of the individual optimization comparable to the solutions of the global-MILP, the algorithm is based on the longitudinal motion model and trigger mechanism of the previous section. Also the discretized timehorizon $T = \{0, \ldots, N\}$ is kept. In contrast to the global-MILP, the variables for the cars and traffic-lights are not calculated as a whole. In contrast, smaller MILPs for each car $c \in C$ individually are solved iteratively.

The passage of the cars over the intersection is based on a first-come, first-served principle. Again, we want to maximize the covered distance for each car. To this end, the proposed algorithm, which is visualized in Figure 5.1, performs the following steps for each point in time $t \in T$. An ordered list S handles the cars which are to be optimized. It is initialized with the empty set. Prior to the first optimization, the algorithm's current time step t is increased until the set $\hat{C}_t := \{c \in C | \bar{t}_c = t\}$ containing all cars that enter the network in the current time step is not empty and appended to the queue S. As long as S is not empty, the following steps are performed iteratively. First, an MILP for the first member of S called $\bar{c}$ is solved. In most cases, a car which enters the network in the current time step, i. e., $\bar{t}_{\bar{c}} = t$, is considered. For simplicity, we omit the index $\bar{c}$ for the current car and regard the MILP given by:

$$
\begin{aligned}
& \max_{a,v,s,\chi^{in},\chi^{out},\chi} && \sum_{t \in T} s_t && && (5.1)\\
& \text{s.t.:} && s_{t+1} = s_t + v_t \cdot dt && \forall t \in T \setminus \{N\}, && (5.2)\\
& && v_{t+1} = v_t + a_t \cdot dt && \forall t \in T \setminus \{N\}, && (5.3)\\
& && \frac{1}{dt} \cdot (a_{t+1} - a_t) \geq j^{min} && \forall t \in T \setminus \{N\}, && (5.4)\\
& && \frac{1}{dt} \cdot (a_{t+1} - a_t) \leq j^{max} && \forall t \in T \setminus \{N\}, && (5.5)\\
& && s_t = 0 && \forall t \in \{0, \dots, \bar{t}_c\}, && (5.6)\\
& && v_t = \bar{v} && \forall t \in \{0, \dots, \bar{t}_c\}, && (5.7)\\
& && a_t = 0 && \forall t \in \{0, \dots, \bar{t}_c\}, && (5.8)\\
& && v^{min} \leq v_t && \forall t \in T, && (5.9)\\
& && v_t \leq v^{max} && \forall t \in T, && (5.10)\\
& && a^{min} \leq a_t && \forall t \in T, && (5.11)\\
& && a_t \leq a^{max} && \forall t \in T, && (5.12)\\
& && s_{pred(c),t} - s_t \geq l_{pred(c)} + g && \forall t \in T, && (5.13)\\
& && \chi^{in}_{tl,t} + \chi^{out}_{tl,t} - \chi_{tl,t} \leq 1 && \forall tl \in TL_c,\ t \in T, && (5.14)\\
& && \chi^{in}_{tl,t} + \chi^{out}_{tl,t} \geq 1 && \forall tl \in TL_c,\ t \in T, && (5.15)\\
& && \left(S^{end}_{tl} - \check{s}\right) \cdot \chi^{out}_{tl,t} + s_t \geq S^{end}_{tl} && \forall tl \in TL_c,\ t \in T, && (5.16)\\
& && \left(S^{start}_{tl} - \hat{s}\right) \cdot \chi^{in}_{tl,t} + s_t \leq S^{start}_{tl} && \forall tl \in TL_c,\ t \in T. && (5.17)
\end{aligned}
$$

The slightly modified objective function compared to the global-MILP is explained below. Note that in Constraint (5.13), the value of the predecessor's currently covered distance $s_{pred(c),t}$ is fixed since the optimal solution for the vehicle's predecessor is already determined, because it enters the network earlier. Remember that no overtaking maneuvers are allowed. In the first step, each vehicle schedules a preferred transit through the network individually, only regarding preceding vehicles in order to avoid collisions. The constraint which couples conflicting traffic-lights

$$
\sum_{tl \in TL_l} \chi_{tl,t} \leq 1 \quad \forall t \in T,\ l \in \{1, \dots, L\} \tag{5.18}
$$

is handled outside the MILP. In case that any violation of this constraint is detected, the MILP is reoptimized while fixing some of the values for $\chi_{tl,t}$ according to the following rules.

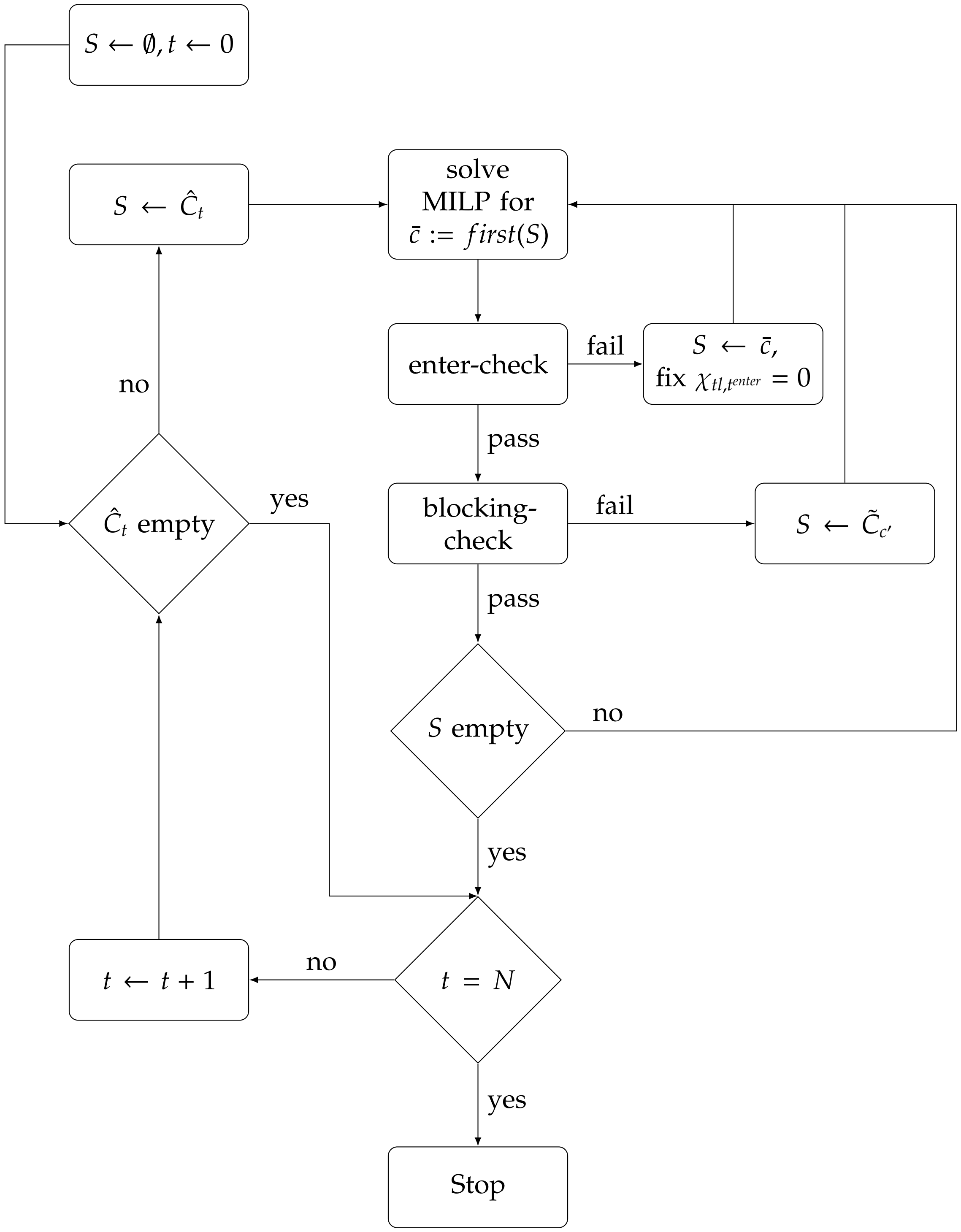

$S \leftarrow \emptyset, t \leftarrow 0$
$S \leftarrow \hat{C}_t$
solve MILP for $\bar{c} := first(S)$
enter-check
fail
$S \leftarrow \bar{c}$, fix $\chi_{tl,t^{enter}} = 0$
pass
no
$\hat{C}_t$ empty
yes
blocking-check
fail
$S \leftarrow \tilde{C}_{c'}$
pass
S empty
no
yes
$t \leftarrow t + 1$
no
$t = N$
yes
Stop

Figure 5.1: Flow chart of the iterative algorithm.

Thus, in the next step, called *enter-check*, the car's scheduled passage through the trigger zone is analyzed: if no conflict with conflicting cars occurs, the car's variables are fixed. We also call the particular car to be fixed. To this end, we have a look at the particular time step $t^{enter} \in T$ in which the car planned to enter the trigger zone of a traffic-light $tl \in TL$: if a violation of Constraint (5.18) occurs – due to a conflicting car being on the intersection area – the MILP is resolved for $\bar{c}$ while fixing $\chi_{tl,t^{enter}}$ to 0. In other words, if another car on a conflicting road already scheduled to be on the intersection, the passage of $\bar{c}$ is rescheduled while respecting the other car. We can do even better and fix $\chi_{tl,t}$ to 0 for all following time steps $t > t^{enter}$ until the conflicting car scheduled to leave the trigger zone. Currently, $\chi_{tl,t} = 1$ is allowed, even if no car is in the trigger zone of tl. We have to modify the solution after the optimization such that $\chi_{tl,t} = 1$ holds if and only if a car is on the traffic-light's trigger zone. Otherwise, conflicts may be detected falsely. This does not have any influence on the value of the objective function.

In contrast, if no violation of Constraint (5.18) for t^{enter} occurs, it is checked if there are no conflicts in succeeding time steps. In other words, the passage of an already fixed car c' on a conflicting lane has to be rescheduled if it planned to enter the trigger zone after the current car $\bar{c}$. To this end, it is checked if there is a $\hat{t} \in \{t^{enter} + 1, ..., t^{leave} - 1\}$ with $\chi_{tl',\hat{t}} = 1$ for a conflicting traffic-light $tl' \in TL$, while t^{leave} is the time $\bar{c}$ leaves the trigger zone. If this is the case, the MILPs for the car c' that induces $\chi_{tl',\hat{t}} = 1$ and all its already fixed successors are reoptimized. For this purpose, c' and its successors ordered according to their respective arrival-times are added to the set $\tilde{C}_{c'}$ which is prepended to S. This step is called *blocking-check*.

As soon as S is empty, t is increased until $\hat{C}_t$ is nonempty, or the end of the time horizon is reached, i. e., $t = N$, which indicates the algorithm's termination.

Due to the fact that the cars do not consider their successors, they behave in a rather uncooperative way in front of the traffic-light. This means, that if a car has to decelerate in front of the traffic-light, because of the passage of conflicting cars, it happens that it may stop in the very beginning of the network. Consequently, succeeding cars may not be able to avoid a collision, due to bounds on the acceleration and jerk. Therefore, we consider not only the covered distance in the final timestep, but also in the prior steps. The resulting objective function for each car reads as:

$$\max \quad \sum_{t \in T} s_t. \tag{5.19}$$

We refer to this algorithm, because of its behavior of locally optimizing the trajectories, as *greedy-algorithm*. A description in pseudocode is given in Algorithm 5.1. In Section 6.6, we investigate the performance of the greedy-algorithm and compare it to the global-MILP and the RACC.

Theorem 5.1. *The greedy-algorithm terminates after finitely many iterations. In the worst case, the number of iterations is exponential in the number of cars.*

Proof. In order to see that the first statement holds, we consider for Algorithm 5.1 that for each pair of cars c, d, where c caused a failed blocking-check for d, d cannot cause a failed blocking-check for c in the same inner iteration. Thus, no infinite loops occur. Additionally, for each car at most $|C| - 1$ many enter-checks can fail per while-iteration.

Algorithm 5.1: Pseudocode of the greedy-algorithm.

```
initialize with empty queue $S := \emptyset$ and $t := 0$. Set all cars to be not fixed;
while $S = \emptyset$ do
    append all cars $c$ to queue $S$ with $\bar{t}_c = t$;
    $t := t + 1$
end
while $t \neq N$ do
    repeat
        remove first element $\bar{c}$ of queue $S$;
        if $\bar{c}$ is not fixed then
            fix current car $\bar{c}$;
            initialize MILP for $\bar{c}$;
        end
        solve MILP;
        if enter-check failed then
            fix $\chi_{tl,t^{enter}}$ (and possibly for subsequent time steps) to zero;
            insert $\bar{c}$ at the beginning of $S$;
            continue;
        end
        if blocking-check failed then
            prepend blocking cars and their successors to $S$ and unfix them;
            continue;
        end
    until $S = \emptyset$;
    while $t \neq N$ do
        if $\exists\, c \in C$ with $\bar{t}_c = t$ then
            append $c$ to queue $S$;
            break;
        end
        increment current time step $t := t + 1$;
    end
end
```

The exponential number of iterations is due to the blocking-checks. Let $N_{|C|}$ be the number of iterations which are to be solved in the worst case for $|C|$ many cars, when regarding solely blocking-checks. Clearly, if $|C| = 1$, then $N_{|C|} = 1$ as no failed checks can occur. It then holds that $N_{|C|} = 2 \cdot N_{|C|-1} + 1$. This can be seen by following inductive argumentation: increasing the number of cars from $|C|$ to $|C| + 1$, triggers in the worst case all iterations, which were necessary for $|C|$ many cars ($N_{|C|}$). Additionally, another iteration for the new car is necessary ($N_{|C|} + 1$). In the worst case, this new car triggers a reoptimization of all other cars again, due to a failed blocking-check. Hence, the number of iterations sums up to $2 \cdot N_{|C|} + 1$.

This recursive formulation reads explicitly as $N_{|C|} = 2^{|C|} - 1$. Again, we can show this by an inductive statement:

$$\begin{aligned} N_1 &= 1 \\ N_{|C|+1} &= 2 \cdot N_{|C|} + 1 \\ &= 2 \cdot \left(2^{|C|} - 1\right) + 1 \\ &= 2^{|C|+1} - 1. \end{aligned}$$

Considering also failed enter-checks, at most $|C| - 1$ many additional iterations may be performed in each inner loop of Algorithm 5.1. This leads to $(|C| - 1) \cdot \left(2^{|C|} - 1\right)$ iterations in the worst case.

□

5.2 Extensions and Adaptions

A major advantage of the greedy-algorithm compared to algorithms for the global-MILP is that it can be executed on distributed computation units. More specifically, we do not need a central unit, which is aware of all cars and traffic-lights with their respective states and specifications. In fact, each car can compute its individual trajectory. We only have to take care that the scheduled passing times of the vehicles are managed and reoptimizations, due to conflicting cars, are triggered.

5.2.1 A Real-World System for Optimized Traffic Flow

With the aid of wireless communication among vehicles and between vehicles and infrastructural devices, the presented algorithm could be suitable for a real-time procedure. In Section 6.6, we show experimentally that the greedy-algorithm yields promising runtimes regarding real-time systems.

Such an application called *greedy-cruise-control* could be designed as follows: whenever a car is within a certain range with respect to a traffic-light, the system is activated. This is, in a sense, equivalent to a car entering the network in the greedy-algorithm. Information about the traffic-light's position, geometry of the intersection, and so on can be exchanged via wireless communication, cf. Section 3.4.1, using the intersection-message. The solve-step of the algorithm can then be performed on a computational device in the car. Necessary information about preceding cars – via messages of type CAM and DENM – can be exchanged either directly between the cars or could also be distributed by the traffic-light.

Once the car calculated its preferred trajectory and passing times over the intersection, it communicates this information to the traffic-light. There, the enter-check and blocking-check are performed. In other words, the traffic-light manages the passing times of the cars and adapts its light-switching schemes accordingly. If any conflicts are detected, the traffic-light triggers a reoptimization in the respective cars analogously to the mechanism of the ordered list in the outline of the greedy-algorithm. To this end, it shares information about the timesteps, which hinder the particular car to pass the intersection as preferred. By fixing the appropriate trigger-variables, the car incorporates these information for the reoptimization process. Afterwards, the calculated passing times (and possibly the trajectory) are communicated to the traffic-light. This process is performed until no further changes are triggered. Neither by the traffic-light for reasons explained above nor by the car itself. As the available messages for C2X-communication do not invoke data which are crucial for the discussed functionality, either the present message types have to be extended, or new messages have to be introduced. In particular, the possibility to transmit calculated trajectories and passing times from the car to the traffic-light has to be provided. Additionally, the traffic-light must be able to broadcast information about possibly blocked time steps to the car.

As we already discussed, performing calculated trajectories exactly is only hard to achieve in practice. Therefore, the greedy-cruise-control could contain a mechanism which allows a vehicle to trigger a reoptimization. This mechanism also has to be covered by (newly introduced or extended) wireless communication messages.

In such a practical system, we do not operate on a fixed setting as described in Section 5.1. Hence, situations might occur, where the algorithm does not terminate – especially, if each car can trigger a reoptimization. Because of bounds on the physical quantities, it might also happen that no feasible solutions exist for single cars, e. g., because of reoptimizations which are triggered shortly before the car enters the intersection area. For this reason, a mechanism should be included, which fixes the switching times of the traffic-light some seconds in advance inhibiting these effects. Nevertheless, the greedy-cruise-control might realize some security fallbacks analogously to the mechanism of the RACC. For example, the car's ACC, cf. Section 3.4.3.3, could always overrule the system for security reasons.

Another issue that has to be considered is, how cars which are not equipped with the greedy-cruise-control can be incorporated. To this end, we can use the car's ACC-system and the security fallback explained above. A more sophisticated approach would be that each car is aware of all equipped cars and their respective positions by wireless communication. Consequently, if the radar sensor detects a preceding car which is not broadcasting these information, the greedy-cruise-control could be disabled.

Additionally, we have to ensure that the traffic-light's signal-states, which are based on the calculations of equipped cars, do not change rapidly. More generally, the resulting switching schemes should allow human drivers to react to the signal states properly. In absence of cars with enabled greedy-cruise-control, the traffic-light performs its usual switching scheme.

5.2.2 Additional Traffic-Light Regulations

In Section 4.2.5, we already discussed legal regulations for switching schemes of traffic-lights. These can be incorporated in the greedy-algorithm by simply adding a third kind of check after the blocking-check. This *traffic-light-check* can easily be included into the greedy-cruise-control. As the other checks are executed by the traffic-light's processing unit, the traffic-light-check can also be performed there. In particular, after the blocking-check is finished, a valid solution for the time horizon $[0, t]$ exists. The traffic-light-check examines the planned regulations of the traffic-light's switching scheme in increasing order in time. Once a violation is detected, the indicator variables for this time step (and possibly further time steps) are fixed accordingly. Afterwards, those cars, which have not completely passed the intersection until t, are reinserted into S.

Lemma 5.2. *The extension of the greedy-algorithm to respect legal traffic-light regulations increases the maximum number of iterations by the factor N for a considered time grid of* $\{0, \dots, N\}$.

Proof. Based on the proof of Theorem 5.1 we count the number of loops occurring additionally when introducing traffic-light-checks. After the conflict-check and blocking-check is performed, it is possible that the current switching scheme is invalid. This can happen at most t times if the greedy-algorithm currently considers the subgrid $\{0, \dots, t\}$ and leads to an additional factor of N in the worst case. □

In the following section, we investigate the performance and quality of the greedy-algorithm's solutions and compare them to the other methods for improving traffic flow, which we have introduced in the course of this thesis. We will also measure the impact of different kinds of parameters for legal traffic-light regulations, as well as the impact of different percentages of non-equipped cars.

6 Numerical Results

In Sections 3, 4, and 5, we introduced different methods to improve traffic flow at traffic-light controlled intersections. These methods include an acceleration controller (RACC), which is already working in a car, cf. Section 3, as well as MILPs which calculate traffic flow from a global and an individual point of view, cf. Sections 4 and 5. In this section, we investigate these approaches regarding different parameters:

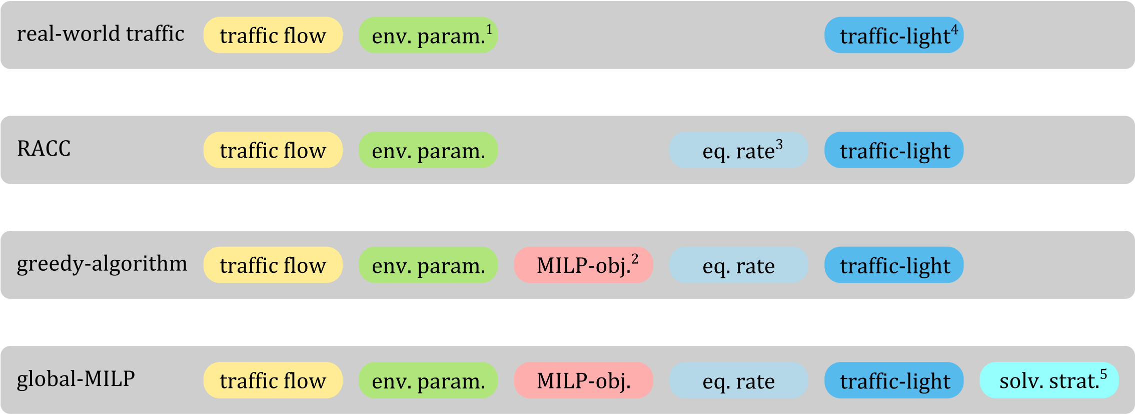
- The value of the global-MILP's *objective function*. For retrieving this value from solutions of the greedy-algorithm, some postprocessing has to be done.
- The mean *travel time* and *waiting time* of all cars in the network. While travel time measures the time it takes for each car to traverse the network, the waiting time measures the difference between travel time and theoretical time for an unobstructed traversal of the network. Both values are suitable for indicating the quality of traffic flow according to [25].
- The mean simulated *fuel consumption* and *CO_2-emissions* of all cars in the network. We also refer to these parameters as *environmental parameters*.
- The greedy-algorithm and RACC can be run online in a traffic simulation software. This allows us to vary the percentage of cars which are equipped with the respective system. The aforementioned parameters are measured with respect to different *equipment rates*.
- As stated in Section 4.2.5, there are *legal regulations* for the behavior of traffic-lights. If possible, the influence of these rules on the other parameters is measured.
- The RACC is additionally considered in terms of the quality of the performed trajectories in the car.

Figure 6.1 illustrates which solving method or system is evaluated in terms of which parameter. Note that all parameters are evaluated for all cars in the scenario: those which are optimized by one of the considered methods, and those which mirror real-world traffic. We call the former cars *equipped* and refer to the latter ones as *non-equipped*.

Before we discuss the results of the experimental data in Sections 6.4–6.6, we present the traffic-simulation software we used in Section 6.1, define a consistent experimental setting in Section 6.2, and consider the representation of real-world traffic in the simulation, cf. Section 6.3. At the end of this section, we rank the different methods and discuss possibilities for introducing them in real-world systems.

6.1 Traffic Simulation Software

In order to analyze and rate the effects on traffic which are achieved by the applications and methods in this thesis, it is necessary to represent the motion of cars on a road as well as traffic-lights and their signal states. The chosen software



real-world traffic	traffic flow	env. param.[1]			traffic-light[4]	
RACC	traffic flow	env. param.		eq. rate[3]	traffic-light	
greedy-algorithm	traffic flow	env. param.	MILP-obj.[2]	eq. rate	traffic-light	
global-MILP	traffic flow	env. param.	MILP-obj.	eq. rate	traffic-light	solv. strat.[5]

Figure 6.1: Overview of the different criteria which were measured for simulated real-world traffic and the different solving methods.
Annotations. [1] environmental parameters fuel consumption and CO_2-emissions, [2] objective value of the global-MILP, [3] equipment rate, [4] influence of (different) traffic-light regulation-rules, [5] performance of solving strategies

should allow to control the movements of all cars on the road and the traffic-light's signal-states very accurately and in fine time steps. A visual representation would be an additional feature that would facilitate debugging processes and provide a convenient style of presenting our systems. Hence, a microscopic traffic simulation software seems appropriate for our purposes. Such a software is related to microscopic traffic models, cf. Section 4.1.2, and allows the simulation of single cars and other entities in the network.

In this thesis, we use the *SUMO (Simulation of Urban MObility)* software framework, cf. [57], to represent the developed methods. SUMO is a free and open-source traffic simulation suite which allows modeling of intermodal traffic systems including road vehicles, public transport and pedestrians. Included with SUMO is a GUI which visualizes the processes in the network. SUMO comes with a variety of possibilities for evaluating the simulated traffic. These are, e. g., CO_2-emissions and fuel consumption according to [101]. Networks of roads can be either implemented using files in an .xml-style or can be imported from different sources, e. g., OpenStreetMap. The behavior of traffic-lights and cars can be accessed via *TraCI (Traffic Control Interface)*, which allows to retrieve values of simulated objects and to manipulate their behavior online. TraCI is available in different programming languages, e. g., Java and C++. We make use of the implementation of TraCI in Python as it is the best maintained one.

6.2 Network and Experimental Setting

We consider two different kinds of road network. These are fixed throughout this section to provide measurements which are comparable among the different methods. Thus, all of them are tested on the *single-intersection network*, cf. Figure 6.2, and on the *four-intersections network*, a schematic illustration of which is given in Figure 6.3. The single-intersection network consists of a single intersection of two roads, each with one lane in either direction. All lanes have a width of 2.5 meters and a stretch of 200 meters of road is added in each direction from the intersection. The four-intersections network consists of four roads with one lane in each direction. The roads are laid out in a grid-like pattern with two roads being

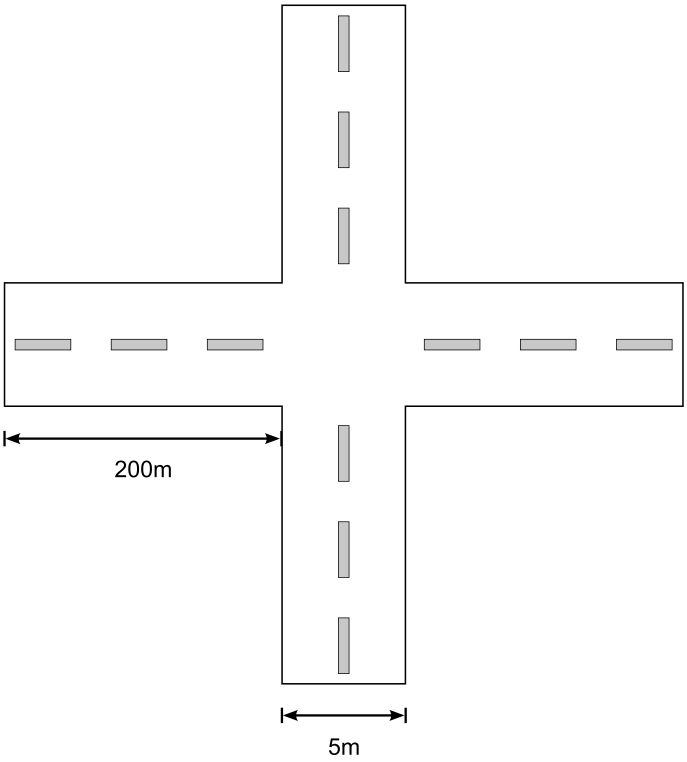


Figure 6.2: Layout of the single-intersection network.

oriented in north-south direction and the other two in east-west direction such that the resulting four intersections form the corners of a square. Each lane is 2.5 meters wide, and stretches of road are added such that each lane has 200 meters in front of the first intersection, 100 meters between the first and the second intersection and 200 meters behind the second intersection. As we only consider straight movement of cars without turning maneuvers, we allow opposite traffic-lights to be green simultaneously. More specifically, opposite traffic-lights always behave in the same way. Thus, for two opposite traffic-lights tl, tl' one has:

$$\chi_{tl,t} = \chi_{tl',t} \qquad \forall t \in T.$$

We can adjust the global-MILP slightly by omitting half of the trigger variables for the traffic-lights and formulate the Constraints (4.41), which relate the trigger zones and traffic-lights, accordingly.

The testing instances are the same for all different methods and algorithms, except for the global-realistic-MILP and the tailored branch-and-bound process. The reasons are explained in the respective sections. In fact, the arrival-time of each car is fixed for each testing instance. It is obtained by the following principle: a mean rate of cars per lane and minute for a certain amount of minutes is fixed (20 vehicles for the single-intersection network and 10 vehicles for the four-intersections network). The actual number of cars for each minute is randomly distributed according to a Poisson distribution, which according to [31] is suitable for generating traffic-related data. Specific arrival-times are distributed equidistantly over the minute. If present, the fixed switching schemes of the traffic-lights include a red-amber phase, a green phase, an amber phase, and

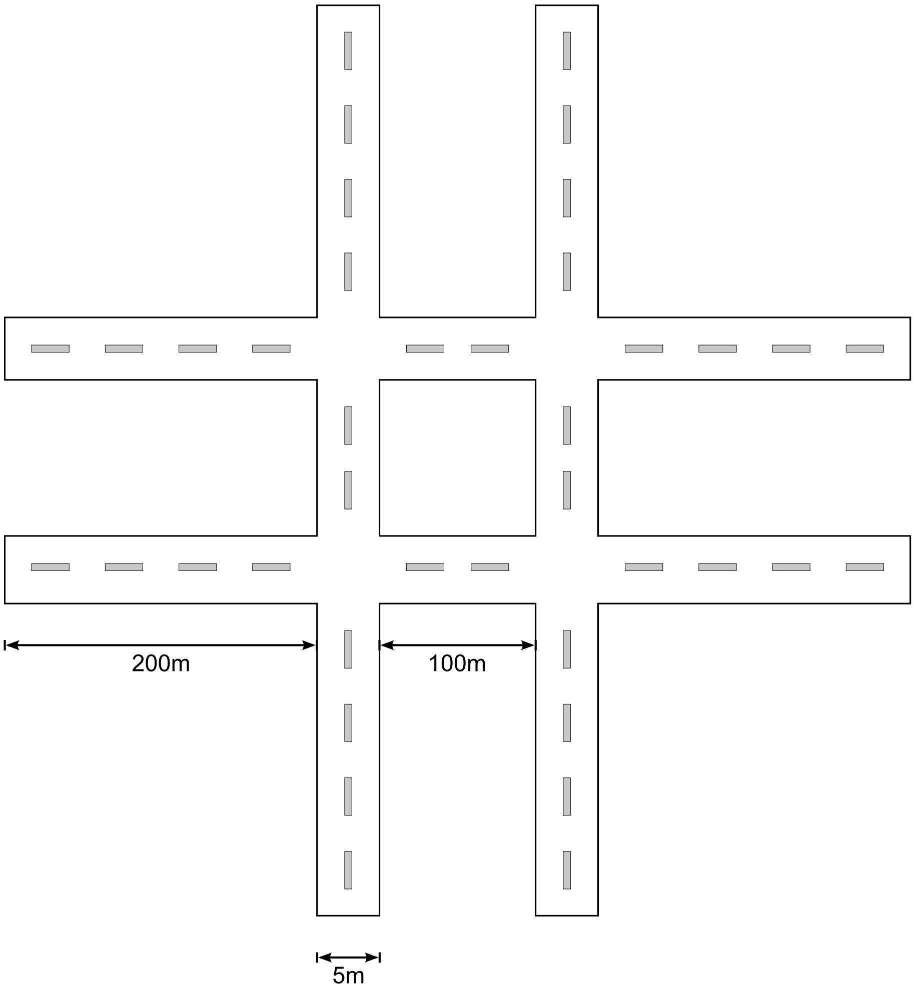


Figure 6.3: Layout of the four-intersections network.

a red phase for all traffic-lights of the intersection. The last red phase contains an evacuation time that guarantees that no vehicle is on the intersection when a conflicting lane is set to green, cf. [26]. Each cycle consists of two succeeding and equal phases for the two sets of conflicting (and pairwise intersecting) lanes. We denote the duration of the different phases by: duration red-amber phase | duration green phase | duration amber phase | duration red phase.

For each scenario (which is represented by a single column in the tables below), five different instances are generated and processed. The evaluated parameters are geometric means of all five instances and apply for single cars, except for the global objectives. For all methods, we set the sampling rate to 10 Hz, which means $dt = 0.1$ for the MILPs. Analogously, the sampling rate of the RACC's iterative algorithm as well as the discretization of the time horizon in the OCPs is set to 0.1 seconds. The time horizon, which occurs in the MILPs is always defined to be $\mathcal{T} := [0, 250]$. The physical bounds on the motion of the cars are chosen similarly for each method according to Table 6.1. All experiments were run on a system with Intel Core i5 CPU with 1.8 GHz, 8 GB memory and a 64-bit Windows 7 operation system. CPLEX V12.6 serves as MILP-solver for all optimization problems occurring here. The experiments concerning the branch-and-bound algorithm in Section 6.5.3 as well as the greedy-algorithm in Section 6.6 were executed with AMPL in version 20160325.

Table 6.1: Bounds on the states of the vehicles which are equivalent for each car and method.

	Value	Description
v_c^{min}	0	minimum velocity in m/s
v_c^{max}	13	maximum velocity in m/s
a_c^{min}	-3.5	minimum acceleration in m/s^2
a_c^{max}	2.5	maximum acceleration in m/s^2
j_c^{min}	-3	minimum jerk in m/s^3
j_c^{max}	3	maximum jerk in m/s^3
l_c	5	length of the car in m
g_c	0	security gap of the car in m

6.3 Real-World Traffic

A crucial part of analyzing the effects on traffic is to simulate real-world traffic appropriately. Usually, traffic is simulated in SUMO using car-following models, cf. Section 4.1.2. By default, the model by Krauß [59] is enabled which provides different parameters to adapt the behavior of cars in the simulation. Table 6.2 shows the adjustable parameters and typical values for each of them.

Unfortunately, these parameters do not lead to a realistic behavior of cars when passing urban traffic-lights. More specifically, the starting maneuvers and the way cars follow another starting car lack a satisfying realism. It seems not to be clear which values these parameters should attain to mirror real-world traffic at traffic-light intersections. In order to overcome this unsatisfying situation, we analyzed recorded data of vehicles passing an intersection in the city of Braunschweig. This particular intersection is equipped with technology to record traffic participants both visually with multiple stereo cameras and with radar sensors. We already referred to this intersection, which belongs to the Anwendungsplattform Intelligente Mobilität (AIM), in Section 3.4.5. The camera system does not only record the traffic on the intersection area visually but also identifies multiple parameters of traffic participants, e. g., position, velocity, acceleration, and type (car, van, truck, bike, bicycle, pedestrian). This data – which is visualized in Figure 6.4 – is available in a sampling rate of 4 Hz. Together with the information about the traffic-light's signal-states, we identify some of the parameters for the car-following model, cf. Table 6.3 and [48]. As the object identifier of the recording system is limited in terms of the monitored area and maneuvers of cars, we are only able to determine a subset of the available parameters. These are subsequently implemented in SUMO in order to achieve a more realistic behavior of vehicles

Table 6.2: Adjustable parameters for the default car-following model in SUMO with default values.

Parameter	Value	Description
accel	3.0	acceleration ability in m/s^2
decel	3.5	deceleration ability in m/s^2
sigma	0.5	driver imperfection $\in [0, 1]$
tau	2.0	time gap to front vehicle in s
minGap	2.5	gap to front vehicle if halting in m
maxSpeed	13.9	driver's desired velocity in m/s

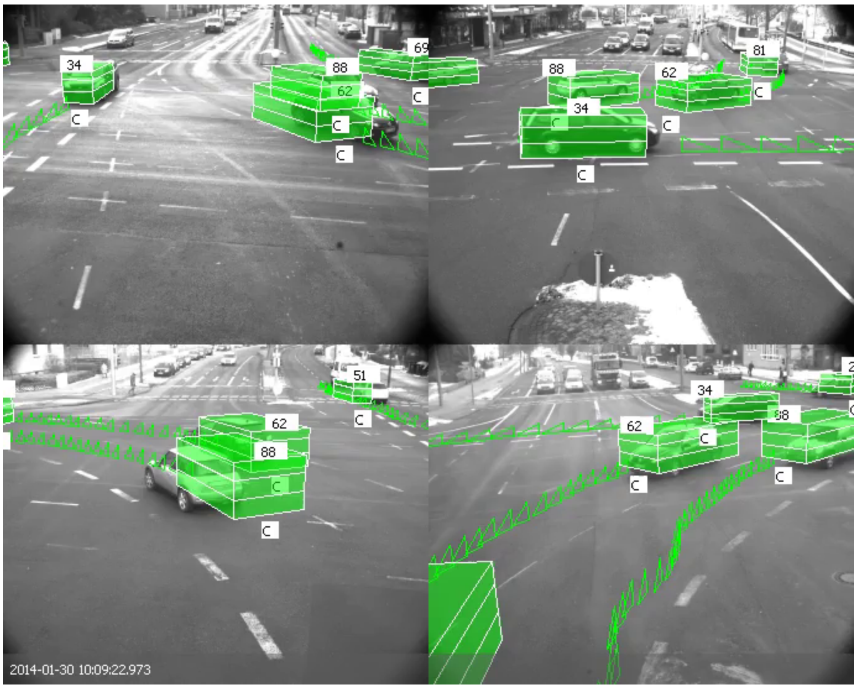


Figure 6.4: Screenshot of the video data used for calibrating the parameters of the car-following model. The green boxes indicate identified objects, green triangles visualize their predicted movement.

which are not equipped with one of the systems or solving methods. The remaining parameters are kept unchanged from the default parametrization. Note that the value *maxSpeed* does not coincide with v^{max}, which denotes the maximum allowed velocity. While maxSpeed influences the acceleration in the applied car-following model in the first place, the actual performed velocity of all cars is bounded by v^{max} to keep it consistent among the different solving methods. In addition to the behavior of real-world vehicles, the recordings at the intersection were used to measure the amount of cars that occur at an intersection during the rush hour. Thus, the testing instances for experiments on the single-intersection network show a mean arrival-rate of cars according to these measurements.

In Table 6.4 and Table 6.5, we present results for simulation of pure real-world traffic on the two networks. Comparing the parameter values in the different switching schemes, one can observe that the scheme which lasts one minute per cycle in both networks provides the best results. This outcome is quite intuitive: most of the time is lost when cars have to decelerate and accelerate in order to respect a red traffic-light. Once the cars are in motion, i. e., the traffic *flows*, there is not much friction. In both networks, the emissions and fuel consumption behave accordingly.

Table 6.3: Parameters for the default car-following model in SUMO with values identified via evaluation of real-world data.

Parameter	Value	Description
accel	1.8	acceleration ability in m/s^2
tau	1.0	time gap to front vehicle in s
maxSpeed	13.6	driver's desired velocity in m/s

Table 6.4: Obtained values for real-world traffic on the single-intersection network with different switching schemes for the traffic-lights.

	1s | 9s | 3s | 2s[1]	1s | 24s | 3s | 2s[1]
travel time [s]	96.96	69.90
waiting time [s]	64.98	37.82
fuel consumption [ml]	67.87	53.31
CO_2 emissions [g]	154.97	121.74

Annotations. Values are geometric means per vehicle of five testing instances.

[1] duration red-amber phase | duration green phase | duration amber phase | duration red phase

Note that we do not compare the measurements of the two different networks against each other. This is mainly due to the fact that we consider loose traffic in the bigger network. Otherwise, the global-MILP would not be solvable in reasonable time. We refer to Section 6.5 for a further discussion on this issue.

6.4 Numerical Results for the RACC

In this section, we consider the RACC, cf. Section 3. To this end, we evaluate the quality of performed trajectories of the car by comparing planned trajectories and actually performed ones. Afterwards, we have a look at the effect of the RACC on real-world traffic using SUMO. Thus, we use an implementation of the acceleration controllers of the deferred-transit and pole-start-regime via the ACADO-framework, which are also running in the test vehicle. Apart from that, the functionality of the system as well as the data exchange between SUMO and the ACADO-controllers is implemented in Python. Wireless communication between the traffic-lights and the approaching cars is not further modeled in terms of delay times, lost messages, or other issues arising in a realistic setting. The C2X-range is fixed at 200 meters. In fact, once a car reaches this distance to a traffic-light, the RACC is enabled. Vehicle-specific parameters are chosen according to Table 6.1. Additionally, the value for v_{pref}, which is the preferred velocity of the car when passing the intersection, is set to $13\,m/s$. The time horizons for the deferred-transit and pole-start-regime are chosen to last $15\,s$. The minimal velocity for performing a deferred-transit is equal to $5\,m/s$. Finally, the stopping-gap during the pole-stop-regime equals $10\,m$.

Remember that besides the acceleration controllers, the RACC includes functionalities of the car's ACC-system, e. g., when the car-following regime is activated. Due to no implementation of the ACC which could be invoked in the simulation

Table 6.5: Obtained values for real-world traffic on the four-intersections network with different switching schemes for the traffic-lights.

	1s | 9s | 3s | 2s[1]	1s | 24s | 3s | 2s[1]
travel time [s]	79.99	65.70
waiting time [s]	40.36	26.08
fuel consumption [ml]	65.33	55.07
CO_2 emissions [g]	149.20	125.76

Annotations. Values are geometric means per vehicle of five testing instances.

[1] duration red-amber phase | duration green phase | duration amber phase | duration red phase

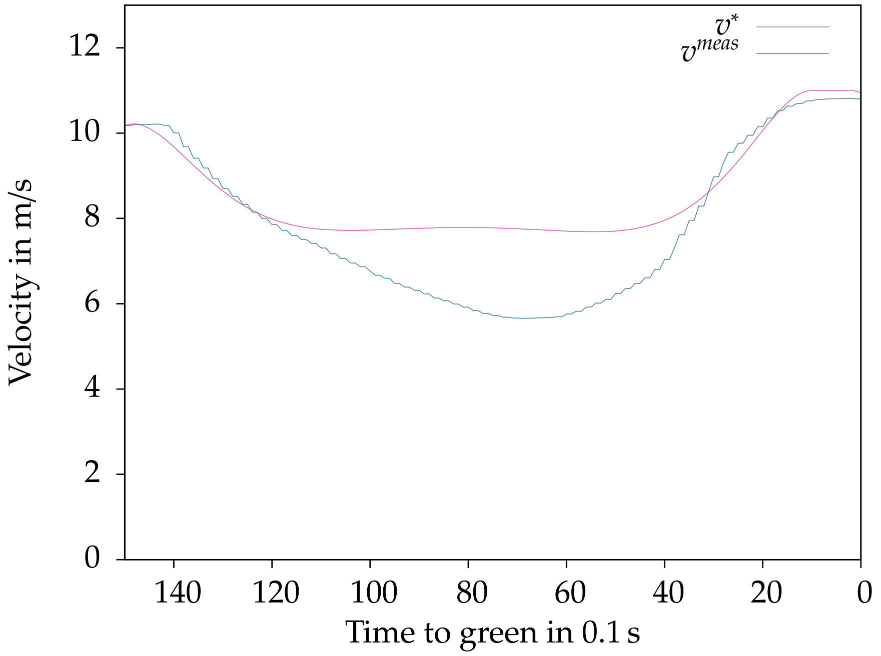


Figure 6.5: Velocities according to initially planned trajectory v^* and actually performed one v^{meas}.

being available, we model the car's behavior in this case via the car-following model mentioned above. This model is also parametrized like real-world traffic. Certainly, the behavior of real drivers differs from the trajectories induced by the ACC. But, beyond the fact that we do not have an implementation of the ACC and somehow have to incorporate following maneuvers, we can adjust the ACC of the test vehicle. This way, it is possible to reproduce the behavior of the cars in the simulation to a certain standard in the test vehicle. As the RACC has no influence on the behavior of the traffic-lights, we run simulations for different switching schemes, which we already used above.

6.4.1 Quality of Performed Trajectories

Before we dive into the evaluation of the effects on real-world traffic, we are interested in the performance of the implemented controllers in a real car. For the purposes of the discussion, we focus on the deferred-transit-regime. Figures 6.5 and 6.6 illustrate deviations between the solution of an OCP for the whole time horizon and the actually performed velocity and distance in a test drive for a real car. Note that the depicted solution of the OCP is calculated in the very beginning of the maneuver. The performed trajectories are results of an MPC-process, which basically resolves the OCP considering actualized measurements, cf. Section 2.1.2. Especially the deviation between initially planned velocity and the measurements during the test drive seems to be rather large with up to 2 m/s. Also a time gap of about half a second between the planned deceleration in the beginning of the maneuver and the actually performed deceleration is obvious. In fact, this gap is even bigger for an acceleration process, although it is not clearly visible in the graph. This is due to the fact that the car is already accelerating when the acceleration-phase in the initial solution begins – because of deviations from the solution in the first part of the maneuver.

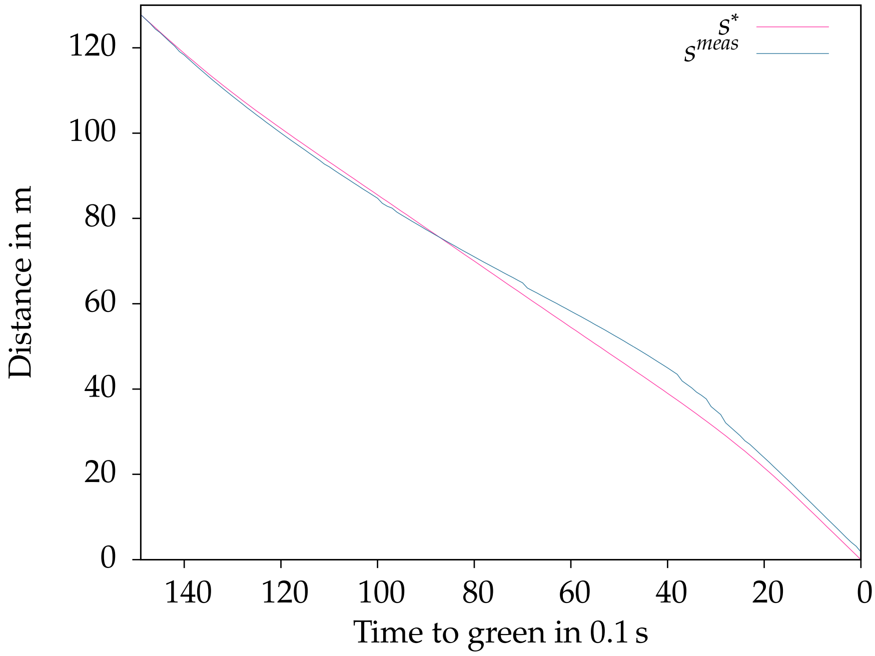


Figure 6.6: Distances according to initially planned trajectory s^* and actually performed one s^{meas}.

Table 6.6 shows these inaccuracies in terms of the *mean squared error (MSE)* between the initial solution and actual realization in the car. The mean squared error is defined as:

$$MSE(x) = \frac{1}{N} \cdot \sum_{i=1}^{N} \left(x_i^* - x_i^{meas} \right)^2$$

for a variable x and N measurements x_i^{meas} at time points $i \in \{1, \dots, N\}$. In our test drives N equals 150 due to the length of the time horizon of 15 s and the MPC's discretization of 0.1 s. Possible reasons for these errors are dead times that arise between triggering a certain acceleration or deceleration and the actual performance by the engine. These dead times are about 0.5 s and 1 s. In general, acceleration processes ($a > 0$) cause higher dead times which is mainly caused by the more complex physical process. Hopefully, this effect diminishes for electrical engines. Besides these dead times, the process certainly is confounded by measurement-errors of the car's position, velocity, and acceleration leading to inaccurate solutions. The solving times of the single OCP's fulfill our purposes and are between 0.001 s and 0.1 s for nearly all iterations.

In the end, it is important that the deviation in the last time step – when the traffic-light switches to green – the car's velocity is sufficiently close to the desired velocity without being greater, and the distance to the stopping line is close to zero. Considering again the graphs in Figures 6.5 and 6.6, we can assess only slight differences of 1.8 m and 0.18 m/s ($v_{pref} = 11\ m/s$ here). Table 6.6 depicts these values for two more test drives where all of them are satisfying besides relatively great deviations during the maneuver.

Table 6.6: Mean squared errors of initially planned trajectories and actually performed ones considering acceleration, velocity, and distance. Additionally, the total deviation concerning distance and velocity in the last time step, i. e., when the traffic-light switches to green is given.

	MSE(a)	MSE(v)	MSE(s)	$\lvert s^*_{t_f} - s^{meas}_{t_f}\rvert$	$\lvert v^*_{t_f} - v^{meas}_{t_f}\rvert$
test drive 1	0.28	1.24	7.60	1.81	0.18
test drive 2	0.84	3.05	51.73	1.84	0.06
test drive 3	0.62	3.90	86.93	3.55	0.81

Annotations. MSE means the mean squared error between initial optimal solution and actually performed trajectory with respect to respective variable.

6.4.2 Visualization in SUMO

At this point, we regard the representation of the RACC's functionality in SUMO. In Figures 6.7–6.13, screenshots of a single intersection in SUMO can be seen. The colored bars indicate the color the traffic-light is showing in the depicted moment. Orange illustrates the red-amber phase, which is enabled in german traffic-lights between the red phase and the green phase and usually lasts one second, cf. [26]. The different colors of the cars encode the regime which is currently activated. Red indicates the pole-stop-regime, green the pole-start-regime, dark blue the deferred-transit-regime, cyan the free-transit-regime, and yellow denotes the car-following-regime. White cars are not equipped with the RACC and therefore only controlled via the car-following model.

Figures 6.7–6.9 show the impact of the pole-stop and subsequent pole-start-regime compared to non-equipped cars. The first vehicle on the lane going from left to right is stopping with a gap of 10 meters in front of the red traffic-light while the non-equipped cars going from right to left are stopping directly in front of it. One can see the equipped car accelerating in the pole-start-regime during the red phase. Hence, it passes the stopping line after the traffic-light switched to green with a higher velocity than the non-equipped cars. Additionally, the white vehicles still have to react to the switch to green. Note that in the simulation, even non-equipped vehicles accelerate during the red phase when following an equipped car, as they only consider the leading vehicle. In Section 3.4.6, we already discussed results from surveys revealing this behavior by real drivers.

Figures 6.10–6.11 illustrate the behavior of cars with activated deferred-transit-regime: without falling below v^{min} the car going from left to right passes the intersection, while the non-equipped cars stop in front of the traffic-light. In both scenarios, it seems that cars running the RACC exploit a single green phase more efficiently, i. e., more vehicles are able to pass the intersection during a single green phase. Possibly, this could result in an improved traffic flow.

Finally, Figures 6.12–6.13 illustrate the free-transit-regime. The equipped car is passing the intersection while keeping its desired velocity. In contrast, the non-equipped car is not aware of an imminent switch to green and therefore decelerating in front of the traffic-light. Again, an improvement of the overall traffic flow is hopefully achieved by the behavior of the equipped car. We investigate the impact of the RACC on traffic flow in the section below.

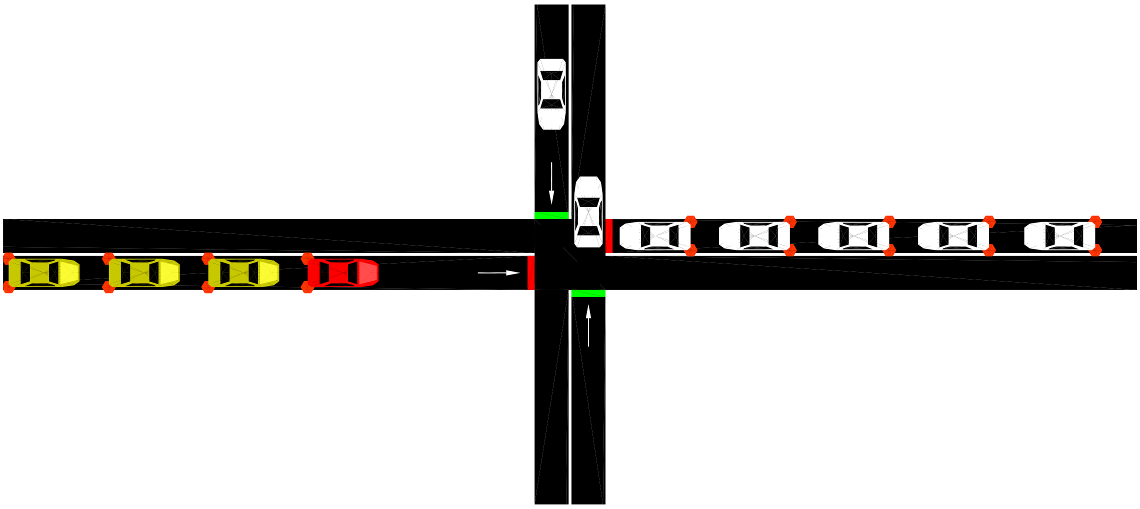


Figure 6.7: Visualization of the RACC using SUMO. The red car performs the pole-stop-regime. Yellow cars follow according to the car-following-regime. White cars model real-world traffic.

6.4.3 Effects on Traffic Flow

Table 6.7 and Table 6.8 show results of simulations according to the explanations in Section 6.2. First, we have a look at the differences between real-world traffic evaluated in the former section and RACC-traffic with an equipment rate of 100 %. At first glance, we can assess an improvement in travel time and waiting time for all combinations of network and switching schemes. The traffic in scenarios with the shorter cycle time benefits more from the RACC, in particular with a decrease of up to 28 % regarding the waiting time. This result corresponds to the intuition that scenarios with a shorter cycle time benefit more as the RACC mainly optimizes the starting maneuvers. Once the traffic flows more or less constantly, the RACC has no impact. Besides the fact that in the four-intersections network possibly more starting maneuvers occur, the gain in traffic flow is higher in the single-intersection network. This is possibly due to the higher density of cars in the smaller network. The values of the environmental parameters show a similar behavior.

Table 6.7: Obtained values for traffic induced by the RACC on the single-intersection network with different switching schemes for the traffic-lights and percentage of equipped cars.

	1s | 9s | 3s | 2s[1]			1s | 24s | 3s | 2s[1]		
	10 %[2]	50 %[2]	100 %[2]	10 %[2]	50 %[2]	100 %[2]
travel time [s]	95.23	87.90	78.95	69.49	68.44	66.96
waiting time [s]	63.23	55.85	46.91	37.44	36.39	34.88
fuel consumption [ml]	66.69	61.78	55.01	52.77	50.23	48.11
CO_2 emissions [g]	152.27	141.07	125.61	120.49	114.70	109.86

Annotations. Values are geometric means per vehicle of five testing instances.
[1] duration red-amber phase | duration green phase | duration amber phase | duration red phase,
[2] percentage of cars which are equipped with the RACC

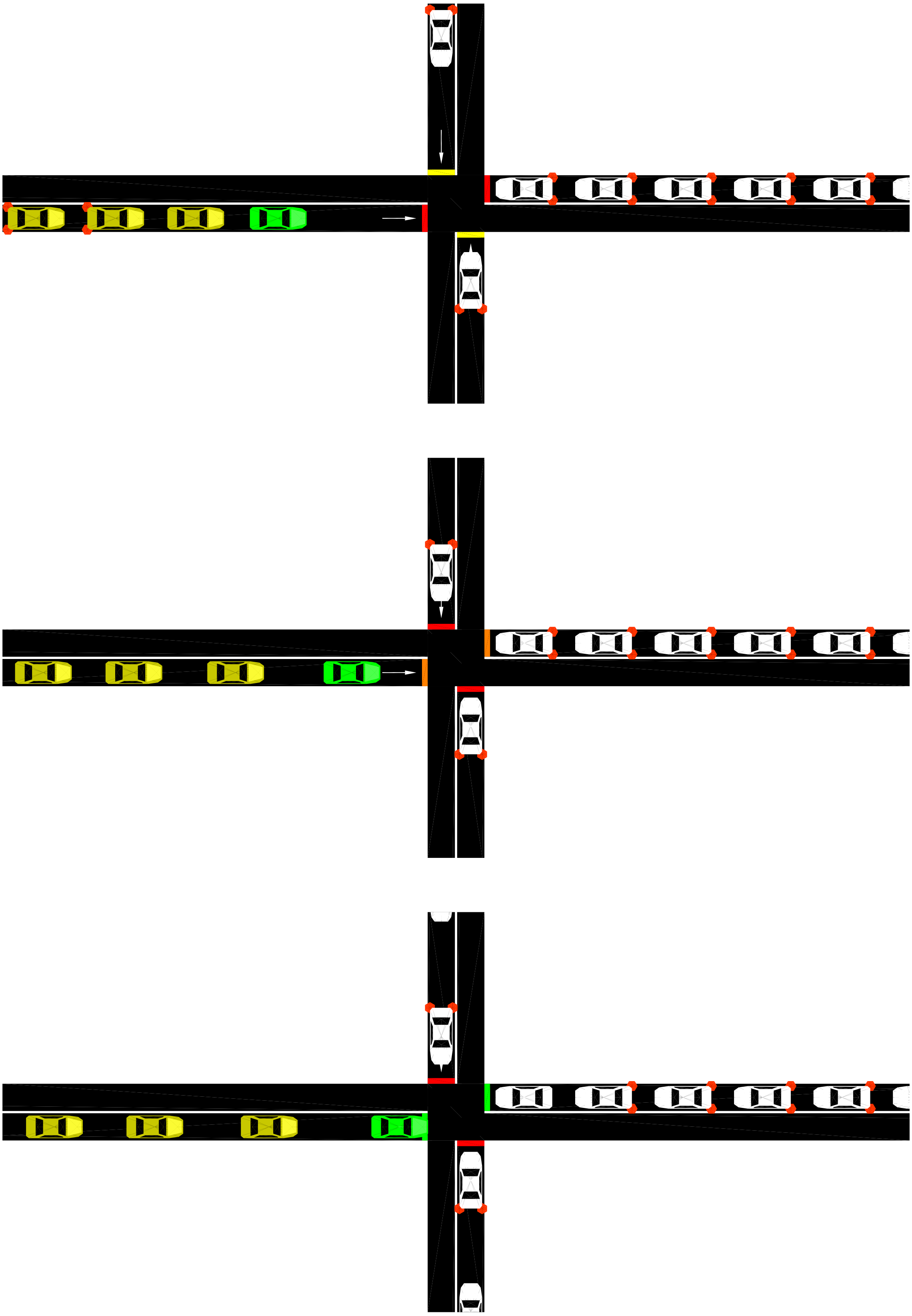


Figure 6.8: Visualization of the RACC using SUMO. The green car performs the pole-start-regime while the traffic-light is still red. Yellow cars follow according to the car-following-regime. White cars model real-world traffic.

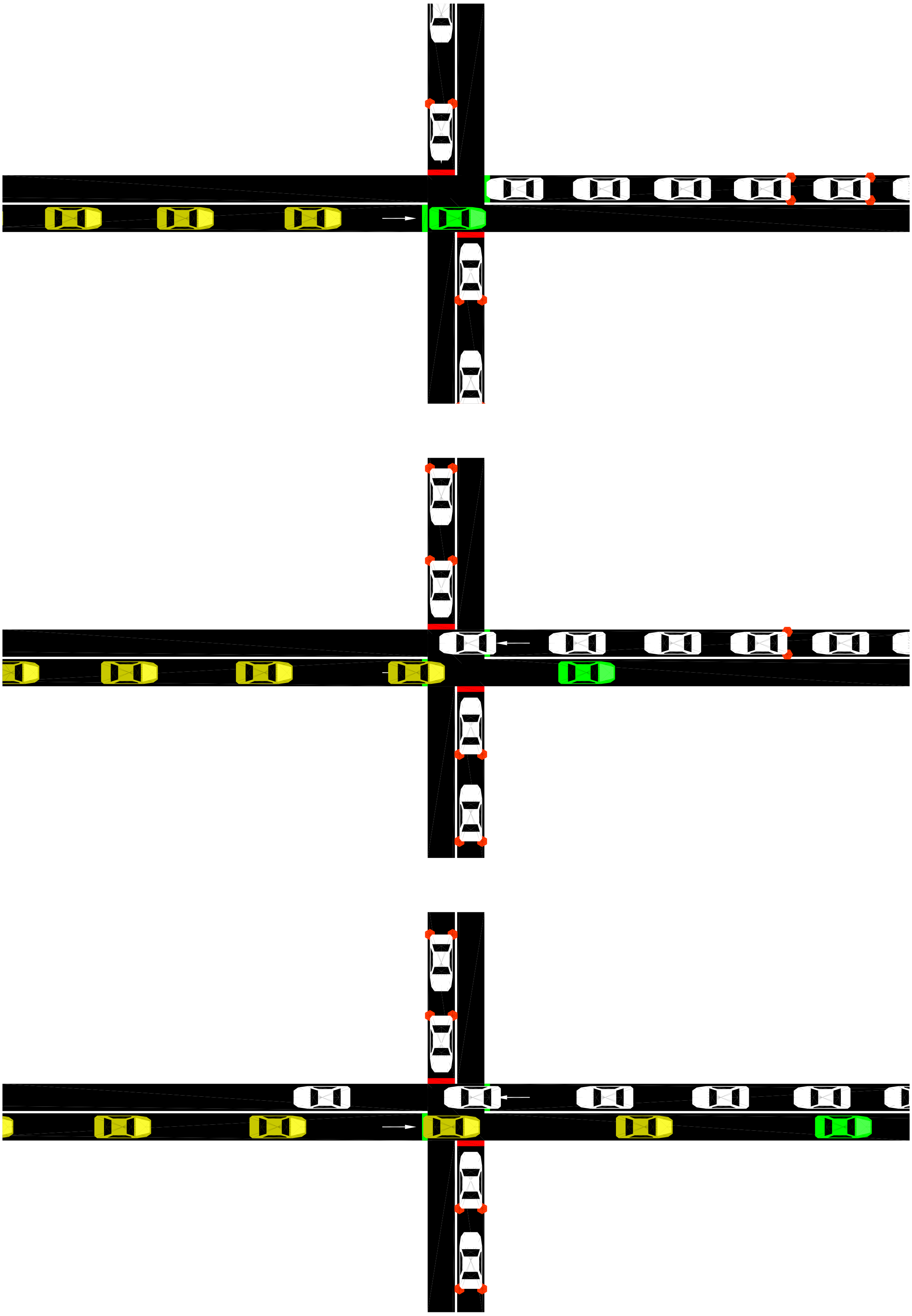


Figure 6.9: Visualization of the RACC using SUMO. The green car performed the pole-start-regime. Because of its acceleration during the red phase, it passed the stopping line with a higher velocity after the traffic-light switched to green. Hence, the green phase might be used more efficiently in terms of traffic flow. Yellow cars follow according to the car-following-regime. White cars model real-world traffic. The difference between the traveled distances of equipped and non-equipped cars is visible.

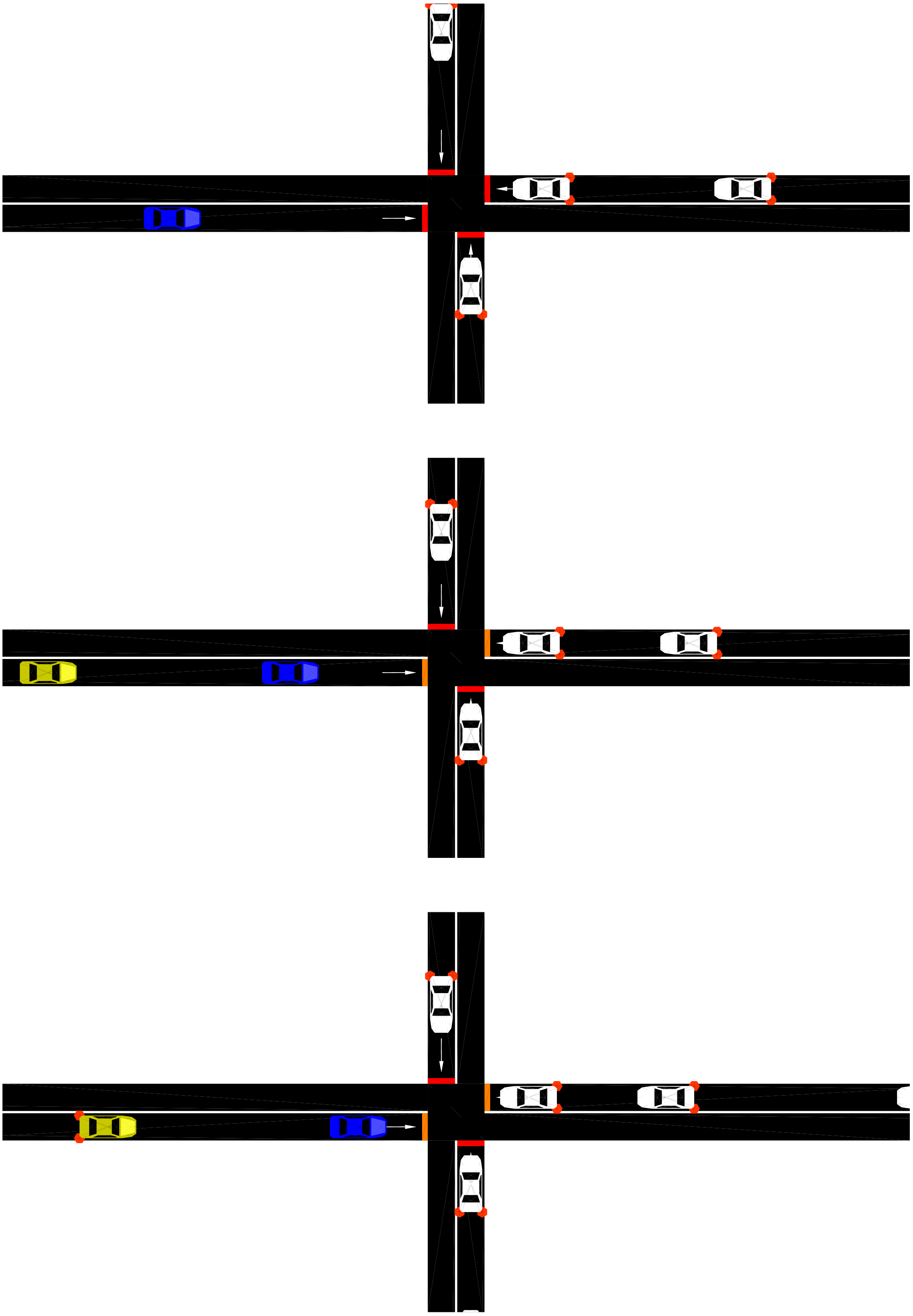


Figure 6.10: Visualization of the RACC using SUMO. The blue car performs the deferred-transit-regime. The yellow car follows according to the car-following-regime. White cars model real-world traffic.

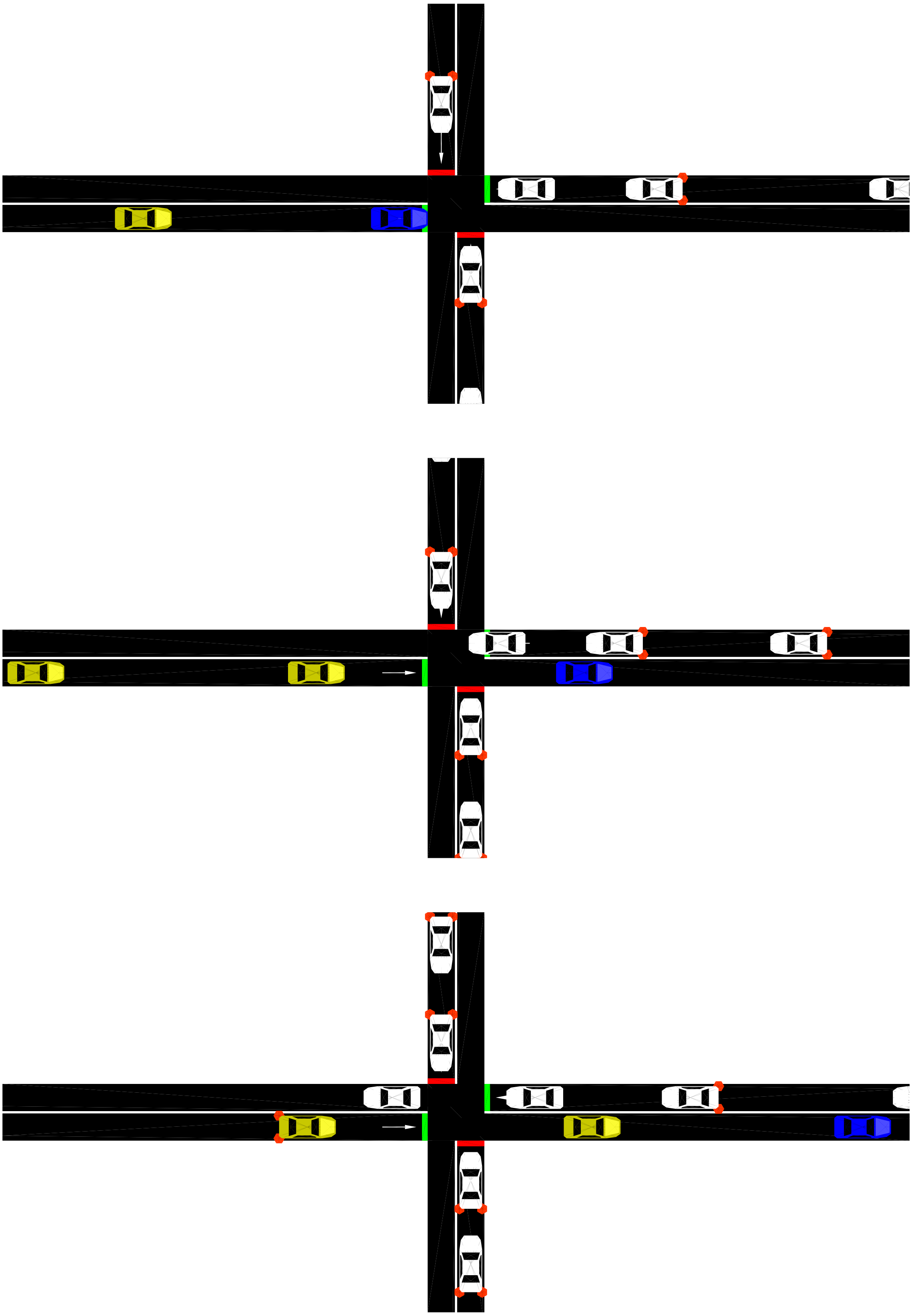


Figure 6.11: Visualization of the RACC using SUMO. The blue car performs the deferred-transit-regime and passes the intersection with the minimum value of desired velocity and 40 kilometers per hour. Hence, the green phase might be used more efficiently in terms of traffic flow. The yellow cars follow according to the car-following-regime. White cars model real-world traffic. The difference between the traveled distances of equipped and non-equipped cars is visible.

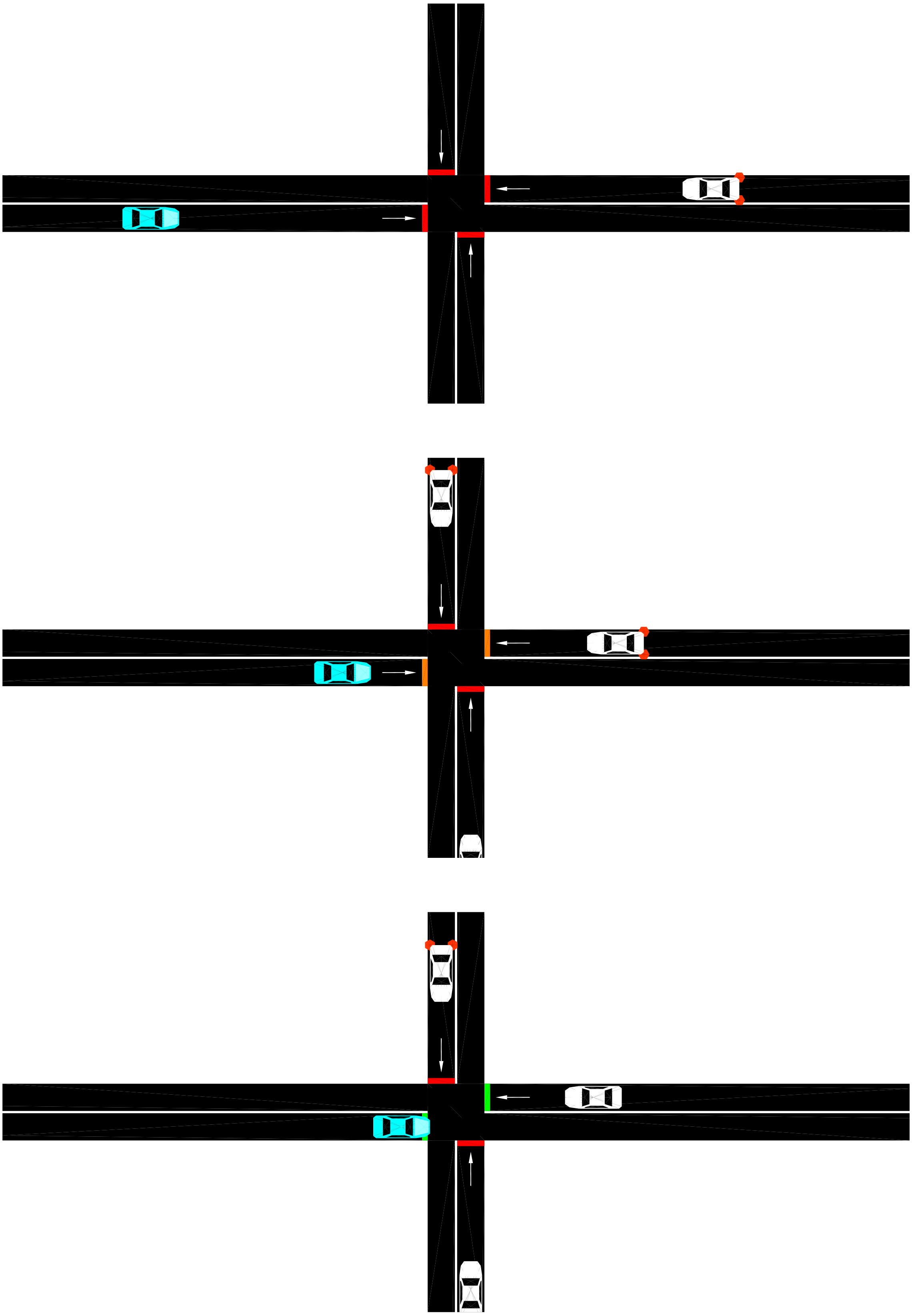


Figure 6.12: Visualization of the RACC using SUMO. The cyan car performs the free-transit-regime and passes the intersection with the desired velocity. White cars model real-world traffic. Due to the unawareness of an imminent switch to green, the non-equipped car going from right to left decelerates in front of the traffic-light.

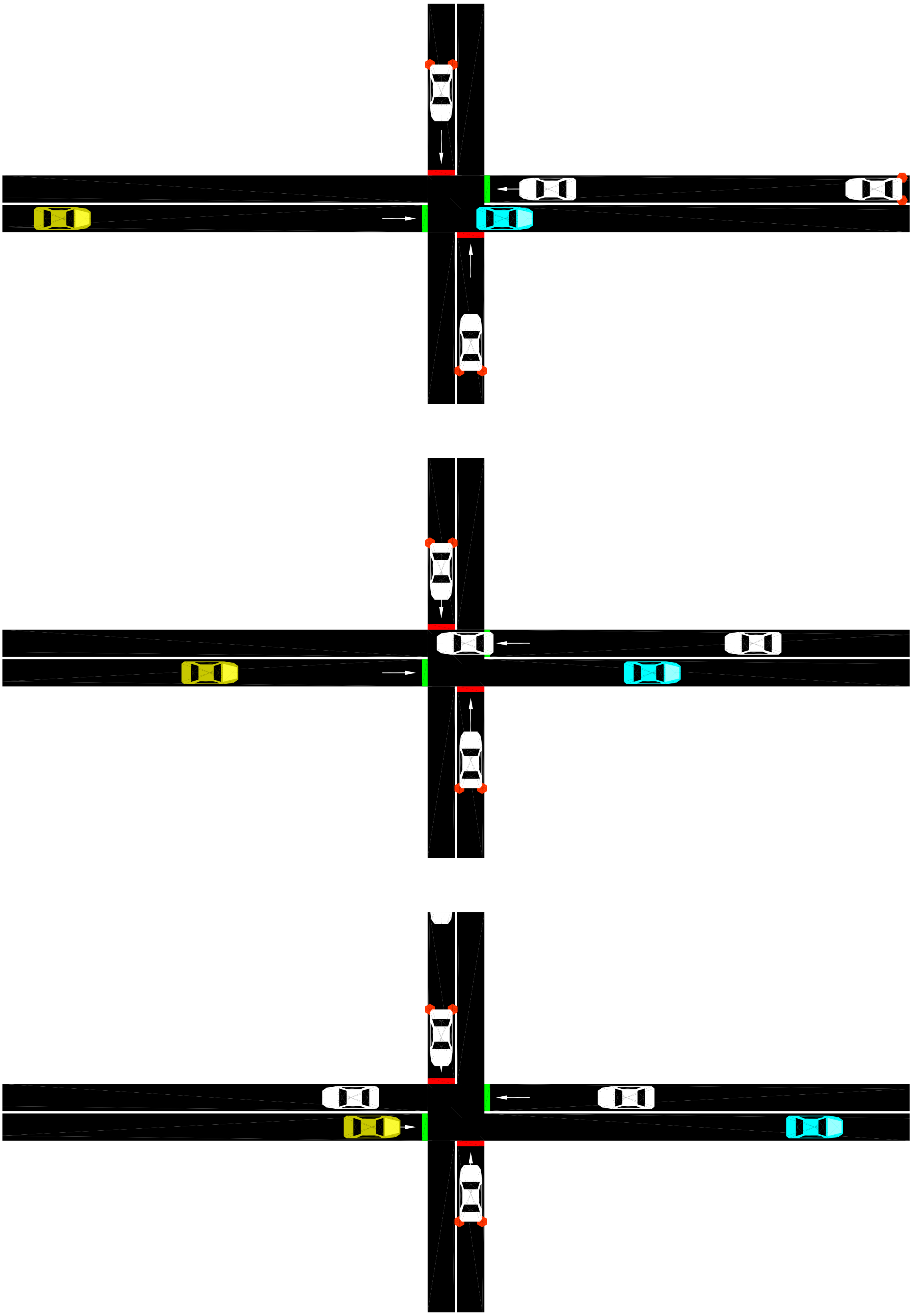


Figure 6.13: Visualization of the RACC using SUMO. The cyan car performed the free-transit-regime and passed the intersection with its desired velocity. In contrast, the non-equipped car going from right to left had to decelerate and accelerate in front of the traffic-light. The resulting difference in traveled distances is visible.

Table 6.8: Obtained values for traffic induced by the RACC on the four-intersections network with different switching schemes for the traffic-lights and percentage of equipped cars.

	1s | 9s | 3s | 2s[1]			1s | 24s | 3s | 2s[1]		
	10 %[2]	50 %[2]	100 %[2]	10 %[2]	50 %[2]	100 %[2]
travel time [s]	79.37	78.63	70.91	65.18	64.44	63.13
waiting time [s]	39.73	39.06	31.26	25.55	24.82	24.50
fuel consumption [ml]	65.06	63.60	57.18	54.67	53.66	51.73
CO_2 emissions [g]	148.58	145.23	130.58	124.85	122.55	118.14

Annotations. Values are geometric means per vehicle of five testing instances.
[1] duration red-amber phase | duration green phase | duration amber phase | duration red phase,
[2] percentage of cars which are equipped with the RACC

Focusing solely on the RACC, we recognize the same development of the values between the different switching schemes as in the real-world traffic: in both networks, traffic flow is higher for the longer cycle times of the traffic-lights. Better values for CO_2-emissions and fuel consumption can be assessed accordingly.

At this point, we want to investigate the impact of the equipment rate. For a fixed equipment rate, the differences between the scenarios with different cycle times reveal analogous trends for all measured parameters. When the network and switching scheme is fixed, all parameters improve when the equipment rate is increased. We can assess a decrease in travel time of up to 17 % for the short-cycled single-intersection network when increasing the equipment rate from 10 % to 100 %. A sightly lower gain can be determined for the short cycle time in the four-intersections network. We discussed reasons for a higher gain when introducing the RACC for shorter traffic-light cycles above. Worth noticing is the fact that in none of the analyzed scenarios, the RACC leads to a decrease in traffic flow – even for low equipment rates. There is a small increase in CO_2-emissions and fuel consumption for the short-cycled four-intersections network with an equipment rate of 10 %. However, the overall effect of the RACC seems to be rather negligible for this scenario when regarding the differences in travel and waiting time.

6.4.4 Effects on Cities

The effect of the RACC on traffic is also discussed in the UR:BAN-project, cf. [58]. The setting of the simulation which serves as a basis to measure these effects slightly differs from the one used for the experiments in this section. In particular, also roads with multiple parallel lanes allowing different turning maneuvers are considered. Additionally, the generated traffic and the switching schemes of the traffic-lights differ from the previously analyzed data. As the effects of the RACC on traffic for a single intersection are similar to those discussed above, we do not stress them again. But beyond these evaluations, the results are scaled up for whole cities based on the amount and distribution of different types of intersection layouts. In fact, a decrease of up to 15% in travel time is assessed for traffic within a distance of 300 meters in front of traffic-light controlled intersections. Likewise, fuel consumption is reduced by up to 7%.

Table 6.9: Values in the experimental setting for parameters that induce regulations on the traffic-light's switching scheme.

	Value	Type
pulse interval	$10 \cdot dt$ (1 s)	grid-like time steps
green period	$50 \cdot dt$ (5 s)	lower bound
red period	$50 \cdot dt$ (5 s)	lower bound
cycle time	0 (due to high runtimes)	lower bound
evacuation time	$30 \cdot dt$ (3 s)	lower bound

6.5 Numerical Results for the global-MILP

In this section, we discuss the quality of the global-MILP's solutions in terms of the known parameters. In particular, we consider travel time, waiting time, CO_2-emissions, and fuel consumption as in the sections above. To this end, the solutions calculated offline via CPLEX are transferred into SUMO. The Python interface TraCI allows us to control the motion of all cars in each timestep. Thus, we can retrieve the parameters of interest for the solutions of the global-MILP. Afterwards, we investigate the performance of the solving process. To this end, the solving strategies which are developed in Section 4.5 are compared among each other.

6.5.1 Effects on Traffic Flow

Before we have a look at the particular parameters, we note that incorporating legal traffic-light regulations in the global-realistic-MILP cannot be realized as in the sections above. There, a fixed cycle is realized in the SUMO-simulation, which the cars react to online. In the global-realistic-MILP, cf. Section 4.2.5, legal regulations are expressed by lower bounds on the duration of the green and red phase, as well as by a lower bound on the cycle time, which is the time between two successive green phases. Additionally, an evacuation time, in which no car is allowed to enter the intersection, is enforced. To ensure further realism, switches are only possible on grid points of the pulse interval, i. e., only every second. Table 6.9 provides specific values for these bounds in the experimental data. Note that for our particular scenario with two conflicting traffic-lights, the values for bounds on the green period and red period coincide.

In Table 6.10, results concerning the quality of traffic flow and environmental parameters are displayed. The two scenarios of the global-MILP provide an increase in traffic flow (in terms of a very short waiting time) of 99 % and 98 % compared to the RACC, while the values for the environmental parameters decrease

Table 6.10: Obtained values for the traffic induced by solutions of the global-MILP and global-realistic-MILP.

	single-intersection	single-intersection(realistic)	four-intersections
travel time [s]	31.86	31.89	39.57
waiting time [s]	0.32	0.33	0.34
fuel consumption [ml]	31.00	31.50	38.36
CO_2-emissions [g]	70.80	71.97	87.64

Annotations. Values are geometric means per vehicle of five testing instances.

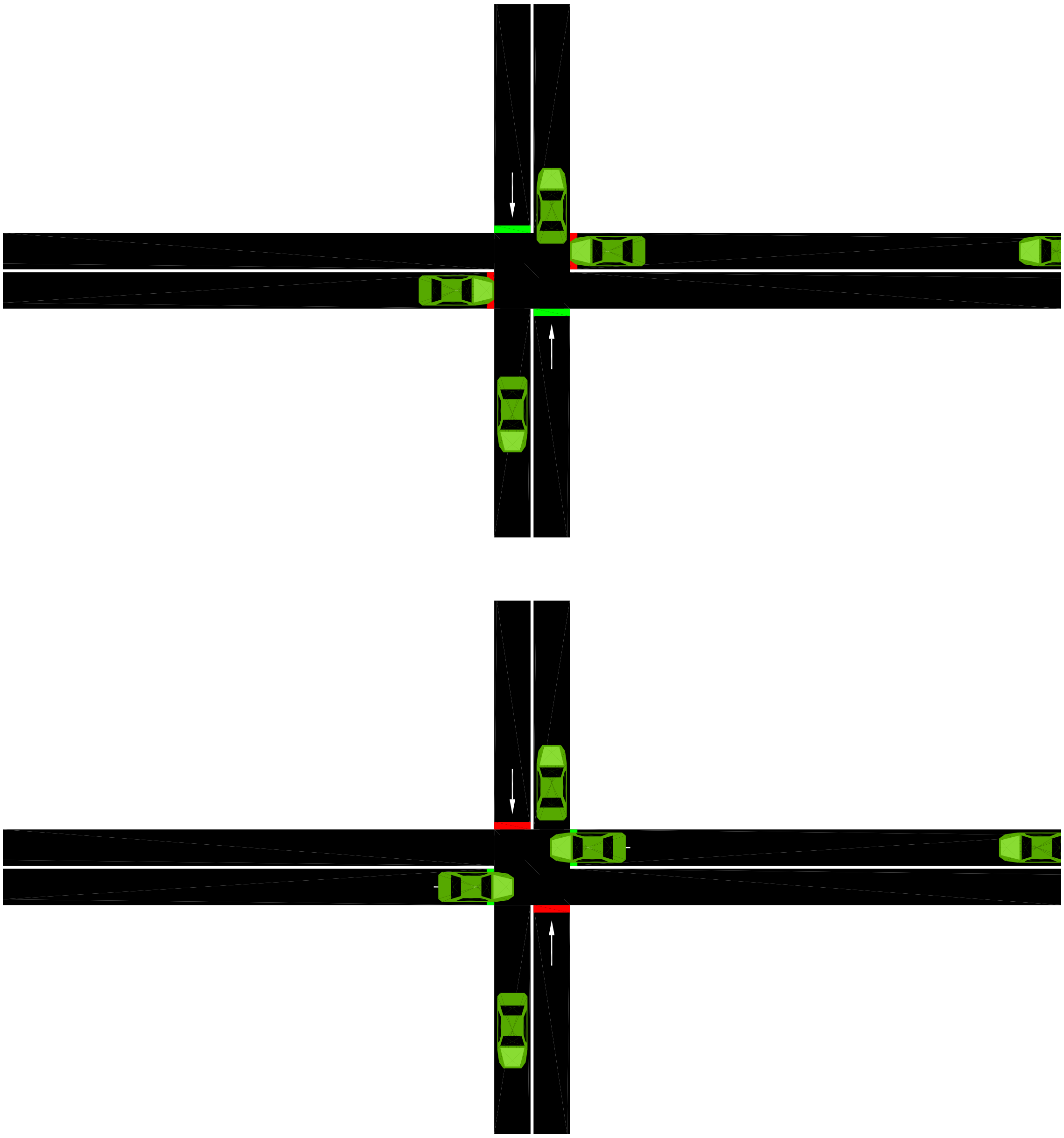


Figure 6.14: Situation in a solution of the global-MILP. The cars do not respect any security gaps when passing the intersection.

about 30 %–50 %. Certainly, these values are difficult to achieve in a real-world application. Besides the high runtimes we discuss below, the cars in the solutions of the global-MILP do not respect any security gaps when following each or when passing the intersection. Figure 6.14 visualizes an example of such a situation. Furthermore, the traffic-light's switching scheme does not incorporate any legal regulations. In fact, no traffic-light as entity with regulatory purposes is needed at all. One can argue that for future traffic with all cars driving autonomously, this would be a considerable setting. To this end, it would be possible to establish a communication among all cars on the road, e. g., with C2X-technology. Based on a consistent set of rules about right of way at intersections, autonomously driving cars could coordinate their passages over the intersection without any regulatory intervention by another unit, e. g., a traffic-light. Beyond the fact that such a scenario is hard to imagine in the near future, a reliable technical solution for regulating traffic at intersections should certainly always be able to incorporate traffic which does not communicate with the other participants and is not moving autonomously. Besides common non-autonomous cars, this can be pedestrians and cyclists. We introduced such a system, which can be seen as a combination of optimizing traffic based on recent technology and conventional traffic-light-governed right of way regulation, in Section 5.2.1 as greedy-cruise-control. In Section 6.6, we discuss experimental results for this system.

The values for simulations of the global-realistic-MILP are determined on the single-intersection network only. Moreover, we consider very loose traffic flow here making a comparison with the other methods, e. g., the RACC, obsolete. This is due to the very high complexity of the problem and the resulting runtimes, which we discuss below and are the main purpose of the experiments for the global-realistic-MILP. In particular, here, the traffic consists of 8 cars per lane and minute on average for a time horizon of 30 seconds. Regarding the waiting time, we can assess that these rules are more or less redundant for the solution.

6.5.2 Iterative Solving Algorithm

We now discuss the performance of the iterative solving algorithm developed in Section 4.5.1. The experiments differ in the methods which are used to determine the optimal solution of the global-(realistic)-MILP. In Figure 4.10, a flow chart visualizes these methods. Basically, in each set of chosen methods the collision-prevention, conflict-resolution, and model-growth are included. Let us shortly recap the major ideas:

- start with relaxed global-(realistic)-MILP for a shorter time horizon. To this end, omit all indicator variables, constraints for collision-prevention with the predecessor, and all constraints concerning trigger variables,
- add constraints preventing collisions with the predecessor if needed (collision-prevention),
- add constraints and indicator variables if conflicts on the intersection are detected (conflict-resolution),
- add constraints and indicator variables if conflicts on the intersection are detected iteratively until no further conflicts arise in the current MILP (iterative conflict-resolution),
- expand the time horizon of the current MILP if needed and add necessary variables and constraints (model-growth)
- solve the greedy-algorithm in advance and provide its solution as start heuristic to CPLEX (greedy-start).

The first set of experiments realizes collision-prevention, conflict-resolution, and model-growth. The second set differs in so far as it additionally provides CPLEX with the greedy-algorithms's optimal solution as MIP-start. It can be used by the solver as initial node in the internal branch-and-bound tree. Thus, a better bound might be available which possibly speeds up the solving process. In the third set of experiments, no greedy-solution is considered. In exchange, the more sophisticated iterative conflict-resolution (ICR) is enabled. Finally, in the fourth set, all of the methods: model-growth, collision-prevention, iterative conflict-resolution, and

Table 6.11: Obtained values network for the global-MILP on the single-intersection network.

	global-MILP	set 1[1]	set 2[2]	set 3[3]	set 4[4]
runtime[5]	-	201.27	179.81	160.12	154.48
objective	-	394 637.00	394 636.80	394 636.60	394 636.50
constraints	3 602 820	153 318.22	153 380.57	153 225.98	153 380.24
cont. variables	1 202 500	151 845.78	151 973.59	151 766.52	151 961.57
binary variables	800 000	1 499.87	1 454.4	1 489.46	1 463.45
no. outer iterations[6]	-	6.85	3.37	2.70	2.17
no. ICR iterations[7]	-	-	-	7.23	2.99

Annotations. Values are geometric means per vehicle of five testing instances.
[1] model-growth, collision-prevention, conflict-resolution, [2] additional greedy-solution as MIP-start, [3] iterative conflict-resolution instead of simple conflict-resolution,[4] additional greedy-solution as MIP-start and iterative conflict-resolution instead of simple conflict-resolution, [5] in CPU-seconds, [6] number of outer solver iterations (see Fig. 4.10), [7] number of iterations in the iterative conflict-resolution per call.

greedy-solution are considered. Tables 6.11 and 6.12 give observed values for all of the sets of experiments. In the first column, the total size of the pure global-MILP in terms of number of constraints and continuous and binary variables is presented. The full description of the polytope would be quite complex and result in very high solving times for finding an optimal solution. In fact, CPLEX was not able to determine even a single solution for the considered testing data. Note that the experimental data solved with the different methods slightly differ in their objectives. This is due to CPLEX's abortion criterion which identifies a solution as optimal if the gap between primal and dual bound is below a certain threshold of 1 % per default.

Regarding the runtimes, we can observe that in all cases the solving process benefits from the presented heuristics. Instances on the single-intersection network benefit more from applying the solution heuristics. We observe an improvement of roughly 25 % with all heuristics activated, while the biggest gap in terms of runtime between set 1 and a more complex solving approach for the four-intersections network is about 14 %. Also, the differences in runtime regarding the various

Table 6.12: Obtained values for the global-MILP on the four-intersections network.

	global-MILP	set 1[1]	set 2[2]	set 3[3]	set 4[4]
runtime[5]	-	823.67	924.16	712.79	821.33
objective	-	405 882.90	405 882.80	405 882.40	405 883.70
constraints	8 672 830	201 032.16	201 477.56	201 074.34	201 559.96
cont. variables	1 247 500	199 139.61	199 457.17	199 120.75	199 493.95
binary variables	825 000	1 827.83	1 945.52	1 873.40	1 972.17
no. outer iterations[6]	-	7.42	4.43	3.13	3.66
no. ICR iterations[7]	-	-	-	10.05	6.32

Annotations. Values are geometric means per vehicle of five testing instances.
[1] model-growth, collision-prevention, conflict-resolution, [2] additional greedy-solution as MIP-start, [3] iterative conflict-resolution instead of simple conflict-resolution,[4] additional greedy-solution as MIP-start and iterative conflict-resolution instead of simple conflict-resolution, [5] in CPU-seconds, [6] number of outer solver iterations (see Fig. 4.10), [7] number of iterations in the iterative conflict-resolution per call.

Table 6.13: Obtained values with traffic-light regulations on the single-intersection network.

	global-realistic-MILP	set 1[1]	set 3[2]	global-MILP (set 4)[3]
runtime[4]	-	4 402.48	1 897.87	3.01
objective	-	50 249.10	50 253.60	50 931.00
constraints	322 034	27 593.63	198 954.01	15 951.69
cont. variables	127 500	19 862.85	24 325.63	15 854.31
binary variables	87 500	3 100.52	6 092.47	84.12
no. outer iterations[5]	-	42.74	3.18	1.00
no. ICR iterations	-	-	85.85	1.00

Annotations. Values are geometric means per vehicle of five testing instances.
[1] model-growth, collision-prevention, conflict-resolution, [2] iterative conflict-resolution instead of simple conflict-resolution, [3] additional greedy-solution as MIP-start and iterative conflict-resolution instead of simple conflict-resolution, [4] in CPU-seconds, [5] number of outer solver iterations (see Fig. 4.10), [6] number of iterations in the iterative conflict-resolution per call.

sets of methods are not consistent in the two networks. A possible reason for this might be the different densities of traffic in both networks. Besides this issue, the iterative solving algorithm drastically decreases the problem size for both networks, making it solvable in reasonable time in the first place.

Regarding the number of constraints and continuous and binary variables which are necessary to determine the final solution, we cannot identify immense differences between the several sets of methods. In the experiments on the single-intersection network, the fastest configuration is the one with the second most added constraints and continuous variables. Apparently, the overall runtime correlates with the number of outer solver iterations, cf. Figure 4.10, mostly for the single-intersection network. ICR and greedy-start seem to influence the solving process by decreasing the number of necessary outer iterations. Adding both methods to the algorithm, i. e., using set 4, reduces the amount of outer loops even more for the single-intersection network. In fact, ICR seems to benefit from a greedy MIP-start in terms of performed iterations. In Section 6.6, we discuss runtimes of the greedy-algorithm giving us an idea of how big the percentage of runtime to determine the MIP-start is.

We will now rate the impact of the iterative solving algorithm on the global-realistic-MILP, cf. Section 4.2.5. Table 6.13 shows results for traffic on the single-intersection network. Remember that we did not perform experiments on the same testing instances as for the global-MILP. In fact, the traffic flow for the global-realistic-MILP is relatively loose. Otherwise, a solution would not be computable in reasonable time. Note that due to runtime restrictions, the bound on the cycle time is not included, cf. Table 6.9, and we did not implement the pulse interval. Thus, constraints of type (4.50), and (4.46)–(4.47) are left out. The greedy-algorithm is also not implemented for the global-realistic-MILP and cannot serve as a start heuristic. The last column shows the performance of the iterative algorithm for the global-MILP on this loose experimental data. As there is only a slight difference in the objective value between the global-MILP (column 4) and the global-realistic-MILP (columns 1 – 3), the regulations for the traffic-light's switching scheme seem not to be very restrictive for the testing data. This is also quite intuitive due to the loose character of the traffic. The average number of performed outer iterations supports this statement. Performing only a single outer

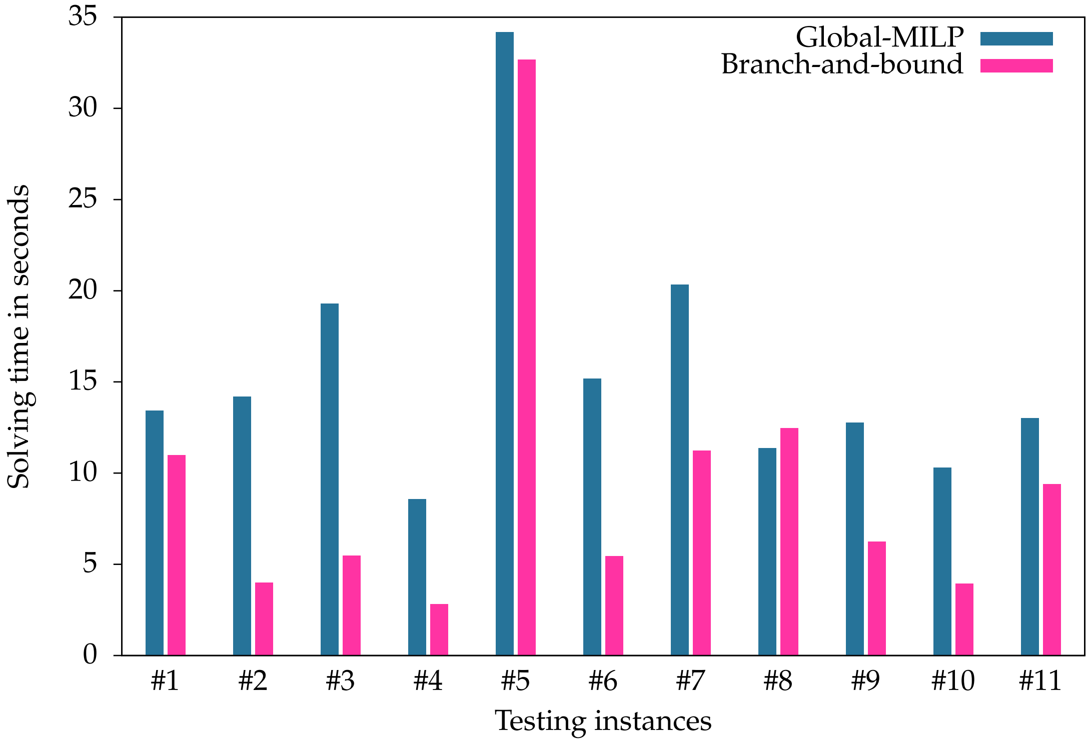


Figure 6.15: Solving times of testing instances for the global-MILP without any solving strategies and the tailored branch-and-bound algorithm.

iteration implies that only a single MILP was solved. In particular, the solution provided by the greedy-algorithm is obviously optimal for the global-MILP. Thus, the runtime in the last column is made up of the runtime for the greedy-algorithm, and a single solving process of the relaxed global-realistic-MILP, which includes binary variables added by CPLEX because of the provided initial solution.

Regarding the runtimes for solving the global-realistic-MILP, we can assess that making the problem more complex, e. g., by adding further constraints to the switching-scheme regulation or making the traffic more dense, would lead to unreasonable runtimes. The main statement of these observed values is the massive decrease in complexity which is achieved by the iterative solving algorithm. The number of constraints, which are present in the final formulation of the global-realistic-MILP is about 40 % lower for the method set 3 and reduces by 90 % for method set 1 compared to the total formulation of the MILP. Also, the number of continuous and binary variables reduces massively. Besides, the average runtime of method set 3 is less than 50 % of the runtime of method set 1, although the final MILP in the iterative process is more complex. As it is the case in the global-MILP, the number of outer iterations seems to be crucial for the runtime. This is also quite reasonable as each outer iteration depicts the necessity to solve an MILP. The difference in complexity between the global-realistic-MILP and the global-MILP for this testing data is not as big as the runtimes would suggest. At most, the number of binary variables differs, which seem to make the problem very difficult. Hence, it seems reasonable to consider binary variables particularly in the solving process. In Section 2.2, we discussed different methods to handle mixed integer programs and consecutively developed a tailored branch-and-bound process for

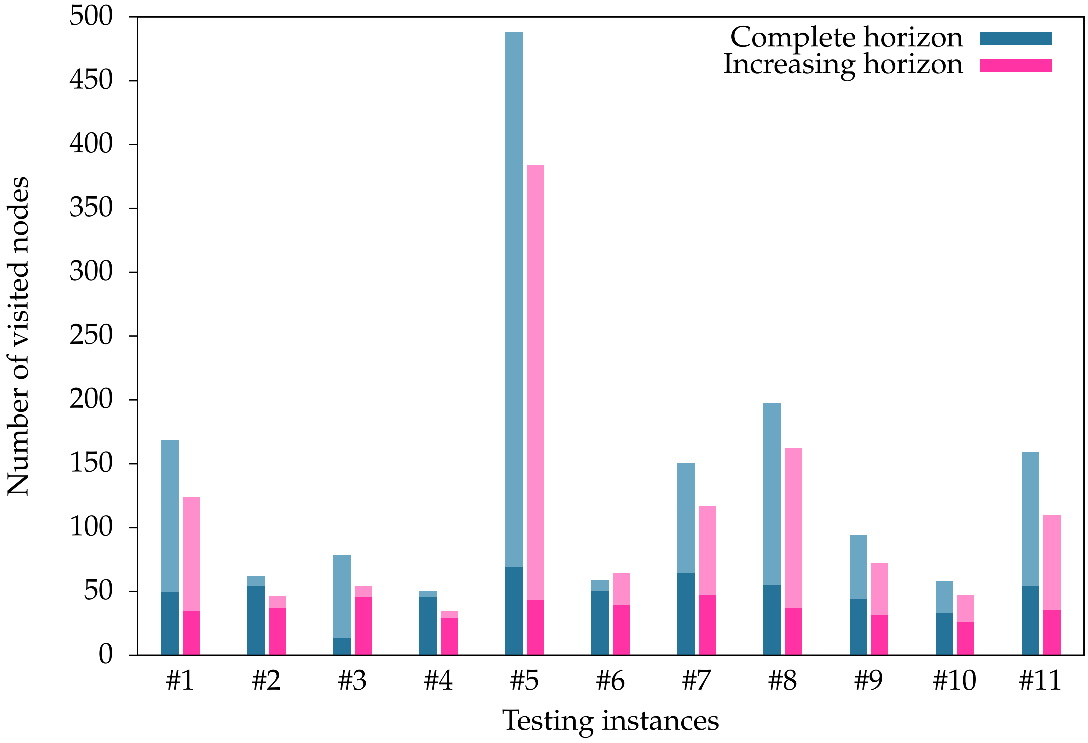


Figure 6.16: Number of visited nodes during the tailored branch-and-bound algorithm. Blue bars indicate measurements for solving an LP for the complete time horizon in each node. Red bars indicate measurements for solving LPs for smaller time horizons which grow in the course of the process. In the latter case, bounds are obtained by forward simulation. The dark bars denote in which node the optimal solution is found.

solving the global-MILP in Section 4.5.2. Subsequently, the performance of this method is presented.

6.5.3 Tailored Branch-and-Bound Process

In Section 4.5.2, we introduced a branch-and-bound algorithm that exploits the structure of the global-MILP. We will now describe experiments we performed on certain testing instances in which we measured the respective solving times for the global-MILP by CPLEX without any solving strategies and the branch-and-bound method. Furthermore, we evaluate the number of visited nodes and solving time per node for two different variants of the branch-and-bound algorithm. While each LP is solved by CPLEX, the model representation as well as the branch-and-bound's logic is implemented in AMPL. The testing instances are rather small compared to experiments above: eight to twelve cars in the whole single-intersection network with a time horizon of 100 seconds. This is due to the fact that otherwise, the global-MILP given by its complete description would not be solvable in reasonable time by CPLEX.

Figure 6.15 shows the times for solving the global-MILP without any solving strategies and the solving times for the branch-and-bound algorithm. Note that we present results for the variant of the branch-and-bound which solves an LP in each node defined on a growing subset of the whole time horizon here (called *increasing horizon method*). Additionally, the heuristic for the bounding step is enabled. For

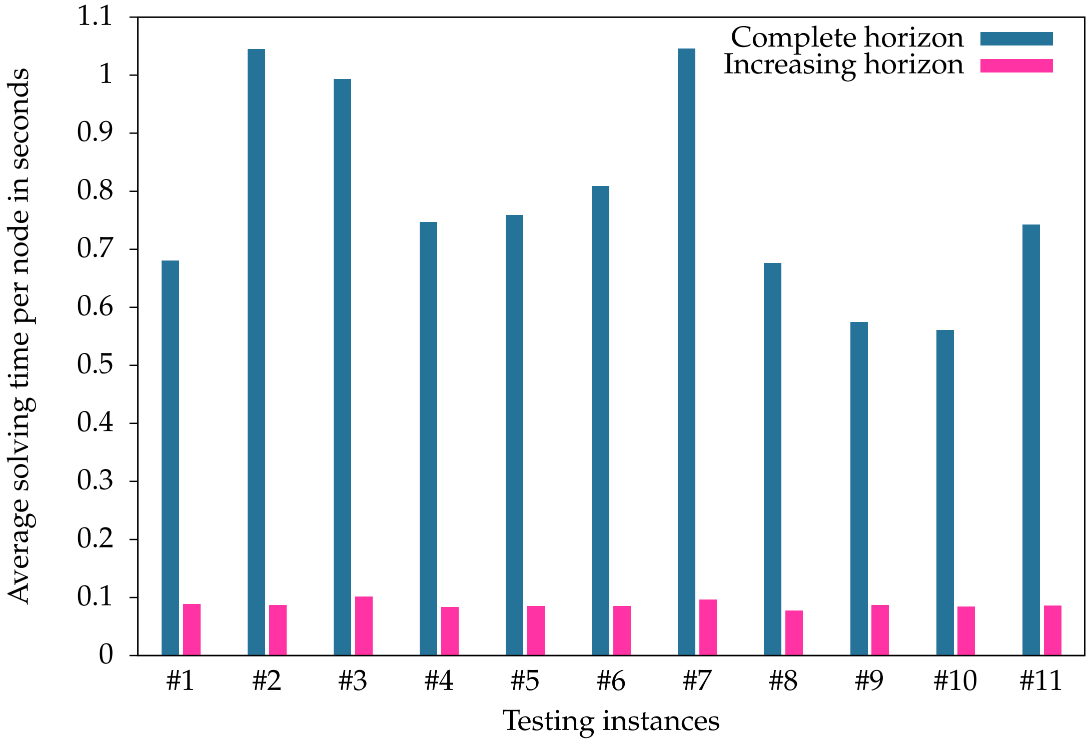


Figure 6.17: Average solving times per node during the tailored branch-and-bound algorithm. Blue bars indicate measurements for solving an LP for the complete time horizon in each node. Red bars indicate measurements for solving LPs for smaller time horizons which grow over the course of the process. In the latter case, bounds are obtained by forward simulation.

the testing instances, the optimization problems were always solved to optimality, except for two instances. We obtain an optimal solution which differs by less than 0.1 % from the optimal value, with this difference being possibly caused by numerical issues or tolerances of the solver. All instances but one benefit from the branch-and-bound method and reveal advantages in solving time of up to 72 %. Besides, the overall runtime for the branch-and-bound is in fact higher, due to additional times which are necessary for calculating the bounds and other crucial steps. However, the performance for these calculations can be strongly improved by more sophisticated implementations and data structures, e. g., by using CPLEX's C++ library.

In addition to identifying advantages of the branch-and-bound process compared to solving the global-MILP without any solving strategies solely by CPLEX, we compare the two different variants of the branch-and-bound algorithm as presented in Section 4.5.2 in terms of number of visited nodes and average solving time for each LP. Figures 6.16 and 6.17 visualize the respective outcomes. It is worth noticing that, apart from slightly fewer nodes being visited for the increasing horizon method, the average solving times are considerably higher when an LP for the whole time horizon is solved in each node. The biggest difference in average solving time per node is 91 %.

Besides the advantage considering solving times, the experiments do not provide evidence for the performance of the branch-and-bound algorithm when solving bigger problems. It also remains open for future investigations, how the

Table 6.14: Obtained values for the greedy-algorithm and global-MILP on the single-intersection network.

	greedy-algorithm	global-MILP
runtime [s]	43.78	154.48
MILP-objective	394 612.65	394 636.50
number of optimizations	170.05	-
number of optimizations per car	1.07	-
runtime per optimization [s]	0.24	-

Annotations. Values are geometric means per vehicle of five testing instances.

iterative solving algorithm would behave if the developed branch-and-bound method was incorporated.

6.6 Numerical Results for the Greedy-Algorithm

Here, we consider the performance and effects on traffic of the greedy-algorithm, cf. Section 5.1, and greedy-cruise-control developed in Section 5.2.1. The process is similar to the experiments above: the effects on traffic measured in terms of travel time, waiting time, and the environmental parameters are obtained by simulations in SUMO. To this end, the functionality of the greedy-cruise-control is implemented in Python using SUMO's interface called TraCI. The necessary calculations in CPLEX are invoked via CPLEX's Python-interface. Thus, the greedy-cruise-control can be run online in SUMO, which allows us to consider not only equipped cars, but also mixed traffic with different percentages of cars running the system. Non-equipped cars are again controlled via the car-following-model. Hence, the greedy-algorithm can be identified with the greedy-cruise-control with an equipment rate of 100 %. Furthermore, different regulations of the traffic-light's switching scheme are considered. The communication between traffic-lights and cars as well as among different cars (for avoiding collisions on a lane) is modeled as for the RACC. The range is again fixed at 200 meters. In Section 5.2.1, we suggest to disable the greedy-cruise-control as soon as a non-equipped leader is detected. For the experiments, we introduce a threshold of 40 meters. If a non-equipped car is closer than this value, the system is disabled for the succeeding equipped car. Therefore, it is possible that no optimizations at all might occur in the experiments.

Table 6.15: Obtained values of the greedy-algorithm and global-MILP on the four-intersections network.

	greedy-algorithm	global-MILP
runtime [s]	71.98	712.79
MILP-objective	405 799.44	405 882.40
number of optimizations	221.98	-
number of optimizations per car	1.38	-
runtime per optimization [s]	0.28	-

Annotations. Values are geometric means per vehicle of five testing instances.

Table 6.16: Obtained values for traffic induced by the greedy-algorithm with no traffic-light regulations and equipment rate of 100 % on the single-intersection and four-intersections network.

	single-intersection	four-intersections
travel time [s]	31.88	40.00
waiting time [s]	0.34	0.38
fuel consumption [ml]	31.14	38.21
CO_2 emissions [g]	71.15	87.29

Annotations. Values are geometric means per vehicle of five testing instances.

6.6.1 Runtimes

First, we compare the performance of the greedy-algorithm in terms of runtime and objective value to the respective values of the global-MILP. To this end, we perform calculations with CPLEX and AMPL and postprocess the solution of the greedy-algorithm in order to obtain the value of the global-MILP's objective function. Table 6.14 and Table 6.15 show the according results on the same testing instances we used for the global-MILP. For convenience, we included runtimes and objective value of the global-MILP, which we already presented above. For both networks we assess an advantage of the greedy-algorithm considering the runtime of up to 90 % on the four-intersections network, whereas the value of the global-MILP's objective function differs only slightly. In particular, the difference is less than 1 %. Certainly, this gap might increase for scenarios with more dense traffic. However, at least for the scenarios of the single-intersection network, we consider traffic densities which occur at an intersection in the city of Braunschweig during rush hour. Crucial for an assistance system, such as the greedy-cruise-control, is a reasonable runtime of the underlying optimization procedure. In both tables, we determine runtimes of only a few milliseconds per optimization. As an implementation of the greedy-cruise-control might be based on parallel calculations on each car, the runtimes support the statement that it should be suitable for practical purposes.

6.6.2 Effects on Traffic Flow

As the values of the global-MILP's objective function suggest, the traffic flow induced by the greedy-algorithm is nearly as good as the globally optimized traffic flow. Also, the environmental parameters are nearly the same for both methods. Table 6.16 shows the appropriate measurements. Note that we do not investigate any traffic regulations for traffic with an equipment rate of 100 % as these would be in a sense pointless. In contrast to this, the experiments considering the global-realistic-MILP serve for a discussion on the complexity and impact of the iterative solving algorithm.

In Table 6.17 and Table 6.18, the measurements for experiments with different percentages of equipped cars and traffic-light regulations for the greedy-cruise-control are displayed. Note that we do not consider a fixed switching scheme as for the RACC as this would not be suitable here. Instead, the green time for each traffic-light is bounded from below and above by the values given in the tables. In case that no equipped car triggered a switch of the traffic-light, the particular cycle

Table 6.17: Obtained values for traffic induced by the greedy-cruise-control on the single-intersection network with a certain percentage of equipped cars and additional regulations for the switching behavior of the traffic-lights.

	1 s | 9 – 14 s | 3 s | 2 s[1]		1 s | 24 – 29 s | 3 s | 2 s[1]	
	10 %	50 %	10 %	50 %
travel time [s]	84.75	83.43	76.00	76.06
waiting time [s]	52.48	51.28	44.16	44.03
fuel consumption [ml]	49.73	50.13	47.86	47.94
CO_2 emissions [g]	113.56	114.49	109.30	109.49
number of optimizations	2.00	19.68	1.85[2]	15.51
number of optimizations per guided car	0.15	0.25	0.07[2]	0.20

Annotations. Values are geometric means per vehicle of five testing instances.
[1] duration red-amber phase | duration green phase | duration amber phase | duration red phase, the green phase is variable as otherwise the mechanism of greedy-cruise-control would be pointless, [2] For one testing instance no optimization is performed. The number of optimizations is determined to be 0.01 as otherwise the geometric mean would be 0.

is performed while realizing the maximum bounds. This implicitly complicates the comparison to the simulations of real-world traffic, cf. Section 6.3.

Concerning the single-intersection network, we can barely determine any differences between the respective equipment rates. A relatively big gap compared to fully-equipped traffic regarding quality of traffic flow and the environmental parameters is obvious. This is possibly due to the fact that the greedy-cruise-control is disabled as soon as a non-equipped leader closer than 40 meters is detected. The low values for the number of optimizations per car support this assumption. Additionally, no traffic-light regulation at all is present for the greedy-algorithm's experiments as this would be pointless for an equipment rate of 100 %. Compared to real-world traffic, we assess improvements for the short cycle time in terms of waiting time of about 19 % and about 25 % for the environmental parameters. The latter ones also decrease for the longer cycle time, as we observe a reduction of about 12 %. In contrast, the traffic flow seems to deteriorate slightly, i. e., waiting time increases about 18 %. Reasons for this could be that the cycle times for the

Table 6.18: Obtained values for traffic induced by the greedy-cruise-control on the four-intersections network with a certain percentage of equipped cars and additional regulations for the switching behavior of the traffic-lights.

	1 s | 9 – 14 s | 3 s | 2 s[1]		1 s | 24 – 29 s | 3 s | 2 s[1]	
	10 %	50 %	10 %	50 %
travel time [s]	78.22	66.56	80.68	71.48
waiting time [s]	38.41	26.88	40.87	31.56
fuel consumption [ml]	57.19	55.12	60.05	57.03
CO_2 emissions [g]	130.06	125.90	137.14	130.24
number of optimizations	23.44	140.79	24.71	129.04
number of optimizations per guided car	1.46	1.76	1.54	1.60

Annotations. Values are geometric means per vehicle of five testing instances.
[1] duration red-amber phase | duration green phase | duration amber phase | duration red phase, the green phase is variable as otherwise the mechanism of greedy-cruise-control would be pointless.

real-world scenario do not exactly match with those from the simulations of greedy-cruise control. Also, the number of optimizations per guided car is quite low for all scenarios – and especially for the simulations with longer cycle time. This means that a lot of equipped cars did not manage to enable the greedy-cruise-control due to close preceding cars. Probably, the threshold of 40 meters is chosen too large.

Regarding the four-intersections network, we assess noticeable differences between the particular equipment ratios. Due to the loose character of traffic, the amount of equipped cars whose greedy-cruise-control is not disabled is higher as in the small network. Besides the development of travel time and waiting time when changing the equipment rate, the values for the environmental parameters also improve when the equipment rate is increased. Nonetheless, there is also a slight deterioration in terms of traffic flow and environmental parameters for the longer cycle time. Reasons might be the same as for the single-intersection network. On both networks, traffic flow benefits from shorter green times, which is consistent with the experiments for the RACC.

6.7 Comparison of Different Approaches and Future Research

We shortly recall the main concepts of the different methods which are developed in the course of this thesis:

- the RACC implements a regime-based acceleration controller for an individual car reacting solely to the actual and intended behavior of the traffic-light,
- the greedy-algorithm calculates trajectories for cars individually and negotiates times for passing the intersection with the traffic-light,
- the global-MILP models an overall optimized traffic flow for all participants that is calculated offline.

Summing up the experiments, we can assess that all methods lead to an improvement in terms of traffic flow and environmental parameters compared to nowadays real-world traffic flow – even for low equipment rates. For some of these methods, the effects of certain parameters, e. g., the equipment rate, or cycle times of the traffic-lights, are investigated. Roughly speaking, the enhancements are bigger for shorter cycle times of the traffic-lights and grow with an increase in the equipment rate, cf. Sections 6.4.3 and 6.6.2.

Figures 6.18 and 6.19 provide a visualization for the single-intersection network concerning waiting time and fuel consumption for the different methods. Remember that the big gap in the values for the greedy-algorithm is certainly due to an absence of traffic-light regulation for an equipment rate of 100 %. In terms of the average waiting time, we can assess an improvement of up to 28 % for the RACC for the single-intersection network and 23 % for the four-intersections network. A major benefit in terms of waiting time is achieved by the greedy-algorithm. In fact, a gain of at most 99 % for both networks compared to real-world traffic can be observed. Surprisingly, the overall optimized traffic flow according to the global-MILP achieves traffic flows which are just slightly better. Possibly this gap will enlarge for denser traffic.

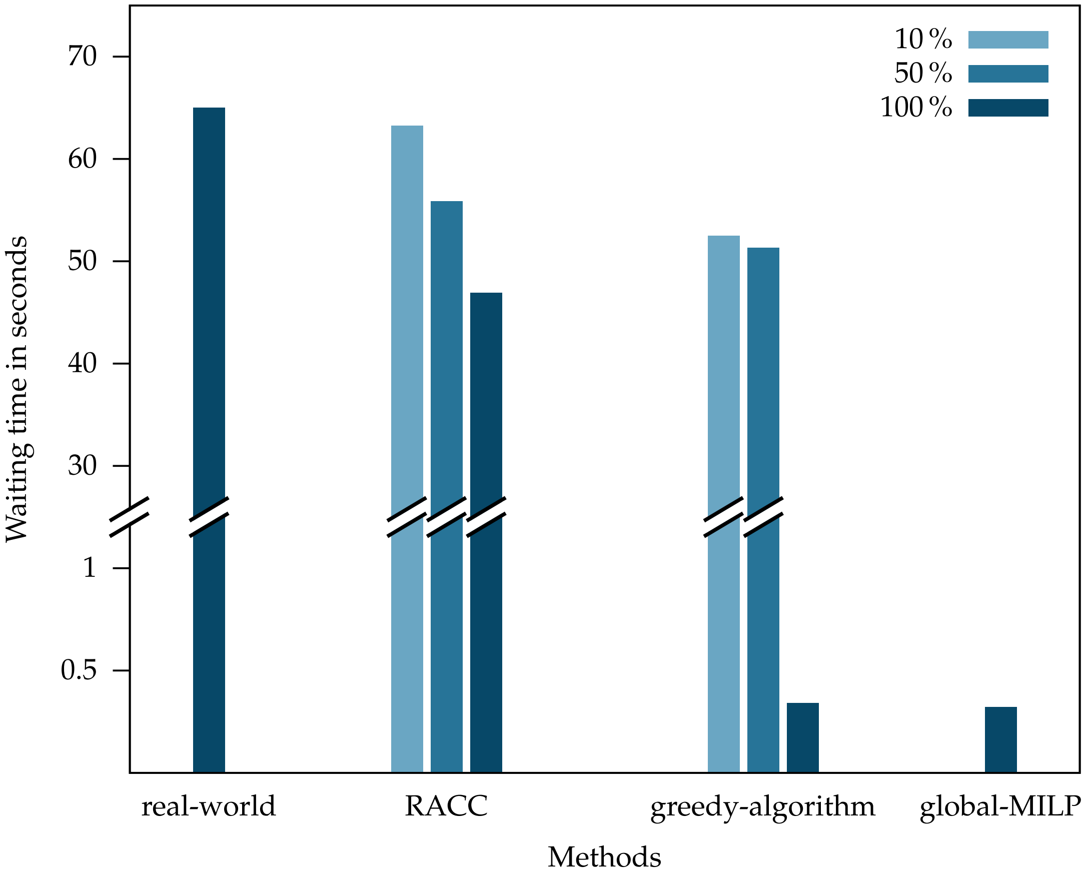


Figure 6.18: The average waiting time of cars for real-world traffic and the different methods on the single-intersection network with the shorter cycle-time if present. The different colors depict different equipment rates.

Regarding fuel consumption we can derive analogous outcomes: for the RACC, a decrease of up to 19 % and 12 % is measured during the experiments on the single- and four-intersections network, respectively. The outcomes for the greedy-algorithm and the solutions of the global-MILP are again on the same level. Improvements of up to 54 % for fuel consumption on the single-intersection network and 41 % on the four-intersections network can be observed.

The outcome of the experiments suggests a high potential for enhancements of traffic in different dimensions using cooperative systems. A major takeaway-message of this thesis is that calculating optimal solutions concerning the behavior of cars and traffic-lights from an individual point of view in contrast to a global one can result in a massive reduction of the problem's complexity. In parallel, the benefit for traffic as a whole is surprisingly close to what can be achieved by global optimization – at least for the greedy-cruise-control. Additionally, applications which consider each car individually seem to be more attractive for a contemporary introduction: participants who do not run the respective system can easily be considered, and different variants of an application, e. g., due to various implementations by different car-manufacturers, can work concurrently. Furthermore, different levels of a particular application are considerable. Examples are the RACC and the greedy-cruise-control: while a subset of cars might be able to receive messages from an infrastructural device only and react to it, another

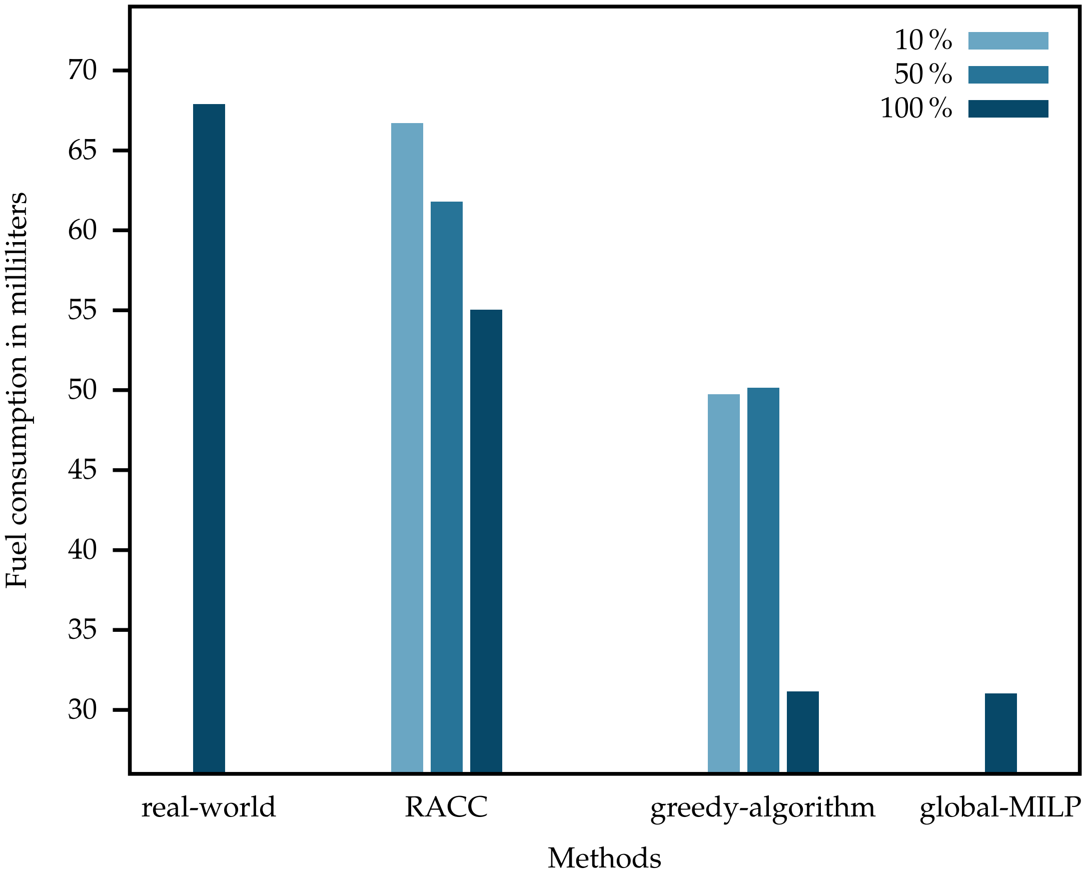


Figure 6.19: The average fuel consumption of cars for real-world traffic and the different methods on the single-intersection network with the shorter cycle-time if present. The different colors depict different equipment rates.

subset of cars might be capable of transmitting messages to the device as well and negotiate a time for a transit. Even cars which do not run any of these systems might be included in this scenario. We proposed an extension of the RACC, which could also be considered for an application based on the greedy-algorithm, where not only the actual movement of other cars is detected and reacted to. In fact, cars might exchange information about intended maneuvers among each other and incorporate these information. Some of the dead times that occur when an acceleration controller is applied could be avoided, probably leading to an even bigger improvement in traffic flow.

In contrast, a traffic-scenario which is optimized from a global point of view, as it is the case in the global-MILP, requires that all participants run the particular system – although different parametrizations could be allowed. Furthermore, some efforts have to be made in order to derive algorithms and methods which can provide solutions for a globally optimal traffic flow in reasonable time for considerable instances. Certainly, there is still potential beyond the considerations stated in this thesis. Nonetheless, the introduced global-MILP constitutes a benchmark for other algorithms or applications. Besides the discussed pros and cons of the respective systems and optimization approaches, the main advantage of suboptimal individual systems – especially the RACC – is that they can be implemented with contemporary technology and have been successfully tested.

Future research might focus on further solving strategies. It seems promising to combine the presented branch-and-bound algorithm and the iterative solving algorithm, as both methods lead to noticeable reductions in solving times during the respective experiments. One possibility is to incorporate the branch-and-bound algorithm in the solving procedure of the MILPs occurring in the iterative solving algorithm. Moreover, there certainly is still potential in improving the respective solving methods themselves, e. g., by calculating better bounds and introducing other kinds of cuts during the iterative conflict resolution.

In fact, for developing (non-optimal) driver-assistance systems or other applications, it is an important tool to value the resulting behaviors of the cars. Thus, extending the global-MILP, such that additional maneuvers, e. g., lane changes and turning maneuvers, can be modeled seems quite reasonable. Of course, also the possibility to investigate more complex networks and traffic situations, which is mainly achieved by improving the solving process, would be desirable.

References

[1] T. Achterberg, T. Koch, and A. Martin. Branching Rules Revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] G. Araniti, C. Campolo, M. Condoluci, A. Iera, and A. Molinaro. LTE for Vehicular Networking: A Survey. *Communications Magazine, IEEE*, 51(5): 148–157, 2013.

[3] B. Asadi and A. Vahidi. Predictive Cruise Control: Utilizing Upcoming Traffic Signal Information for Improving Fuel Economy and Reducing Trip Time. *IEEE Transactions on Control Systems*, 19(3):707–714, 2011.

[4] ASTM International. ASTM E2213-03, Standard Specification for Telecommunications and Information Exchange Between Roadside and Vehicle Systems - 5 GHz Band Dedicated Short Range Communications (DSRC) Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2010.

[5] E. Balas. Disjunctive Programming. *Annals of Discrete Mathematics 5*, 5:3–51, 1979.

[6] N. Bellomo and C. Dogbe. On the Modeling of Traffic and Crowds: A Survey of Models, Speculations, and Perspectives. *SIAM Review*, 53(3):409–463, 2011.

[7] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-Integer Nonlinear Optimization. *Acta Numerica*, 22:1–131, 2013.

[8] J. Betts and W. Huffman. Mesh Refinement in Direct Transcription Methods for Optimal Control. *Optimal Control Applications and Methods*, 19(1):1–21, 1998.

[9] L. Biegler. Solution of Dynamic Optimization Problems by Successive Quadratic Programming and Orthogonal Collocation. *Computers & Chemical Engineering*, 8(3-4):243–247, 1984.

[10] H. Bock and K. Plitt. A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems. In *Proceedings of the 9th Ifac World Congress*, page 242–247. Pergamon Press, 1984.

[11] P. Bonami, A. Lodi, A. Tramontani, and S. Wiese. On Mathematical Programming with Indicator Constraints. *Mathematical Programming*, 151(1):191–223, 2015.

[12] B. Borchers and J. Mitchell. An Improved Branch and Bound Algorithm for Mixed Integer Nonlinear Programs. *Computers & Operations Research*, 21(4): 359–367, 1994.

[13] P. Brucker. *Scheduling Algorithms*. Springer, Berlin, 2004.

[14] M. Bussieck and A. Prüssner. Mixed-Integer Nonlinear Programming. *SIAG/OPT Newsletter: Views & News*, 14(1):19–22, 2003.

[15] V. Chvátal. Edmonds Polytopes and Weakly Hamiltonian Graphs. *Mathematical Programming*, 5(1):29–40, 1973.

[16] W. Cook, R. Kannan, and A. Schrijver. Chvátal Closures for Mixed Integer Programming Problems. *Mathematical Programming*, 47(1-3):155–174, 1990.

[17] R. Dakin. A Tree-Search Algorithm for Mixed Integer Programming Problems. *The Computer Journal*, 8(3):250–255, 1965.

[18] M. Diehl, H. Bock, and J. Schlöder. A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005.

[19] K. Dresner and P. Stone. A Reservation-Based Multiagent System for Intersection Control. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 530–537, 2004.

[20] K. Dresner and P. Stone. Multiagent Traffic Management: An Improved Intersection Control Mechanism. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 471–477, 2005.

[21] K. Dresner and P. Stone. Sharing the Road: Autonomous Vehicles Meet Human Drivers. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1263–1268, 2007.

[22] Elektrobit Automotive GmbH. Automotive Data and Time-Triggered Framework, 20.09.2016. URL `https://www.elektrobit.com/products/eb-assist/adtf/`.

[23] J. Erdmann. Combining Adaptive Junction Control with Simultaneous Green-Light-Optimal-Speed-Advisory. In *Proceedings of the 5th International Symposium on Wireless Vehicular Communications (WiVeC)*, page 1–5, 2013.

[24] M. Fischetti, A. Lodi, and A. Tramontani. On the Separation of Disjunctive Cuts. *Mathematical Programming*, 128(1-2):205–230, 2011.

[25] Forschungsgesellschaft für Straßen- und Verkehrswesen e.V. *Handbuch für die Bemessung von Straßenverkehrsanlagen*. FGSV-Verlag, Köln, 2001.

[26] Forschungsgesellschaft für Straßen- und Verkehrswesen e.V. *Richtlinien für Lichtsignalanlagen: Lichtzeichenanlagen für den Straßenverkehr*. FGSV-Verlag, Köln, 2010.

[27] A. Frangioni and C. Gentile. Perspective Cuts for a Class of Convex 0–1 Mixed Integer Programs. *Mathematical Programming*, 106(2):225–236, 2006.

[28] T. Frankiewicz, L. Schnieder, and F. Köster. Application Platform for Intelligent Mobility Test Site Architecture and Vehicle2X Communication Setup. In *Proceedings of the 19th ITS World Congress*, 2012.

[29] J. Frasch. Parallel Algorithms for Optimization of Dynamic Systems in Real-Time. PhD Thesis, 2014.

[30] B. Friedrich, P. Wagner, W. Niebel, A. Herrmann, S. Naumann, O. Bley, O. Kutzner, M. Maurer, F. Saust, T. Schüler, H. Poppe, M. Junge, and J. Langenberg. *KOLINE: Kooperative und optimierte Lichtsignalsteuerung in städtischen Netzen: Schlussbericht zum Forschungsprojekt, Förderkennzeichen 19P9002, gefördert vom Bundesministerium für Wirtschaft und Technologie*. 2013.

[31] R. Gallager. *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2013.

[32] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A series of books in the mathematical sciences. Freeman, New York, 1979.

[33] N. Gartner, J. Little, and H. Gabbay. Optimization of Traffic Signal Settings by Mixed-Integer Linear Programming: Part I: The Network Coordination Problem. *Transportation Science*, 9(4):321–343, 1975.

[34] N. Gartner, J. Little, and H. Gabbay. Optimization of Traffic Signal Settings by Mixed-Integer Linear Programming: Part II: The Network Synchronization Problem. *Transportation Science*, 9(4):344–363, 1975.

[35] C. Gershenson. Self-Organizing Traffic Lights. *arXiv preprint nlin/0411066v2*, 2008.

[36] J. Ghosh. Batch Scheduling to Minimize Total Completion Time. *Operations Research Letters*, 16(5):271–275, 1994.

[37] A. Giridhar and P. Kumar. Scheduling Automated Traffic on a Network of Roads. *IEEE Transactions on Vehicular Technology*, 55(5):1467–1474, 2006.

[38] S. Glaser, B. Vanholme, S. Mammar, D. Gruyer, and L. Nouveliere. Maneuver-Based Trajectory Planning for Highly Autonomous Vehicles on Real Road With Traffic and Driver Interaction. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):589–606, 2001.

[39] O. Günlük and J. Linderoth. Perspective Reformulations of Mixed Integer Nonlinear Programs with Indicator Variables. *Mathematical Programming*, 124(1-2):183–205, 2010.

[40] R. Gomory. Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[41] I. Grossmann and S. Lee. Generalized Convex Disjunctive Programming: Nonlinear Convex Hull Relaxation. *Computational Optimization and Applications*, 26(1):83–100, 2003.

[42] I. Grossmann and J. Ruiz. Generalized Disjunctive Programming: A Framework for Formulation and Alternative Algorithms for MINLP Optimization. In *Mixed Integer Nonlinear Programming*, pages 93–115. Springer, New York, 2012.

[43] M. Grötschel. *The Sharpest Cut: The Impact of Manfred Padberg and His Work*. Society for Industrial and Applied Mathematics, Philadelphia, 2004.

[44] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, New York and Berlin, 1988.

[45] S. Göttlich, M. Herty, and U. Ziegler. Modeling and Optimizing Traffic Light Settings in Road Networks. *Computers & Operations Research*, 55:36–51, 2015.

[46] S. Göttlich, A. Potschka, and U. Ziegler. Partial Outer Convexification for Traffic Light Optimization in Road Networks. *Optimization-Online e-prints*, 2015. URL `http://www.optimization-online.org/DB_FILE/2015/11/5198.pdf`.

[47] C. Hargraves and S. Paris. Direct Trajectory Optimization Using Nonlinear Programming and Collocation. *Journal of Guidance, Control, and Dynamics*, 10 (4):338–342, 1987.

[48] A. Hetzel. Sensorbasierte Auswertung von Objektdaten an Lichtsignalanlagen und Kalibrieren des Fahrzeugfolgemodells Intelligent Driver Model (IDM): Diploma Thesis, 2015.

[49] Y. Hong, J. Kim, J. Kwangson, and C. Park. Estimation of Optimal Green Time Simulation Using Fuzzy Neural Network. In *Proceedings of 8th International Fuzzy Conference*, volume 2, page 761–766. 1999.

[50] B. Houska, H. Ferreau, and M. Diehl. ACADO Toolkit: An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[51] M. Jung. Relaxations and Approximations for Mixed-Integer Optimal Control. PhD Thesis, 2013.

[52] K. Katsaros, R. Kernchen, M. Dianati, and D. Rieck. Performance study of a Green Light Optimized Speed Advisory (GLOSA) Application Using an Integrated Cooperative ITS Simulation Platform. In *Proceedings of the 7th International Wireless Communications and Mobile Computing Conference*, 2011.

[53] F. Kehrle, J. Frasch, C. Kirches, and S. Sager. Optimal Control of Formula 1 Race Cars in a VDrift Based Virtual Environment. *IFAC Proceedings Volumes*, 44(1):11907–11912, 2011.

[54] A. Kesting, M. Treiber, and D. Helbing. Enhanced Intelligent Driver Model to Access the Impact of Driving Strategies on Traffic Capacity. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4585–4605, 2010.

[55] U. Kiencke and L. Nielsen. *Automotive Control Systems: For Engine, Driveline, and Vehicle*. Springer, Berlin and Heidelberg, 2005.

[56] C. Kirches. Fast Numerical Methods for Mixed–Integer Nonlinear Model–Predictive Control. PhD Thesis, 2010.

[57] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker. Recent Development and Applications of SUMO-Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3-4):128–138, 2012.

[58] F. Kranke, S. Sorgatz, F. Weinert, and H. Poppe. *Urbaner Raum: Benutzergerechte Assistenzsysteme und Netzmanagement: Schlussbericht der Volkswagen AG zum Forschungsprojekt UR:BAN, Förderkennzeichen 19P11007S, gefördert vom Bundesministerium für Wirtschaft und Energie*. 2016.

[59] S. Krauß. Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics. PhD Thesis, 1998.

[60] A. Land and A. Doig. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3):497, 1960.

[61] A. Land and S. Powell. Computer Codes for Problems of Integer Programming. *Annals of Discrete Mathematics 5*, 5:221–269, 1979.

[62] E. Lawler and D. Wood. Branch-and-Bound Methods: A Survey. *Operations Research*, 14(4):699–719, 1966.

[63] W. Lawrenz and N. Obermöller. *CAN: Controller Area Network: Grundlagen, Design, Anwendungen, Testtechnik*. VDE-Verlag, Berlin, 2011.

[64] M. Lübbecke. Column Generation. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[65] M. Lübbecke and J. Desrosiers. Selected Topics in Column Generation. *Operations Research*, 53(6):1007–1023, 2005.

[66] S. Le Vine, A. Zolfaghari, and J. Polak. Autonomous Cars: The Tension Between Occupant Experience and Intersection Capacity. *Transportation Research Part C: Emerging Technologies*, 52:1–14, 2015.

[67] J. Lee and S. Leyffer, editors. *Mixed Integer Nonlinear Programming*. The IMA Volumes in Mathematics and its Applications. Springer, New York, 2012.

[68] D. Leineweber. *Efficient Reduced SQP Methods for the Optimization of Chemical Processes Described by Large Sparse DAE Models*, volume 613 of *Fortschritt-Berichte VDI Reihe 3, Verfahrenstechnik*. VDI-Verlag, Düsseldorf, 1999.

[69] J. Lenstra, A. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Ann. of Discrete Math.*, 1:343–362, 1977.

[70] S. Leyffer. Deterministic Methods for Mixed Integer Nonlinear Programming. PhD Thesis, 1993.

[71] S. Leyffer. Integrating SQP and Branch-and-Bound for Mixed Integer Nonlinear Programming. *Computational Optimization and Applications*, 18(3):295–309, 2001.

[72] M. Lighthill and G. Whitham. On Kinematic Waves. II. A Theory of Traffic Flow on Long Crowded Roads. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 229(1178):317–345, 1955.

[73] S. Lin, B. de Schutter, Y. Xi, and H. Hellendoorn. Fast Model Predictive Control for Urban Road Networks via MILP. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):846–856, 2011.

[74] J. Linderoth and M. Savelsbergh. A Computational Study of Search Strategies for Mixed Integer Programming. *INFORMS Journal on Computing*, 11(2): 173–187, 1999.

[75] S. Lämmer and D. Helbing. Self-Control of Traffic Lights and Vehicle Flows in Urban Road Networks. *Journal of Statistical Mechanics: Theory and Experiment*, (4):4–19, 2008.

[76] H. Mir and F. Filali. LTE and IEEE 802.11p for Vehicular Networking: A Performance Evaluation. *EURASIP Journal on Wireless Communications and Networking*, (89):1–15, 2014.

[77] M. Modsching, R. Kramer, and K. ten Hagen. Field Trial on GPS Accuracy in a Medium Size City: The Influence of Built-Up. In *3rdWorkshop on Positioning, Navigation and Communication*, page 209–218, 2006.

[78] I. Nowak and S. Vigerske. LaGO: A (Heuristic) Branch and Cut Algorithm for Nonconvex MINLPs. *Central European Journal of Operations Research*, 16 (2):127–138, 2008.

[79] G. de Nunzio, C. Wit, P. Moulin, and D. Di Domenico. Eco-Driving in Urban Traffic Networks Using Traffic Signals Information. *International Journal of Robust and Nonlinear Control*, 2015.

[80] M. Osborne. On Shooting Methods for Boundary Value Problems. *Journal of Mathematical Analysis and Applications*, 27(2):417–433, 1969.

[81] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, 2012.

[82] L. Pontrjagin, V. Boltyanskii, and R. Gamkrelidze. *Mathematical Theory of Optimal Processes*. Gordon and Breach, New York, 1986.

[83] K. Preuk and E. Stemmler. Interdisziplinäre Bewertung der Auswirkung eines Fahrers mit Ampelassistenz auf nicht-ausgestattete Fahrer. *VDI Wissenforum*, 2015.

[84] R. Raman and I. Grossmann. Modelling and Computational Techniques for Logic Based Integer Programming. *Computers & Chemical Engineering*, 18(7): 563–578, 1994.

[85] J. Rawlings and D. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publ, Madison, 2009.

[86] P. Richards. Shock Waves on the Highway. *Operations Research*, 4(1):42–51, 1956.

[87] Robert Bosch GmbH. *Fachwörterbuch Kraftfahrzeugtechnik: Autoelektrik, Autoelektronik, Motor-Management, Fahrsicherheitssysteme*. Vieweg, Wiesbaden, 2005.

[88] R. Russell and L. Shampine. A Collocation Method for Boundary Value Problems. *Numerische Mathematik*, 19(1):1–28, 1972.

[89] S. Sager. *Numerical Methods for Mixed-Integer Optimal Control Problems*. Der andere Verlag, Tönning and Lübeck and Marburg, 2005.

[90] J. Sanchez, M. Galan, and E. Rubio. Genetic Algorithms and Cellular Automata: A New Architecture for Traffic Light Cycles Optimization. In *Proceedings of the Congress on Evolutionary Computation*, page 1668–1674, 2004.

[91] J. Sanchez, M. Galan, and E. Rubio. Applying a Traffic Lights Evolutionary Optimization Technique to a Real Case: Las Ramblas Area in Santa Cruz de Tenerife. *IEEE Transactions on Evolutionary Computation*, 12(1):25–40, 2008.

[92] C. Schießl and K. Oeltze. Benefits and Challenges of Multi-Driver Simulator Studies. *IET Intelligent Transport Systems*, 9(6):618–625, 2015.

[93] L. Schnieder and R. Krenkel. Betreibermodell einer Forschungsinfrastruktur für die Entwicklung intelligenter Mobilitätsdienste im realen Verkehrsumfeld. In *16. Symposium Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel (AAET)*, pages 108–116. 2015.

[94] B. de Schutter. Optimizing Acyclic Traffic Signal Switching Sequences Through an Extended Linear Complementarity Problem Formulation. *European Journal of Operational Research*, 139(2):400–415, 2002.

[95] B. de Schutter and B. de Moor. Optimal Traffic Light Control for a Single Intersection. *European Journal of Control*, 4(3):260–276, 1998.

[96] S. Sorgatz, F. Kranke, and H. Poppe. Der Kreuzungslotse von Volkswagen: Urbane Assistenz für einen verbesserten Verkehrsfluss und Fahrkomfort. *Straßenverkehrstechnik*, 60(7):425–432, 2016.

[97] R. Tachet, P. Santi, S. Sobolevsky, L. Reyes-Castro, E. Frazzoli, D. Helbing, C. Ratti, and T. Tang. Revisiting Street Intersections Using Slot-Based Systems. *PLOS ONE*, 11(3):e0149607, 2016.

[98] K. Tan, M. Khalid, and R. Yusof. Intelligent Traffic Lights Control by Fuzzy Logic. *Malaysian Journal of Computer Science*, 9(2), 1996.

[99] M. Treiber, A. Hennecke, and D. Helbing. Congested Traffic States in Empirical Observations and Microscopic Simulations. *Physical Review E*, 62 (2):1805–1824, 2000.

[100] T. Tsang, D. Himmelblau, and T. Edgar. Optimal Control via Collocation and Non-Linear Programming. *International Journal of Control*, 21(5):763–768, 1975.

[101] Umweltbundesamt. *Handbuch Emissionsfaktoren des Straßenverkehrs Version 3.1*. INFRAS, Zürich, 2010.

[102] G. Vainikko. The Convergence of the Collocation Method for Non-Linear Differential Equations. *USSR Computational Mathematics and Mathematical Physics*, 6(1):47–58, 1966.

[103] M. VanMiddlesworth, K. Dresner, and P. Stone. Replacing the Stop Sign: Unmanaged Intersection Control for Autonomous Vehicles. In *Proceedings of the The 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1413–1416, 2008.

[104] Verband der Automobilindustrie e.V. simTD - Sichere Intelligente Mobilität Testfeld Deutschland, 2014. URL `http://www.simtd.de/index.dhtml/deDE/index.html`.

[105] A. Vogel, C. Goerick, and W. von Seelen. Evolutionary Algorithms for Optimizing Traffic Signal Operation. In *Proceedings of the European Symposium on Intelligent Techniques (ESIT)*, page 83–91, 2000.

[106] M. Wada, T. Yendo, T. Fujii, and M. Tanimoto. Road-to-Vehicle Communication Using LED Traffic Light. In *Proceeding of the Intelligent Vehicles Symposium*, pages 601–606, 2005.

[107] Y. Wang, B. de Schutter, T. van den Boom, and B. Ning. Optimal Trajectory Planning for Trains Under Fixed and Moving Signaling Systems Using Mixed Integer Linear Programming. *Control engineering Practice*, 22:44–56, 2014.

[108] A. Wächter and L. Biegler. On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming*, 106(1):25–57, 2006.

[109] Wes Office. UR:BAN - Benutzergerechte Assistenzsysteme und Netzmanagement, 2016. URL `http://urban-online.org`.

[110] M. Wiering, J. van Veenen, J. Vreeken, and A. Koopman. Intelligent Traffic Light Control. *Institute of Information and Computing Sciences. Utrecht University*, 2004.

[111] L. Wolsey. Mixed Integer Programming. In Benjamin W. Wah, editor, *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Hoboken, 2007.

[112] Zhang, Y. Malikopoulos, A. and C. Cassandras. Optimal Control and Coordination of Connected and Automated Vehicles at Urban Traffic Intersections. In *Proceedings of the American Control Conference*, pages 6227–6232, 2016.