[parts that will probably change and comments are in blue]

Coordination of Autonomous Traffic in Networks of Intersections using Decomposition Techniques and Machine Learning-Based Scheduling

Jeroen van Riel

Combined Master's Thesis

Industrial and Applied Mathematics
Computer Science and Engineering

Supervisors

Marko Boon
Stella Kapodistria
Mykola Pechenizkiy
Danil Provodin

October 2025

Eindhoven University of Technology

Abstract

Coordination schemes for autonomous vehicles offer clear advantages over conventional traffic management. When considering the problem of optimizing the joint trajectories of individual autonomous vehicles, we identify some fundamental computational challenges. Ordering and prioritization decisions arise naturally in this context, since infrastructure must be shared among multiple vehicles. The combinatorial complexity of these decisions rules out simple enumeration procedures for finding optimal solutions. Furthermore, the infinite-dimensional nature of trajectories complicates the application of standard combinatorial optimization techniques.

To explore these challenges, we consider a simple model of autonomous traffic coordination in a network of intersections, where determining the order in which vehicles cross each intersection on their route is the main combinatorial component. To enable precise and interpretable results, we consider a traffic system where all vehicles are autonomous and homogeneous in terms of dimensions and dynamics. Each vehicle follows its own fixed route through the network and its speed profile is controlled by a central coordination algorithm, assuming perfect communication without loss or delay. In this setting, we stage the traffic coordination task as a trajectory optimization problem.

For a single isolated intersection, we show that—under certain assumptions—the optimization problem admits a bilevel decomposition into (i) an upper-level scheduling problem of determining crossing times at the intersection and (ii) a set of lower-level trajectory optimization problems. The lower-level problems can be solved relatively efficiently using standard numerical methods. The upper-level problem can be solved using mixed-integer linear programming, but this does not scale very well to more vehicles.

To address this scalability issue, we investigate how machine learning can be employed to develop smart heuristics for the upper-level problem. We show that crossing time schedules can be interpreted as a sequence of decisions. We propose a sequence model with a simple recurrent neural network parameterization, which we fit on optimal schedules obtained through mixed-integer linear programming. The resulting heuristic achieves an optimality gap of up to 2%, with near-instant evaluation time compared to mixed-integer linear programming. However, this supervised learning regime still depends on our ability to obtain optimal schedules for training. To overcome this limitation, we show that we can use reinforcement learning to train the sequence model from scratch. Both training regimes produce heuristics that outperform a commonly used greedy scheduling heuristic.

Extending our methodology to multiple intersections introduces additional complexity. In particular, it becomes more involved to guarantee feasibility of the lower-level problems due to the finite capacity of lanes between intersections. We find that this feasibility is characterized by a system of linear inequalities in terms of the crossing times, which enables the bilevel decomposition to extend to the network case. Consequently, the corresponding scheduling problem can again be approached as a sequence modeling task. Compared to the single intersection case, the results show that this case is much harder. A possible explanation is found in the redundancy of the sequence encoding of schedules—multiple sequences correspond to the same schedule—which makes learning inefficient. We find that the order in which the model is evaluated matters for the final performance, confirming previous observations and motivating further research on alternative encodings.

Contents

1 Introduction **5**
- 1.1 Motivation and challenges . . . 5
- 1.2 Literature review . . . 7
- 1.3 Project goals and outline . . . 8

Part I – Isolated Intersection

2 Isolated intersection scheduling **12**
- 2.1 Intersection model . . . 13
- 2.2 From joint optimization to bilevel decomposition . . . 17
  - 2.2.1 Joint optimization using direct transcription . . . 17
  - 2.2.2 Bilevel decomposition . . . 18
  - 2.2.3 Delay minimization . . . 21
- 2.3 Crossing time scheduling . . . 22
  - 2.3.1 Branch-and-bound for MILP . . . 22
  - 2.3.2 Problem-specific cutting planes . . . 23
- 2.4 Notes and references . . . 27

3 Scheduling as sequence learning **29**
- 3.1 Machine learning for combinatorial optimization . . . 30
- 3.2 Sequence representation of optimal schedules . . . 36
- 3.3 Sequential schedule construction . . . 40
  - 3.3.1 Policy optimization . . . 41
  - 3.3.2 Policy parameterizations . . . 43
- 3.4 Experimental results . . . 45
- 3.5 Notes and references . . . 48
  - 3.5.1 Local search . . . 48

Part II – Networks of Intersections

4 Learning to schedule in networks **51**
- 4.1 Notes and references . . . 52

5 Capacitated lanes **53**
- 5.1 Model formulation . . . 53
- 5.2 Single vehicle with arbitrary lead vehicle constraint . . . 55
  - 5.2.1 Necessary conditions . . . 55
  - 5.2.2 Sufficient conditions . . . 57
  - 5.2.3 Deceleration boundary . . . 58
  - 5.2.4 Smoothing procedure . . . 59
  - 5.2.5 Upper boundary solution . . . 62
- 5.3 Lane planning feasibility . . . 62
- 5.4 Optimal solution for haste objective . . . 65
  - 5.4.1 Connecting partial trajectories . . . 66
  - 5.4.2 Algorithm . . . 68

5.5 Feasibility characterization 68
5.6 Notes and references 69

6 Conclusion and discussion **70**

Bibliography **74**

Appendix

A Feasible configurations for single intersection model **76**

B Job shop scheduling **78**

C Neural combinatorial optimization **81**

D Reinforcement learning **84**
D.1 Stationary distribution for finite episodes. 85
D.2 Policy gradient estimation 87

E Miscellaneous **89**

Chapter 1

Introduction

1.1 Motivation and challenges

Given the ongoing advances in self-driving vehicles and wireless communication, it is very natural to study how these new technologies can be applied to enable network-wide traffic coordination. Some of the potential benefits of coordinating the motion of groups of autonomous vehicles are increased network throughput, reduced energy consumption and better guarantees on safety in terms of avoiding dangerous situations.

Hence, it is not surprising that coordination of autonomous vehicles with communication has been studied from a wide range of perspectives. To give some example, a typical distinguishing feature is the level of organization at which coordination is considered [39]. A probably well-known example of *local coordination* is platooning of vehicles, where the aim is to lower energy consumption by reducing aerodynamic resistance. It has been shown that platooning can also result in a more efficient use of intersections. On a larger scale, *global coordination* methods like dynamic route optimization have been proposed to reduce travel delay for all vehicles in the network.

Apart from the organization, the coordination problem has many more modeling aspects. For example, one may think of heterogeneous vehicles—in terms of dynamics or priority—different models of centralized/decentralized communication between vehicles or with the infrastructure, under different guarantees on reliability; complex road topology, curved lanes, merging lanes; implications of mixing human traffic with fully autonomous traffic. We will discuss some common modeling aspects in the literature review below.

Coordination has been considered from various angles, but some fundamental challenges in ensuring safety and efficiency remain poorly understood, even in stylized models with the bare minimal modeling elements such as vehicle dynamics and collision-avoidance constraints. This brings us to the scope of this project. In order to better understand these inherent difficulties and to enable us to obtain concrete interpretable results, we focus on *fully automated mobility systems*, in which all vehicles are assumed to have autonomous driving and communication capabilities. Since we are specifically interested in coordination across multiple vehicles, we assume that the lower-level steering control of individual vehicles is taken care of; we assume that vehicles can follow planned trajectories perfectly. Even though this model does not capture many of the intricacies of our current road network, there are many examples in which such a model is appropriate. For example, think of autonomous vehicles being deployed in so-called confined sites, like ports, mines or quarries; but also warehouses and manufacturing sites, or even indoor farming installations. Moreover, even in the context of urban road traffic, stylized models are relevant, because they provide upper bounds on what we can expect to gain in more realistic situations with, e.g., decentralized control, random failures and problems like communication latency and noisy information.

We will take an *optimization-based perspective* on the coordination problem, in which the joint trajectories of the vehicles in the system are our main decisions. In this context, there are some natural requirements and objectives, which are often conflicting with each other. First of all, guarantees of safety and collision avoidance are fundamental issues. From the perspective of promoting traffic throughput of the system, minimizing travel delay is a very natural

objective. However, there is a tension with the goal of limiting fuel and energy consumption. In multi-user systems like traffic networks, attention to fairness is also of central importance. Other objectives like passenger comfort, e.g., with smooth acceleration/deceleration, have also been considered in this context. Apart from the difficulty of formulating the optimization problem and its objective function, the following challenges have been identified with respect to the algorithmic aspects of solving such optimization problems.

Complex combinatorial decisions. In traffic coordination, combinatorial problems arise naturally from ordering constraints—deciding which vehicle should move first across shared resources such as intersections, merges, or lanes, while taking into account trade-offs between throughput, fairness, and safety. Traditionally, such problems are resolved by implementing traffic lights or static rules for yielding. Autonomous connected vehicles provide us with better ways of handling these conflicts dynamically.

For example, we will focus specifically on the order in which vehicles cross intersections, which can also be thought of as grouping vehicles into platoons and then deciding in which order these platoons are allowed to cross the intersection. Instead of relying on fixed traffic signal phases, we can optimize this decision for the current traffic situation. Other examples include determining the optimal order of vehicles in a platoon that is approaching an intersection with dedicated lanes for turning. On a higher level, determining the routes that vehicles take is a classical combinatorial optimization problem, think for example of the traveling salesman problem.

Even when vehicle routes are fixed, such ordering decisions at one intersection directly influence the arrival process at nearby intersections, hence these decisions are coupled across the network. Due to this coupling, finding the optimal crossing plan becomes harder when larger networks are considered, because of the exponential growth of the space of feasible solutions. This issue is further complicated by the real-time requirement that real traffic systems impose on the computation time.

Continuous-time trajectories under constraints. We are dealing with the optimization of vehicle trajectories, which is in general an infinite-dimensional problem. Such problems can be approached in an end-to-end fashion by *direct transcription* to an equivalent mixed-integer optimization problem, which can be solved using off-the-shelf solvers. This method can be used to compute optimal trajectories up to any precision, by choosing a fine enough time discretization. However, it is exactly this time discretization that causes prohibitive growth of the number of variables with respect to the size of the network and the number of vehicles, so this method is only useful for relatively small problem instances. Moreover, in the context of traffic systems, there are complex constraints and safety requirements that need to be handled with care. Furthermore, optimization methods would benefit a lot from closed-form expressions of trajectories, which is in principle possible in some situations, but the appearance of constraints makes this notoriously difficult in practice.

Joint optimization is hard. There are lots of well-understood solution techniques for purely combinatorial problems. However, in the current context, the combinatorial part of the problem cannot be considered in isolation. The feasibility and cost of each solution to the combinatorial part depends in a non-trivial way on the lower-level continuous-time dynamics and interactions between vehicles. In other words, simultaneously optimizing the combinatorial part and the continuous-time part, to which we refer as *joint optimization*, poses major algorithmic challenges. In the literature, heuristics have been studied, often based on the presumption of decentralized control and communication. However, works in which the combinatorial aspects and the lower-level continuous-time trajectories are optimized in a unified fashion, in which the trade-offs that are made can be clearly identified, are scarce in the current literature.

1.2 Literature review

This section presents a general overview of the relevant literature on *optimization-based methods for traffic coordination in fully automated mobility systems*. Background and literature on the specific *methodologies* that we will use in this thesis are provided at the beginning of each chapter or section where we first need it.

In presenting the literature, we group works according to their most salient feature—be it a specific modeling choice, an algorithmic technique, or a focus on particular issue. Other common features that are worth mentioning are reservation-based policies, use of mixed-integer linear programming, fixed ordering policies instead of optimization, fixed signal phases, platoon-based heuristics.

Before we discuss exemplary works, we mention some existing surveys that we have consulted when conducting this literature review. The survey [39] classifies works mainly on the level of organization. Moreover, apart from vehicle motion around intersections and other conflict spaces, they also review are other interesting coordination challenges in autonomous traffic management, like smart parking and ride sharing, which also have significant societal impact. More specifically focused on autonomous intersection management are [30], where works are classified, first on level of organization, but they also consider vehicle dynamics, conflict detection and scheduling policy.

Reservation-based systems. Instead of an optimization-based perspective, many studies assume that vehicles communicate with a central intersection controller to reserve time and space in the conflict area. The major downside of this line of work is that there is generally no precise control over the trajectories that vehicles take.

The seminal "Autonomous Intersection Management" (AIM) model [13, 14] is a good example of the reservation-based approach. In this framework, vehicles that want to cross the intersection send a request to the central controller to occupy the cells containing their trajectory for a certain amount of time. The central controller then decides to grant or deny these requests based on previously granted requests to facilitate collision-free trajectories. If a request is denied, the vehicle slows down and attempts to obtain a new reservation after some timeout. Note that in the original setting, the central controller does not have complete control over the precise trajectories that vehicles take; each vehicle agent is more or less able to decide their own optimal speed profile. However, the central controller does impose some constraints on the acceleration profile inside the intersection to promote throughput.

Various improvements have been made to the original AIM model to improve the efficiency of the reservation protocol, for example, by having vehicles make better estimations of their expected time of arrivals to make more accurate reservations [4]. Other works consider speed profile recommendations and more fine-grained prioritization of requests by the intersection controller [23]. The model has also been extended to networks of intersections, where dynamic route optimization has been considered as well [22]. Later works propose more precise methods for conflict detection than the original cell-based approach [33, 34]. Finally, communication delays play a major role in practice, so a time-aware extension of AIM has been introduced in [29].

Optimizing the crossing order. Rather than obtaining a crossing order as the result of a sequence of granted reservation requests, later studies assume that the intersection manager explicitly optimizes the sequence in which vehicles cross. Most of these works also address the online setting, accounting for the random arrival of vehicles over time. In this online setting, the general goal is to find a *schedule* of crossing times for the current vehicles in the system and update this schedule every time new arrivals happen, which is the task of the central *crossing time scheduling policy*.

A common method to design such a scheduling policy is to use *rolling-horizon optimization*. For example, taking the AIM model as a starting point, a paradigm shift from reservation requests to *assignments* has been proposed in [33], where a a mixed-integer linear program is solved in a rolling-horizon fashion to determine the best assignments. A similar approach is described in [34], but they formulate the reservation assignment problem as a non-linear optimization problem, which is solved using a tabu search heuristic.

The policy in [49] deals explicitly with the complexity of the crossing order decisions by defining groups of consecutive vehicles on the same lane. The first step is to group vehicles into these so-called "bubbles". All vehicles in a bubble are required to cross the intersection together, while maintaining feasibility with respect to safe trajectories. Next, crossing times are assigned to bubbles while avoiding collisions. Based on this schedule, a local vehicular control method [48] is used that guarantees safety to reach the assigned crossing times.

The work [41] considers the scheduling policy in the context of a polling model, where the intersection, its inbound lanes and vehicles are modeled as server, queues and customers, respectively. Roughly speaking, each time a new vehicle arrives to the system, the crossing order may be adapted, which is done by simulating a polling policy. They show that if the polling policy satisfies some regularity condition and if the lanes are long enough, then every updated crossing order is feasible, in the sense that vehicles in the system that have been assigned a later crossing time than before are able to decelerate in time to avoid collisions. The generation of continuous-time vehicle trajectories satisfying these crossing times is handled by numerically solving an optimal control problem. Building on this work, it has been shown that these trajectories can also be computed directly using closed-form expressions [53].

Finally, we note that crossing order decisions become particularly interesting when vehicles with heterogeneous dimensions and dynamics are considered, which has been investigated in [27]. For example, it makes intuitively sense to assign heavy trucks with slow acceleration/deceleration characteristics to a dedicated lane to avoid interfering with passenger vehicles that are more agile.

Joint optimization and decomposition methods. Finally, we mention some of the few works that have considered the joint optimization perspective, which has also been called "signal-vehicle coupled control (SVCC)". The prominent theme here is trying to come up with good approximation algorithms. A central idea in this line of work is trying to exploit somehow the fact that the problem can be formulated in terms of a decomposition of the upper-level crossing time scheduling problem and lower-level problems for generating continuous-time trajectories.

The approximation method in [26] is based on a bilevel decomposition and considers a quadratic objective involving velocity as a proxy for energy. The first stage optimizes a schedule of vehicle crossing times. It uses approximations of each vehicle's contribution to the total objective as a function of its crossing time. Next, for each vehicle, the second stage computes an optimal trajectory satisfying the crossing time schedule, by solving a quadratic program. This approach has been shown to reduce running times significantly. Unfortunately, the study is limited to a single intersection and it is assumed that each lane approaching the intersection contains exactly one vehicle.

The paper [57] proposes a trajectory optimization scheme for a single intersection, also based on the bilevel decomposition. The lower-level problem is employed to maximize the speed at which vehicles enter the intersection. Both parts of the problem are solved in an alternating fashion, each time updating the constraints of the other part based on the current solution.

Identified gaps. Add a small summary to synthesize the key gaps in the literature.

1.3 Project goals and outline

From the start of this project, the overarching goal has been to develop tractable optimization models and algorithms for the coordination of automated vehicles at intersections. In the initial phase, we considered dynamic and stochastic aspects like random vehicle arrivals. However, after reviewing the current state of the literature, we realized that there are still many unresolved issues in the deterministic settings. In the end, this project has been centered around the following two general goals related to coordination of autonomous vehicles:

- *Develop a simplified yet representative mathematical optimization model for coordination of autonomous vehicles in networks of intersections.*

- *Understand the inherent algorithmic challenges that complicate joint optimization of crossing order and generation of continuous-time trajectories.*

Complementary to these problem-driven goals, the work presented in this thesis has also been inspired by some developments in applying machine learning methods to combinatorial optimization. [this needs a little more introduction] Motivated by these recent successes, we also aim to *illustrate the use of such machine learning algorithms to obtain heuristics for combinatorial problems arising in the context of coordinating autonomous vehicles.*

To make the above broad goals concrete and addressable, our work has been guided by the following research questions:

RQ1: How can the autonomous traffic coordination problem with collision-avoidance constraints at a single isolated intersection be formulated in an optimization framework for multiple autonomous vehicles? (modeling)

RQ2: Under which assumptions can the combinatorial aspect of determining the crossing order be considered in isolation? (decomposition)

RQ3: What is the relative computational complexity of solving the combinatorial problem compared to solving the continuous-time trajectory optimization, given the crossing order? (complexity)

RQ4: How to leverage the recent successes in applying machine learning for combinatorial optimization to solve the combinatorial problems arising in the context of autonomous traffic management? (heuristics)

RQ5: How to extend the previous questions to a network of multiple connected intersections? (scalability)

The contributions of this work can be summarized into three main categories: More appropriate in the conclusion chapter?

(i). Isolating the combinatorial problem. Deciding the crossing order of vehicles at intersections is a central challenge in traffic management. This holds especially for when considering multiple lanes and intersections. However, after fixing these ordering decisions, the remaining problem is often much easier to solve. This observation motivates the decomposition of the trajectory optimization problem into two parts. The upper-level problem determines the times at which vehicles cross the intersections on their routes, to which we will refer as *crossing times*. Once these are fixed, we solve a set of lower-level problems to find the corresponding vehicle trajectories satisfying the crossing times. Our first contribution is to show that, under certain conditions, our joint trajectory optimization problem for vehicles in a network of intersections decomposes into an upper-level crossing time scheduling problem and a set of lower-level trajectory optimization problems. We show that feasibility of the upper-level scheduling problem is completely characterized in terms of a system of linear inequalities involving the crossing times. This allows us to first solve the scheduling problem and then generate trajectories for it once we have the optimal crossing time schedule.

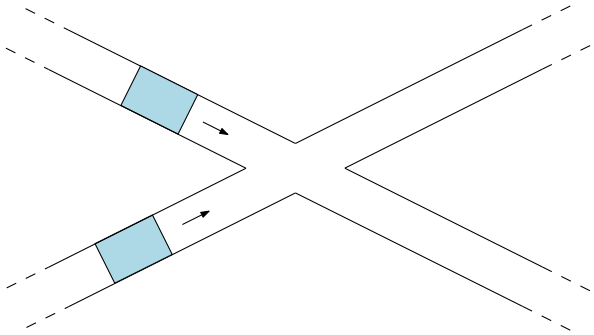
(ii). Learning-based scheduling. Our second contribution is an illustration of how machine learning techniques can be applied to solve scheduling problems in practice, using our crossing time scheduling problem as a case study. Many practical instances of combinatorial problems contain structure that classic solutions techniques try to exploit, e.g., by defining smart heuristics based on human intuition and experience. A recent trend in the literature has been to aim for automation of this manual endeavor, for example by formulating parametric sequence models to capture the conditional probability of optimal solutions, given a problem instance. Specifically, we recognize that the solution of the crossing time scheduling problem can be interpreted as a sequence of decisions. Instead of manually trying to develop good heuristics and algorithms, we try to learn what optimal solutions are, by treating it as a learning task on sequences. As has been noted before, we confirm that the order of evaluation during inference matters a lot for the final solution quality.

(iii). Coordination across multiple intersections. We show a way to extend isolated intersection coordination to multi-intersection networks. We show under which assumptions this problem can still be formulated as a scheduling problem, which turns out to be an extension of the classical job shop scheduling problem. In this setting, feasibility of trajectories deserves special attention.

Outline. In Chapter 2, we consider a simple model of a single intersection, like the example above. After discussing the decomposition method, we present some classical solutions methods to solve the crossing time scheduling problem. We explain how the problem can be treated as a learning problem in Chapter 3. In order to generalize to a network of intersections, we need to give a precise characterization of the feasibility of trajectories in lanes of finite length, which is done in Chapter 5. The resulting scheduling problem is then subjected to a learning algorithm in Chapter 4. We provide some general discussion and pointers for further research in Chapter 6.

Part I

Isolated Intersection



Chapter 2

Isolated intersection scheduling

Efficient coordination of vehicle motion at intersections has been and still is a central challenge in traffic management, because intersections are natural bottlenecks where safety requirements and efficiency objectives are directly in conflict. With the advent of automated vehicles and reliable wireless communication technologies, there is increasing potential to replace traditional traffic signal-based approaches with coordinated trajectory planning methods. Our literature review clearly shows that previous works have approached coordination of automated vehicles from a wide range of perspectives and with varying degrees of model complexity. Furthermore, we highlighted some fundamental computational challenges that are still unresolved in the literature. Specifically, the joint optimization of crossing order and continuous-time vehicle trajectories has received relatively little attention. In order to better understand this issue and to enable precise analytical results, we propose to study a stylized single intersection model.

Formulated mathematically, coordination of automated vehicles at a single intersection can be expressed as a problem of *trajectory optimization*, which we will make precise in Section 2.1. In this formulation, the accelerations of all vehicles are chosen to satisfy their dynamics and collision-avoidance requirements while minimizing some cost function. The following two conflicting cost components are of primary interest: minimizing travel delay, which is essential for capturing the efficiency of traffic flow; and minimizing energy consumption, which is highly desirable in practice because it supports sustainability.

Trajectory optimization problems of this type can be solved directly using general-purpose optimization schemes based on time discretization. As we will illustrate in Section 2.2.1, such methods are straightforward to implement and guarantee to find an optimal solution to the joint optimization problem. However, their running time scales very poorly when considering finer discretization grids or when increasing the number of vehicles in the system, motivating the development of tailored algorithms that exploit the structure of the problem.

Given the observation that each feasible set of trajectories implicitly fixes some order in which vehicles cross the intersection, it is natural to ask whether this ordering decision can be considered separately. In general, the answer is negative; in other words, we cannot first optimize the crossing order, then compute the corresponding continuous-time vehicle trajectories. Still, this decomposition idea is a helpful way to think about the problem, so it is discussed extensively in Section 2.2.2.

It turns out that a proper decomposition is possible, if we use a cost function that only involves delay and if we are willing to impose the additional constraint that vehicles need to cross the intersection at full speed. For this case, we show in Section 2.2.3 that the original infinite-dimensional trajectory optimization problem essentially reduces to a finite-dimensional *scheduling problem*, in which only the sequencing of vehicles and their entry times into the intersection need to be determined. Scheduling problems like this have been studied extensively and can be solved using off-the-shelf methods like mixed-integer linear programming, which we illustrate in Section 2.3. By leveraging insight into the structure of optimal solutions, we are able to formulate three problem-specific cutting planes, yielding an efficient algorithm for coordination at a single intersection.

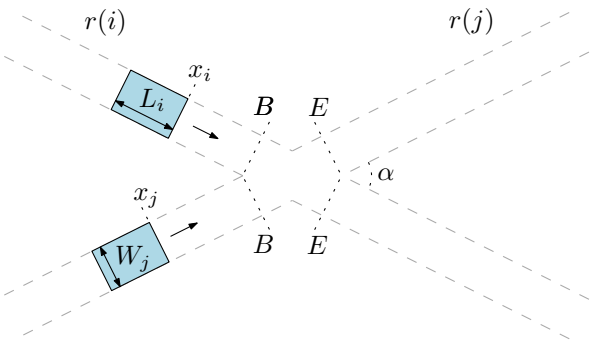


Figure 2.1: Example of two vehicle routes intersecting at some angle α. The indicated positions $x_i = B$ and $x_i = E$ for each vehicle i are chosen such that whenever the vehicle position x_i satisfies either $x_i \leq B$ or $x_i - L_i \geq E$, then the intersection area is not occupied by this vehicle at all and the other vehicle can take any position without causing a collision.

Remark 2.1. *The main starting point of this chapter will be a continuous-time trajectory optimization problem, which is typically studied in the framework of optimal control theory. Since the author does not have a strong background in this field, the aim is not to provide a rigorous treatment from this perspective. Nevertheless, we want to be as precise and complete as possible, because this trajectory optimization problem is the basis for our further development. Therefore, at some points in this chapter, remarks are included regarding the subleties of our formulation, most notably, regarding the occurrence of so-called state constraints.*

2.1 Intersection model

To make the analysis tractable, we restrict our attention to intersections of single-lane roads on which vehicles are traveling without turning maneuvers or overtaking. All vehicles are assumed to be homogeneous, sharing identical dimensions and dynamics, so that their motion can be modeled uniformly. A central controller determines the acceleration, and thus the speed, of each vehicle, under the assumption of perfect communication. Moreover, we do not consider randomness in arrivals or dynamics, so we assume that each vehicle's initial state is known precisely such that the system evolves deterministically as a function of the acceleration control inputs.

We start by defining the geometry of the intersection and the vehicles and analyze the resulting conflict-free joint positions of all the vehicles in the system. After that, we introduce the dynamics of the system and define several objective functions to arrive at the trajectory optimization problem.

Valid configurations. We model each vehicle i in the plane as some rigid body $\mathcal{B}_i$ traveling along some straight line $r(i)$, to which we will refer as the vehicle's *route*. We will assume that vehicles always stay on their route and thus do not make turning maneuvers. Therefore, the longitudinal position of a vehicle along its route can be represented by some scalar $x_i \in \mathbb{R}$. For simplicity, we use a rectangular vehicle geometry, so each $\mathcal{B}_i$ is a translated rectangle of width W_i and length L_i. For technical convenience, we assume this rectangle is an *open* set. We write $\mathcal{B}_i(x_i)$ to denote the corresponding translated rigid body in the plane, where x_i corresponds to the location of the front bumper; the rear bumper position is then $x_i - L_i$. We allow multiple routes to cross in a single point. Of course, this causes some joint vehicle positions to be invalid, because they would correspond to collisions. Before we allow arbitrary numbers of vehicles to have the same route, we briefly investigate the valid configurations of two vehicles, each on its own route.

Consider two routes intersecting at some angle α, as illustrated in Figure 2.1, each with a single vehicle on it. Let these vehicles be denoted as i and j. We can try to characterize the set $\mathcal{X}_{ij} \subset \mathbb{R}^2$ of feasible configurations (x_i, x_j) for which these two vehicles are not in a

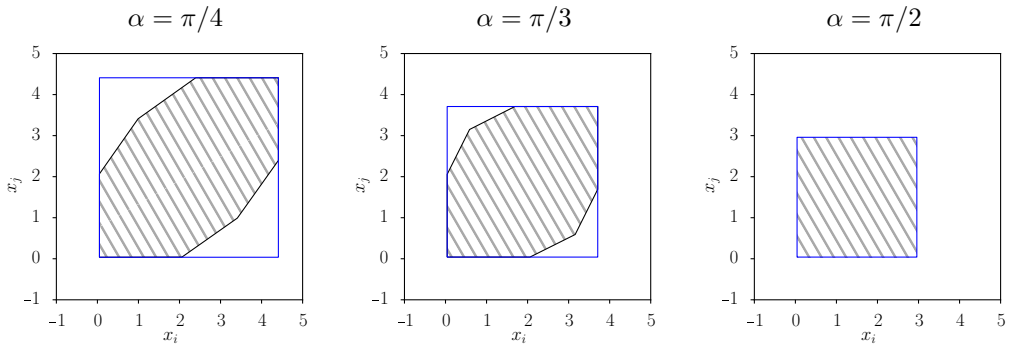


Figure 2.2: For three different intersection angles and fixed vehicle dimensions $W_i = W_j = 1$ and $L_i = L_j = 2$, we plotted the region $\mathcal{X}_{ij}^C$ in configuration space corresponding to collisions as the area marked in grey. The blue square regions correspond to the approximation of the collision area using (2.2). Because we assume a rectangular vehicle geometry, these figures are relatively straightforward to compute, which we briefly explain in Appendix A.

collision, in the sense that their corresponding translated rigid bodies do not intersect. In general, this set can thus simply be defined as

$$\mathcal{X}_{ij} := \{(x_i, x_j) \in \mathbb{R}^2 : \mathcal{B}_i(x_i) \cap \mathcal{B}_j(x_j) = \varnothing\}. \tag{2.1}$$

However, it is often easier to take the opposite perspective, by characterizing the set of conflicting configurations $\mathcal{X}_{ij}^C$.

In the situation with two routes, we fix two reference positions B and E on each route, delimiting the intersection area as shown in Figure 2.1. These positions are chosen such that whenever $x_i \leq B$ or $x_i - L_i \geq E$, it is clear that vehicle i does not occupy the intersection at all, so the other vehicle j is free to take any position $x_j \in \mathbb{R}$. Thus, we obtain the following conservative approximation of the set of conflicting configurations:

$$(B, E + L_i) \times (B, E + L_j) \supseteq \mathcal{X}_{ij}^C. \tag{2.2}$$

Of course, the set of feasible configurations is generally a little larger and depends on the angle α of intersection, as illustrated by the three examples in Figure 2.2. In case of the third example, where the intersections make a right angle $\alpha = \pi/2$, it it not difficult to see that there is actually equality in (2.2).

To keep the presentation simple, we will make the following assumption: all vehicles share the same vehicle geometry, i.e, $L_i \equiv L$ and $W_i \equiv W$. As a shorthand of the vehicle positions that correspond to occupying the intersection area, we will write $\mathcal{E} := (B, E + L)$. This enables us to model any number of intersecting routes, as long as we can assume that $\mathcal{E}$ provides a conservative approximation of all intersection-occupying vehicle positions.

Let us now proceed to arbitrary numbers of vehicles. We will use the following notation for identifying routes and vehicles. Let the routes be identified by indices $\mathcal{R} := \{1, \ldots, R\}$. Let n_r denote the number of vehicles following route r, and let $N := n_1 + \cdots + n_R$ denote the total number of vehicles in the system, for which we have the set of vehicle indices

$$\mathcal{N} = \{(r, k) : k \in \{1, \ldots, n_r\}, r \in \mathcal{R}\}. \tag{2.3}$$

Occasionally, we will also write $\mathcal{N}_r$ to identify the set of vehicles on route r. Given vehicle index $i = (r, k) \in \mathcal{N}$, we use the auxiliary notation $r(i) = r$ and $k(i) = k$.

In order to maintain a safe distance between successive vehicle on the same route, it is clear that vehicles need to satisfy the *safe headway constraints*

$$x_i - x_j \geq L, \tag{2.4}$$

at all times, for all pairs of indices $i, j \in \mathcal{N}$ such that $r(i) = r(j)$ and $k(i) + 1 = k(j)$. Let $\mathcal{C}$ denote the set of all such ordered pairs of indices. Note that these constraints restrict

vehicles from overtaking each other, so their initial relative order is always maintained. In other words, $(i,j) \in \mathcal{C}$ means that i crosses the intersection before j.

Similarly, let $\mathcal{D}$ denote the set of *conflicts*, which are all (unordered) pairs of vehicles $i,j \in \mathcal{N}$ on different routes $r(i) \neq r(j)$. Recall that we introduced $\mathcal{E}$ to denote the interval of positions for which some vehicle is said to *occupy* the intersection. Then, for each conflict $\{i,j\} \in \mathcal{D}$, we impose at all times the *safe crossing constraints*

$$(x_i, x_j) \notin \mathcal{E}^2. \tag{2.5}$$

Trajectory optimization. Next, we introduce the motion dynamics of the vehicles, so let $x_i(t)$ denote the position of vehicle i at time t. Let $\dot{x}_i$ and $\ddot{x}_i$ denote its speed and acceleration, respectively, then we consider the bounds

$$0 \leq \dot{x}_i(t) \leq \bar{v}, \tag{2.6a}$$
$$-\omega \leq \ddot{x}_i(t) \leq \bar{\omega}, \tag{2.6b}$$

for some positive $\bar{v}, \omega, \bar{\omega} > 0$. Given a pair of initial position and velocity $s_i = (x_i^0, v_i^0)$, we write $x_i \in D(s_i)$ if and only if the trajectory x_i has $(x_i(0), \dot{x}_i(0)) = s_i$ and satisfies (2.6).

We now present some possible ways of measuring how desirable an individual vehicle trajectory $x_i(t)$ is, by defining a cost functional $J(x_i)$ that we aim to minimize. For example, consider the following parametric cost functional

$$J_{\alpha,\beta}(x_i) = \int_0^{t_f} \alpha x_i(t) + \beta |\ddot{x}_i(t)| \, \mathrm{d}t. \tag{2.7}$$

First of all, note that the absolute value of the acceleration is often used as a proxy for energy consumption, so $\beta > 0$ is generally desirable. Minimizing energy is in direct conflict with our other main goal, which is to reach the intersection in some reasonable amount of time. However, we have not yet explicitly encoded this. We will explicitly add this goal later, but for now, it is possible to achieve a similar effect by setting $\alpha < 0$, which may be interpreted as rewarding trajectories that "move forward fast" and is thus a natural choice if we want to promote overall throughput of the system. Minimizing $J_{\alpha,\beta}$ with $\alpha = -1$ and $\beta = 0$ will be of particular interest to us later, which we will call the *haste objective*, for ease of reference. To give another example, consider cost functionals of the form

$$J_{v_d}(x_i) = \int_0^{t_f} (v_d - \dot{x}_i(t))^2 + (\ddot{x}_i(t))^2 \, \mathrm{d}t, \tag{2.8}$$

where $v_d > 0$ is some reference velocity. This objective can be interpreted as trying to maintain a velocity close to v_d, while simultaneously minimizing the square of acceleration as proxy of energy consumption.

Given some $J(\cdot)$, we can now conclude our model description by defining the general trajectory optimization problem. Given the pairs of initial vehicle positions and velocities $s := \{s_i = (x_i^0, v_i^0) : i \in \mathcal{N}\}$ and some final simulation time $t_f > 0$, we consider the problem of finding some set $x = \{x_i : i \in \mathcal{N}\}$ of continuous trajectories x_i, that minimize

$$T(s) := \min_x \sum_{i \in \mathcal{N}} J(x_i) \tag{T}$$
$$\text{such that} \quad x_i \in D(s_i) \qquad \text{for all } i \in \mathcal{N}, \tag{T.1}$$
$$x_i(t) - x_j(t) \geq L \qquad \text{for all } (i,j) \in \mathcal{C}, \tag{T.2}$$
$$(x_i(t), x_j(t)) \notin \mathcal{E}^2 \qquad \text{for all } \{i,j\} \in \mathcal{D}. \tag{T.3}$$

Our main interest lies in solving this problem for different instances of s, the remaining "system parameters" $(L, \mathcal{E}, \bar{v}, \omega, \bar{\omega}, t_f)$ are mostly assumed to be fixed.

Remark 2.2. *Problem* (T) *is non-convex in some sense; recall that the set of feasible configurations $\mathcal{X}_{ij}$ for two vehicles is already non-convex. An interpretation of this non-convexity is that we must decide which of the vehicles crosses the intersection first. In the next section, we will discuss ways of dealing with this non-convexity.*

Remark 2.3. *Note that the variables in* (T) *represent continuous-time trajectories, so it is an infinite-dimensional optimization problem. This kind of problems is typically studied in the framework of optimal control theory. In this context, the joint position* $x = \{x_i : i \in \mathcal{N}\}$ *is referred to as the* state *of the system and* $\ddot{x} = \{\ddot{x}_i : i \in \mathcal{N}\}$ *as the* control*. In some sense, acceleration is thus understood to be the main input of the system, such that vehicle position is indirectly determined by integrating twice. For this reason, this simple vehicle dynamics model is commonly referred to as the* double integrator*. It is more common to state optimal control problems in terms of finding some continuous* $x(t) \in \mathbb{R}^n$ *and integrable* $u(t) \in \mathbb{R}^m$*, which need to satisfy the dynamic equations*

$$\dot{x} = f(t, x, u), \quad x(t_0) = x_0, \quad u \in U \subset \mathbb{R}^m, \tag{2.9}$$

for some initial state x_0 *and time* t_0 *and some (compact) control set* U*. Given some final time* t_f*, a running cost function* $\mathcal{L}$ *and some terminal cost function* $\mathcal{K}$*, the goal is to*

$$\min_x \int_{t_0}^{t_f} \mathcal{L}(t, x(t), u(t))\,\mathrm{d}t + \mathcal{K}(t_f, x_f), \tag{2.10a}$$

$$\text{such that } (2.9) \text{ holds}, \tag{2.10b}$$

$$h(x) = 0, \tag{2.10c}$$

$$g(x) \leq 0, \tag{2.10d}$$

with some functions $h : \mathbb{R}^n \to \mathbb{R}^p$ *and* $g : \mathbb{R}^n \to \mathbb{R}^q$ *encoding the so-called state constraints. Functions* f, g, h *are usually assumed to possess certain regularity or smoothness properties. There are general methods to analyze optimal solutions of* (2.10)*, most notably the necessary conditions given by the maximum principle of Pontryagin [35]. However, the occurrence of state constraints like* (2.10c) *and* (2.10d) *make such analysis notoriously difficult and remains an area of ongoing research [21]. For our current problem, note that the collision-avoidance constraints* (T.2) *and* (T.3)*, as well as the speed constraint* (2.6a) *implied in* (T.1)*, are all of the type* (2.10d)*.*

Feasibility. The existence feasible trajectories in (T) generally depends on the initial state s of vehicles in some non-trivial manner.[1] To give a rough idea, we derive some simple sufficient conditions. We exclude initial collisions at time $t = 0$ by requiring

$$x_i^0 - x_j^0 > L \quad \text{for all} \quad (i, j) \in \mathcal{C}. \tag{2.11}$$

Next, observe that the bounds on speed and acceleration imply that it takes at least $\bar{v}/\omega$ time to fully decelerate from full speed, during which the vehicle has traveled a distance of $\bar{v}^2/(2\omega)$. Similarly, a full acceleration from a stop takes $\bar{v}/\bar{\omega}$ time and $\bar{v}^2/(2\bar{\omega})$ distance. Therefore, by assuming that each vehicle starts at full speed $v_i^0 = \bar{v}$ and by imposing a minimum distance to the start of the intersection

$$x_{(r,1)}^0 < B - \bar{v}^2/(2\omega) - \bar{v}^2/(2\bar{\omega}), \tag{2.12}$$

for each first vehicle on every route $r \in \mathcal{R}$, we ensure that there is enough room for all vehicles to come to a full stop. Even further, there is still enough distance for a full acceleration to reach full speed while crossing the intersection.

Assumption 2.1. *The initial states* $s = \{s_i = (x_i^0, v_i^0) : i \in \mathcal{N}\}$ *satisfy* (2.11) *and* (2.12)*.*

[1] Note that we will characterize the feasibility of trajectories for a related problem in Chapter 5. The main difference is that the analysis there will not assume that initial vehicle states are given, but rather the times at which vehicles enter the system.

2.2 From joint optimization to bilevel decomposition

The main goal of this section is to explain how the trajectory optimization problem (T) can be reduced to a scheduling problem. To motivate this reduction, we first show that the problem can be solved by employing a numerical method known as *direct transcription*, but we note that the associated running time is prohibitive in practice. We then show how (T) can in general be rewritten in terms of coupled optimization problems: an upper-level scheduling problem, dictating the crossing times for all vehicles in the system, and a lower-level trajectory optimization problem for the vehicles of each route. We explain how this reformulation has been used as the basis for an approximation algorithm. Finally, we show which assumptions we have to make in order for the decomposition to be *proper*, by which we mean that we can first solve the upper-level problem to find an optimal crossing time schedule, after which the lower-level problems can be resolved.

2.2.1 Joint optimization using direct transcription

The key idea of direct transcription is to reformulate the continuous-time optimal control problem—which is infinite-dimensional—into a finite-dimensional nonlinear optimization problem by discretizing time into a grid. At each grid point, the state and control inputs of every vehicle become decision variables. The vehicle dynamics and the safety requirements, i.e., the headway and collision-avoidance constraints, can then be imposed directly in terms of these decisions variables. This approach allows us to capture the full structure of the problem while making it accessible to modern nonlinear optimization solvers. In what follows, we describe in detail how this general approach can be applied to problem (T) and illustrate its use by solving some examples for two different cost functionals.

We start by defining a uniform discretization grid. Let K denote the number of discrete time steps and let Δt denote the time step size, then we define

$$\mathbb{T} := \{0, \Delta t, \dots, K\Delta t\}. \tag{2.13}$$

For each vehicle i, we use $x_i(t), v_i(t)$ and $u_i(t)$ to denote, respectively, the decision variables for position, speed and acceleration. First of all, we impose the initial conditions by simply adding the constraints $x_i(0) = x_i^0$ and $v_i(0) = v_i^0$ for each $i \in \mathcal{N}$. Using the forward Euler integration scheme, we further relate these three quantities by adding the constraints

$$x_i(t + \Delta t) = x_i(t) + v_i(t)\Delta t, \tag{2.14a}$$
$$v_i(t + \Delta t) = v_i(t) + u_i(t)\Delta t, \tag{2.14b}$$

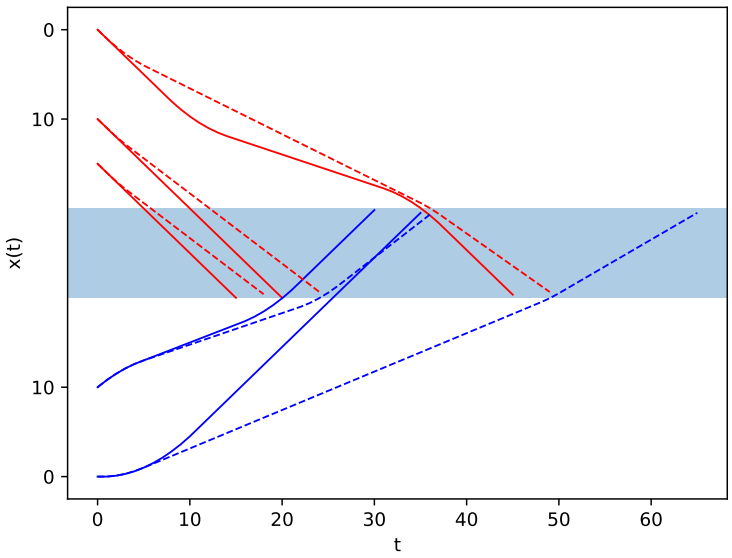
for each $t \in \mathbb{T} \setminus \{K\Delta t\}$ and $i \in \mathcal{N}$. Moreover, we directly include the inequalities $0 \leq v_i(t) \leq \bar{v}$ and $-\omega \leq u_i(t) \leq \bar{\omega}$ to model the vehicle dynamics. For each pair of successive vehicles $(i, j) \in \mathcal{C}$ on the same route, the safe headway constraints can simply be added as

$$x_i(t) - x_j(t) \geq L \quad \text{for each } t \in \mathbb{T}. \tag{2.15}$$

Encoding of the safe crossing constraints needs some additional attention, because they represent a binary "disjunctive" decision. Following the approach in [26], these disjunctive constraints can be formulated using the common big-M method with binary decision variables. For each vehicle $i \in \mathcal{N}$, we introduce two binary decision variables $\delta_i(t), \gamma_i(t) \in \{0, 1\}$ and for each conflict $\{i, j\} \in \mathcal{D}$ and time step $t \in \mathbb{T}$, we introduce the following constraints:

$$x_i(t) \leq B + \delta_i(t)M, \tag{2.16a}$$
$$x_i(t) \geq E + L - \gamma_i(t)M, \tag{2.16b}$$
$$\delta_i(t) + \delta_j(t) + \gamma_i(t) + \gamma_j(t) \leq 3, \tag{2.16c}$$

where M is some sufficiently large number. The idea behind this encoding is as follows. First, observe that setting $\delta_i(t)$ can be thought of as *deactivating* (2.16a), since M is chosen sufficiently large such that the inequality is trivially true. Analogously, setting $\gamma_i(t) = 1$ deactivates (2.16b). Hence, the constraint (2.16c) can be interpreted as limiting the number



i	x_i^0	v_i^0
(1,3)	0	1.0
(1,2)	10	1.0
(1,1)	15	1.0
(2,1)	10	0.8
(2,2)	0	0.0

L	=	5
$\mathcal{E}$	=	$[20, 30]$
$\bar{v}$	=	1
$\omega, \bar{\omega}$	=	0.1
T	=	50
Δt	=	1

Figure 2.3: Example of optimal trajectories minimizing $J_{\alpha,\beta}$ (solid) and J_{v_d} (dashed), obtained using the direct transcription method for cost parameters $\alpha = -1$, $\beta = 100$ and $v_d = 0.6$. The system parameters and initial conditions are listed next to the figure. The y-axis is split such that the top part corresponds to route 1 and the bottom to route 2 and the trajectories are inverted accordingly and drawn with separate colors. The interval of intersection-occupying positions $\mathcal{E}$ is drawn as a shaded region. We do not draw trajectories beyond this region.

of deactivations to three, which is equivalent to requiring at least one of the following four inequalities to hold:

$$x_i(t) \leq B, \quad x_j(t) \leq B, \quad x_i(t) \geq E + L, \quad x_j(t) \geq E + L, \tag{2.17}$$

which means that vehicles i and j cannot both occupy the intersection at the same time t.

In general, the resulting transcribed optimization problem is a mixed-integer nonlinear program. We consider an small problem instance with five vehicles for our cost functional $J_{\alpha,\beta}$, for which the optimal trajectories are shown in Figure 2.3.

Remark 2.4. *We used the forward Euler integration scheme for sake of a simple presentation. However, note that in practice we most likely want to use a more numerically stable method like a higher-order Runge-Kutta scheme or by means of some spline interpolation technique. We refer to [1] for a light introduction to trajectory optimization in general and [28] for a tutorial on the direct collocation method, which also contains a concise overview of different numerical methods (ibid., Section 9).*

2.2.2 Bilevel decomposition

After applying the transcription method above, the safe crossing constraints of each conflict are imposed at every time step, requiring a total of $2T$ binary variables. Therefore, there is some kind of redundancy in this encoding, because the decision to be made is, in principle, a single binary decision for each conflict: which of the two vehicle crosses the intersection first? However, a more direct encoding would require decision variables for the time of entry into and exit out of the intersection for each vehicle. This insight is the key to decomposing the problem into an upper-level problem and a set of lower-level problems. Roughly speaking, the upper-level problem optimizes the time slots during which vehicles occupy the intersection, while the lower-level problems produce optimal safe trajectories that respect these time slots.

Given some feasible trajectory $x_i \in D(s_i)$ for a single vehicle i, we define its (earliest) *entry* time and (earliest) *exit* time, respectively, to be

$$\tau(x_i) := \min\{t \in [0, t_f] : x_i(t) = B\}, \tag{2.18a}$$
$$\xi(x_i) := \min\{t \in [0, t_f] : x_i(t) = E + L\}. \tag{2.18b}$$

Note that the sets in the definition are closed, because x_i is continuous by assumption, but they may be empty. Therefore, we use the convention that "$\min \varnothing = \infty$", such that $\tau(x_i) = \infty$ whenever x_i does not reach the intersection at all and, analogously, we have $\xi(x_i) = \infty$ whenever the end of the intersection is never reached. Furthermore, using the convention that "$(\infty, \infty) = \varnothing$", observe that $\tau(x_i)$ and $\xi(x_i)$ determine the times $t \in (\tau(x_i), \xi(x_i))$ during which vehicle i occupies the intersection. Recall the encoding of the collision constraints using binary variables in (2.16). Similarly, observe that the collision-avoidance constraints

$$(x_i(t), x_j(t)) \notin \mathcal{E}^2 \quad \text{for all } \{i,j\} \in \mathcal{D} \tag{2.19}$$

are completely equivalent to the constraints

$$\big(\tau(x_i), \xi(x_i)\big) \cap \big(\tau(x_j), \xi(x_j)\big) = \varnothing \quad \text{for all } \{i,j\} \in \mathcal{D}. \tag{2.20}$$

The main idea of the decomposition is to make these entry and exit times concrete decision variables of the upper-level problem. Hence, for each vehicle i, we introduce a decision variable y_i for the time of entry and a variable z_i for the time of exit. When the occupancy time slots $\{(y_i, z_i) : i \in \mathcal{N}\}$ are fixed and satisfy (2.20), the trajectory optimization problem essentially reduces to solving a separate lower-level problem for each route.

In order to make this more precise, let us introduce some shorthand notation for collections of parameters and variables pertaining to a single route. Recall that $\mathcal{N}_r$ denotes all vehicles on route r. We write $s_r := \{(x_i^0, v_i^0) : i \in \mathcal{N}_r\}$ to denote the corresponding initial conditions and we write $x_r := \{x_i : i \in \mathcal{N}_r\}$ as a shorthand for a set of trajectories on route r. Consider some route $r \in \mathcal{R}$ with local initial conditions s_r and suppose we are given some fixed local occupancy time slots as determined by $y_r := \{y_i : i \in \mathcal{N}_r\}$ and $z_r := \{z_i : i \in \mathcal{N}_r\}$, then we define the lower-level *control problem*

$$F(y_r, z_r, s_r) := \min_{x_r} \sum_{i \in \mathcal{N}_r} J(x_i) \tag{L}$$

$$\begin{aligned}
\text{s.t.}\quad & x_i \in D(s_i) && \text{for all } i \in \mathcal{N}_r, && \text{(L.1)}\\
& x_i(t) - x_j(t) \geq L && \text{for all } (i,j) \in \mathcal{C} \cap \mathcal{N}_r, && \text{(L.2)}\\
& x_i(y_i) = B && \text{for all } i \in \mathcal{N}_r, && \text{(L.3)}\\
& x_i(z_i) = E + L && \text{for all } i \in \mathcal{N}_r. && \text{(L.4)}
\end{aligned}$$

Note that the feasibility of this problem depends on the initial states as well as the choice of occupancy time slots. Therefore, given initial states s_r, we write $(y_r, z_r) \in \mathcal{T}(s_r)$ to denote the set of occupancy time slots that allow a feasible solution.

The upper-level problem is now to find a set of occupancy timeslots satisfying (2.20), such that the lower-level problem for each route is feasible. Let $s = \{s_r : r \in \mathcal{R}\}$ denote the set of global initial states for all routes and write $y = \{y_r : r \in \mathcal{R}\}$ and $z = \{y_z : r \in \mathcal{R}\}$ to denote a set of global occupancy time slots, then we define the upper-level *scheduling problem*

$$U(s) := \min_{y,z} \sum_{r \in \mathcal{R}} F(y_r, z_r, s_r) \tag{U}$$

$$\begin{aligned}
\text{s.t.}\quad & (y_i, z_i) \cap (y_j, z_j) = \varnothing && \text{for all } \{i,j\} \in \mathcal{D}, && \text{(U.1)}\\
& (y_r, z_r) \in \mathcal{T}(s_r) && \text{for all } r \in \mathcal{R}. && \text{(U.2)}
\end{aligned}$$

Explicit crossing. Without further assumptions, problem (U) is not necessarily equivalent to the original problem (T). As we already noted when defining cost functionals, the issue lies in the fact that some feasible solution x of (T) does not have to cross the intersection at all, i.e., it is not guaranteed that $\tau(x)$ and $\xi(x)$ are finite for x. To illustrate this situation, consider the following (pathological) example.

Example 2.1. Suppose we have two routes, with a single vehicle on each. Each vehicle has initial speed $v_i^0 = \bar{v}$ and some initial distance x_i^0 satisfying the assumption (2.11), such that it can perform a full deceleration ($\ddot{x}_i = \omega$) and come to a stop somewhere before the start of

the intersection at some point $x_i(\bar{v}/\omega) < B$. Consider the cost functional $J_{\alpha,\beta}$ with $\alpha = 1$ and $\beta = 0$, then it is easily seen that the optimal solution is to have both vehicles decelerate immediately as just described. However, this is not a feasible solution for (U).

There are different ways to resolve this issue. A possible approach is to require that all vehicle trajectories satisfy[2]

$$\dot{x}_i(t) \geq \epsilon \quad \text{for all } t \in [0, t_f], \tag{2.21}$$

for some $\epsilon > 0$, which ensures existence of $\tau(x_i)$ and $\xi(x_i)$, assuming t_f is sufficiently large. With this assumption, for a single vehicle per route and a cost functional of the form

$$J(x_i) = \int_0^{t_f} \Lambda(x_i(\tau), \dot{x}_i(\tau), \ddot{x}_i(\tau))\, \mathrm{d}\tau, \tag{2.22}$$

for some convex and quadratic function $\Lambda(x, v, u)$, it has been argued that (T) and (U) are equivalent [25, Theorem 1]. The argument there relies on the fact that the lower-level has a unique solution. However, as we will illustrate shortly hereafter, there are interesting problem settings for which this does not hold. Instead of assumption (2.21), we will explicitly restrict the set feasible trajectories to cross the intersection by introducing the problem variant

$$\begin{aligned}
T^*(s) := \min_x \ & \sum_{i \in \mathcal{N}} J(x_i) && && \text{(T*)}\\
\text{s.t.} \ & x_i \in D(s_i) && \text{for all } i \in \mathcal{N}, && \text{(T.1)}\\
& x_i(t) - x_j(t) \geq L && \text{for all } (i,j) \in \mathcal{C}, && \text{(T.2)}\\
& (x_i(t), x_j(t)) \notin \mathcal{E}^2 && \text{for all } \{i,j\} \in \mathcal{D}, && \text{(T.3)}\\
& \tau(x_i) < \infty && \text{for all } i \in \mathcal{N}, && \text{(T*.4)}\\
& \xi(x_i) < \infty && \text{for all } i \in \mathcal{N}. && \text{(T*.5)}
\end{aligned}$$

Theorem 2.1. *Assume problem* (T*) *is feasible and an optimal solution exists, then this problem is equivalent to the decomposed problem* (U).

Proof. We only have to argue that each feasible solution can be transformed into a feasible solution for the other problem.

Let (y, z) be a feasible solution of (U) and let x be the corresponding set of trajectories obtained by solving (L), which are not necessarily unique. Because x satisfies $x_i(y_i) = B$ and $x_i(z_i) = E + L$, we see that $\tau(x_i) < \infty$ and $\xi(x_i) < \infty$ are trivially satisfied. Because (L.3) and (L.4) are equivalent to (T.3), we see that x is also a feasible solution to (T*).

Conversely, let x be some solution to (T*), then $y_i := \tau(x_i)$ and $z_i := \xi(x_i)$ for all i are uniquely defined. Because x satisfies (T.3), we are sure that y and z satisfy (U.1) and x_r are obviously feasible for the lower-level problems. □

Remark 2.5. *In the above theorem, we assumed feasibility of* (T) *and the existence of an optimal solution. In general, these properties are not trivial to establish and rely on the compactness of the reachable sets, which are, roughly speaking, all the possible configurations a system can achieve after a fixed time; see [35, Section 4.5].*

Remark 2.6. *The constraints* (L.3), (L.4), (T*.4) *and* (T*.5) *are state constraints of a different type than those discussed earlier in Remark 2.3. Namely, $g(x(t)) \leq 0$ is imposed for all times $t \in [t_0, t_f]$. However, the constraint $x_i(y_i) = B$ only holds at some pre-specified intermediate time $y_i \in [t_0, t_f]$ and may be interpreted as some kind of "checkpoint". This case is not typically considered in the literature, but it is possible to reduce the problem into a canonical form that only has such equality constraints at the endpoints of the time interval, i.e., $x(t_0) = x_0$ and $x(t_f) = x_f$, see [11]. Whenever $t_f < \infty$, the constraints $\tau(x_i) < \infty$ and $\xi(x_i) < \infty$ together can be replaced by the equivalent endpoint constraint $x_i(t_f) \geq E + L$.*

[2]The authors of [26] refer to this assumption as "strongly output monotone".

Solving the decomposition. When the decomposition is sound, i.e., if both problems are indeed equivalent, it provides a good basis for developing alternative solution methods. However, the decomposed problem is not necessarily easier to solve. In general, the difficulty of solving (U) lies in the fact that F is a non-trivial function of the occupancy time slots and $\mathcal{T}(s_r)$ is not easily characterizable, e.g., as a system of inequalities.

For a single vehicle per route, so $(y_r, z_r) \equiv (y_i, z_i)$, the approach taken in [26] is to approximate both these objects as follows: they fit a quadratic function for $F(y_i, z_i, s_i)$ and they approximate the set of feasible occupancy slots by considering the polyhedral subset

$$\{(y, z) : y \in [T_i^l, T_i^h],\ l_i(y) \leq z \leq u_i(y)\} \subseteq \mathcal{T}(s_i), \tag{2.23}$$

for some earliest entry time T_i^l and latest entry time T_i^h and strictly increasing affine functions $u_i(\cdot)$ and $l_i(\cdot)$. They provide conditions that guarantee that solutions computed using this approximation method are feasible. To circumvent the need for such approximations, we will make some additional assumptions, which allows us to focus purely on the combinatorial aspect of the problem in the next section.

2.2.3 Delay minimization

We introduce a trajectory cost criterion that acts as a proxy of the amount of *delay* experienced by vehicles, by defining $J_d(x_i) := \tau(x_i)$. This choice makes the problem significantly simpler and avoids the need to approximate F, because we have

$$\begin{aligned} U(s) = \min_{y,z} \quad & \sum_{i \in \mathcal{N}} y_i \\ \text{s.t.} \quad & (y_i, z_i) \cap (y_j, z_j) = \varnothing \qquad \text{for all } \{i, j\} \in \mathcal{D}, \\ & (y_r, z_r) \in \mathcal{T}(s_r) \qquad \text{for all } r \in \mathcal{R}, \end{aligned}$$

which means that the only remaining difficulty is how to deal with feasibility of the lower-level problem. To also make this much simpler, we additionally assume that all vehicles start at full speed $v_i^0 = \bar{v}$ and that vehicles must enter the intersection at full speed, i.e., we add the constraint $\dot{x}_i(\tau(x_i)) = \bar{v}$ to trajectory optimization problem (T*) and, equivalently, we add $\dot{x}_i(y_i) = \bar{v}$ to the lower-level problem (L). These assumptions enable us to exclude the ends of occupancy time slots from consideration, as we will argue next.

Given some occupancy time slot schedule (y, z), every trajectory x_i in a solution x of the lower-level problems satisfies $\dot{x}_i(y_i) = \bar{v}$. Because the end times z_i are not involved in $J_d(\cdot)$, we can let each vehicle continue at full speed across the intersection, without inducing extra cost. This full-speed-crossing takes exactly $\sigma := (B - E)/\bar{v}$ time, so we can fix the end of the occupancy timeslot to be

$$z_i = y_i + \sigma, \quad \text{for all } i \in \mathcal{N},$$

without changing the cost of solutions. When vehicles cross the intersection at full speed, observe that $\rho := L/\bar{v}$ is such that $x_i(y_i + \rho) = x(y_i) + L = B + L$, so it is the time after which the next vehicle from the same route can enter the intersection. Because we might think of the intersection as some "machine" processing each vehicle one by one, we will refer to ρ as the *processing time*. In a similar vein, we will call σ the *switch-over time*, because it can be interpreted as the minimal time the machine needs between serving vehicles from distinct routes. We will return to this machine scheduling analogy in Chapter 5, where we will connect the network scheduling problem with the classical job-shop scheduling problem.

Since the end times z_i are now a trivial function of the start times y_i, to which we will refer as the *crossing times*. Consequently, we can limit ourselves to characterizing the set of feasible crossing times

$$\mathcal{T}_y(s_r) := \{y_r : (y_r, y_r + \sigma) \in \mathcal{T}(s_r)\}, \tag{2.24}$$

where $y_r + \sigma$ is to be understood as $\{y_i + \sigma : i \in \mathcal{N}_r\}$. Assuming that the initial vehicle states allow all vehicles to safely stop before the intersection, for example using the sufficient conditions of Assumption 2.1, these feasible crossing times permit a polyhedral characterization,

which means that it can be shown that $y_r \in \mathcal{T}_y(s_r)$ holds if and only if

$$a_i \leq y_i \quad \text{for all } i \in \mathcal{N}_r, \tag{2.25a}$$
$$y_i + \rho \leq y_j \quad \text{for all } (i,j) \in \mathcal{C} \cap \mathcal{N}_r, \tag{2.25b}$$

where $a_i := (B - x_i^0)/\bar{v}$ is the earliest time at which vehicle i can enter the intersection. A rigorous proof of this fact is outside the scope of the current chapter.[3] This polyhedral characterization cause the trajectory optimization problem to reduce to the *crossing time scheduling* problem

$$\min_{y} \quad \sum_{i \in \mathcal{N}} y_i \tag{C}$$
$$\text{s.t.} \quad a_i \leq y_i \qquad \text{for all } i \in \mathcal{N}, \tag{C.1}$$
$$y_i + \rho \leq y_j \qquad \text{for all } (i,j) \in \mathcal{C}, \tag{C.2}$$
$$(y_i, y_i + \sigma) \cap (y_j, y_j + \sigma) = \varnothing \qquad \text{for all } \{i,j\} \in \mathcal{D}. \tag{C.3}$$

A similar formulation has been previously proposed and analyzed [36]. This is a typical scheduling problem, which can for example be solved within the mixed-integer linear programming framework after encoding the *disjunctive constraints* (C.3) using the big-M technique. This methodology will be applied in the next section.

2.3 Crossing time scheduling

The conclusion of the previous section is that, after making certain assumptions, the trajectory optimization problem (T) reduces to a scheduling problem (C) of finding a schedule of crossing times y, determining the time at which each vehicle in the system crosses the intersection. Given such a schedule, corresponding trajectories can be computed relatively efficiently for each route separately using a direct transcription method. Hence, in the remainder of this chapter and the next chapter, the focus will be on solving the crossing time scheduling problem.

This problem is inherently a combinatorial optimization problem, for which a wealth of general solution techniques is available [15], including algorithms specifically tailored to problems that can be stated in terms of finding optimal time schedules [43, 17]. One of the most prominent distinction among such algorithms lies in whether it guarantees to find an optimal solution or not. Of course, it is desirable to find optimal solutions, but this is often not tractable in practice, because the number of feasible solutions typically explodes whenever larger instances are considered. In other words: enumerating all solutions and evaluating their objective value to find the best one is typically not achievable in any reasonable amount of time. This is the main motivation for the development of so-called heuristics, which discard the optimality guarantee in favor of speed.

This thesis will illustrate both types of approaches. In this section, we show how to leverage the general branch-and-cut framework by formulating (C) as a mixed-integer linear program and defining three types of cutting planes. We conclude this section by an evaluation of the running time improvement of these cutting planes. The next chapter focuses mainly on heuristics and machine learning techniques to tune heuristics to exploit specific structures often found in practical problem instances.

2.3.1 Branch-and-bound for MILP

A very common algorithmic idea in combinatorial optimization is the branch-and-bound strategy, in which the space of feasible solutions is systematically explored using a search tree by iteratively subdividing the feasible region into smaller subproblems (branching), computing bounds on the best possible objective value within each subproblem (bounding),

[3] The analysis would be very similar to that of Chapter 5. There is, however, a subtle difference in how the problem is stated there: whereas we provide initial *positions* here, the problem considered in Chapter 5 is based on initial *times* at which the vehicles enter the system.

and discarding those parts of the tree that contain subproblems whose bound proves them incapable of containing an optimal solution (pruning).

A typical application of this scheme is found in algorithms for solving mixed-integer programming problems. In such problems, we optimize over n real-valued decision variables y_i for which possibly a subset $\mathcal{I} \subseteq \{1, \dots, n\}$ is restricted to be integer-valued. A canonical example of such problems is a mixed-integer linear program, where the goal is to minimize

$$\min_y c^T y \text{ such that } Ay \leq b,\ y \in \mathbb{R}^n,\ y_j \in \mathbb{Z} \text{ for } j \in \mathcal{I}, \tag{MILP}$$

given some matrix $A \in \mathbb{R}^{m \times n}$ and vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. This surprisingly simple setup provides a powerful modeling toolkit to approach many combinatorial optimization problems, because it allows discrete choices to be modeled using integers. We already saw an example of this when we applied the big-M method in our direct transcription of the trajectory optimization problem.

The branch-and-bound algorithm for (MILP) is based on progressively constraining the integer decision variables as we move further from the root node in the search tree. At each node, the integer constraints are relaxed, producing a linear program that can be solved efficiently. The solutions of these relaxations provide the basis of the bounding step: whenever the objective of the relaxation at some node is higher than that of the currently best known feasible solution, the subtree at that node is pruned.

Reformulation. We show how to reformulate the crossing time scheduling problem (C) into (MILP). Note that we only have to rewrite the disjunctive constraints (C.3). For each conflict $\{i, j\} \in \mathcal{D}$, this constraints essentially encodes the crossing order of i and j, i.e., whether i crosses the intersection before j, or vice versa. We can rewrite these constraints using the big-M method by introducing a binary decision variable γ_{ij} for every conflict $\{i, j\} \in \mathcal{D}$, such that setting $\gamma_{ij} = 0$ corresponds to choosing the crossing order $i \to j$ and $\gamma_{ij} = 1$ corresponds to $j \to i$. To avoid redundant variables in a software implementation, it might be desirable to induce some arbitrary ordering of the conflicting vehicles by defining the index set

$$\bar{\mathcal{D}} = \{(i, j) : \{i, j\} \in \mathcal{D},\ r(i) < r(j)\}. \tag{2.27}$$

With this definition, we obtain the reformulation

$$\begin{aligned}
\min_{y,\gamma} \quad & \sum_{i \in \mathcal{N}} y_i & & \\
\text{s.t.} \quad & a_i \leq y_i & & \text{for all } i \in \mathcal{N}, \\
& y_i + \rho \leq y_j & & \text{for all } (i, j) \in \mathcal{C}, \\
& y_i + \sigma \leq y_j + \gamma_{ij} M & & \\
& y_j + \sigma \leq y_i + (1 - \gamma_{ij}) M & & \text{for all } (i, j) \in \bar{\mathcal{D}}, \\
& \gamma_{ij} \in \{0, 1\} & &
\end{aligned} \tag{C'}$$

where $M > 0$ is some sufficiently large number.

Note that this reformulating opens up the possibility to leverage the collective effort that has gone into developing fast general solvers for this problem class: many modern solvers, e.g., SCIP [7] (academic) or Gurobi [19] (commercial), employ specialized interal heuristics and techniques to derive better bounds and thus achieve more pruning of the search tree to speed up the solving process. Furthermore, a wide variety of software tooling is available. For example, we used the AMPL modeling language to write the above formulation in a solver-agnostic specification and use the amplpy[4] package to call the solver from the comfort of Python.

2.3.2 Problem-specific cutting planes

Although the branch-and-bound approach guarantees to find an optimal solution, it does not provide any guarantees on the required running time. Roughly speaking, the running

[4] `https://amplpy.ampl.com/`

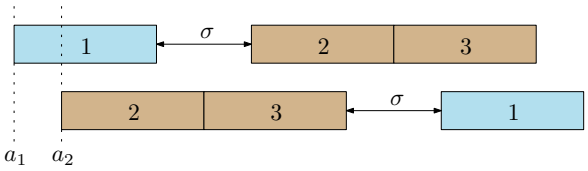


Figure 2.4: Illustration of the two possible sequences of vehicles in Example 2.2.

time depends heavily on the efficiency of the bounding step in pruning the search tree, which might vary wildly among equivalent formulations. A technique that is often used to make bounding more efficient is to add so-called *cutting planes*. The basic idea is that we can almost always introduce additional inequalities without changing the set of feasible solutions

$$\{y : Ay \leq b, A'y \leq b', y_j \in \mathbb{Z}, j \in \mathcal{I}\} = \{y : Ay \leq b, y_j \in \mathbb{Z}, j \in \mathcal{I}\}, \tag{2.28}$$

while achieving more efficient bounding. There are general-purpose schemes for adding such cutting planes $A'y \leq b$, but it often makes sense to also use insights into the specific problem at hand to derive problem-specific cutting planes. The branch-and-bound framework with cutting planes is colloqially referred to as *branch-and-cut*.

We will now illustrate the branch-and-cut methodology by deriving three types of cutting planes for (C'). The first type is related to some kind of redundancy in the encoding of feasible crossing orders. Observe that the constraints $y_i + \rho \leq y_j$ cause a fixed order of crossing for all vehicles on the same routes. Hence, let $pred(i)$ denote the set of all vehicles on route $r(i)$ that cross the intersection strictly before vehicle i and let $succ(i)$ denote those that cross strictly later, so we have

$$pred(i) := \{(r(i), k) : k > k(i)\}, \tag{2.29a}$$
$$succ(i) := \{(r(i), k) : k < k(i)\}. \tag{2.29b}$$

Suppose we have some solution (y, γ) that satisfies $\gamma_{ij} = 0$ for some conflict pair $(i, j) \in \bar{\mathcal{D}}$, so $i \to j$, then it is clear that γ must also satisfy

$$\gamma_{pq} = 0 \quad (\text{so } p \to q) \qquad \text{for all } p \in pred(i),\ q \in succ(j).$$

Using the big-M method, we can equivalently encode this as

$$\sum_{\substack{p \in pred(i) \\ q \in succ(j)}} \gamma_{pq} \leq \gamma_{ij} M. \tag{2.30}$$

Every feasible (y, γ) must satisfy the above inequality for every $(i, j) \in \bar{\mathcal{D}}$, so we can safely add them to (C') without changing the problem. We refer to these inequalities as the *transitive cutting planes*.

Next, we present some insights into the structure of optimal solutions, which serves as the basis for two additional types of cutting planes. Let us first consider some basic examples to sharpen our intuition.

Example 2.2. Consider two routes having one and two vehicles, respectively. Instead of $(1,1), (2,1), (2,2)$, we will use the labels $1, 2, 3$ to keep notation clear. We are interested in how the earliest crossing times a_i influence the order of the vehicles in an optimal schedule. We set $a_1 = 0$, without loss of generality, and assume that $a_3 = a_2 + \rho$. Suppose $a_1 = a_2$, then we see that the order $2, 3, 1$ is optimal, which resembles some sort of "longest chain first" rule. Now suppose that $a_1 < a_2$. For $a_2 \geq a_1 + \rho + \sigma$, the sequence $1, 2, 3$ is simply optimal. For $a_2 < a_1 + \rho + \sigma$, we compare the sequence $1, 2, 3$ with $2, 3, 1$, which are illustrated in Figure 2.4. The first has $\sum_i y_i = (\rho + \sigma) + (\rho + \sigma + \rho) = 3\rho + 2\sigma$, while the second sequence has $\sum_i y_i = a_2 + (a_2 + \rho) + (a_2 + \rho + \rho + \sigma) = 3a_2 + 3\rho + \sigma$. Therefore, we conclude that the second sequence is optimal if and only if

$$a_2 \leq \sigma/3, \tag{2.31}$$

which roughly means that the "longest chain first" rule becomes optimal whenever the earliest arrival times are "close enough".

In the previous example, we see that it does not make sense to schedule vehicle 1 between vehicles 2 and 3, because that would add unnecessary switch-over time σ. This raises the natural question whether splitting such *platoons* of vehicles is ever necessary to achieve an optimal schedule. To answer this question, let us first give a precise definition of platoons, before slightly generalizing the example.

Definition 2.1. A sequence of consecutive vehicles $(r, l+1), (r, l+2), \ldots, (r, l+n)$ from some route r is called a *platoon* of size n if and only if

$$a_{(r,k)} + \rho = r_{(r,k+1)} \qquad \text{for all } l < k < l + n. \tag{2.32}$$

We say that the platoon is *split* in some schedule y, if

$$y_{(r,k)} + \rho < y_{(r,k+1)} \qquad \text{for some } l < k < l + n. \tag{2.33}$$

Example 2.3. Suppose we have two routes $\mathcal{R} = \{A, B\}$, each having exactly one platoon, denoted as $P_A = ((A,1), \ldots, (A, n_A))$, $P_B = ((B,1), \ldots, (B, n_B))$. To simplify notation, we write $a_A = a_{(A,1)}$ and $a_B = a_{(B,1)}$. We assume $a_A = 0$, without loss of generality, and suppose that $n_A < n_B$ and $a_A \leq a_B < n_A\rho + \sigma$. Consider the ways the two platoons can merge by splitting A. Let k denote the number of vehicles of platoon A that go before platoon B and let $\sum y_i(k)$ denote the corresponding sum of crossing times. See Figure 2.5 for an illustration of the situation in case of $a_A = a_B$. For $0 < k \leq n_A$, we have

$$\sum_{i \in \mathcal{N}} y_i(k) = \max\{\sigma, a_B - k\rho\}(n_B + n_A - k) + \sigma(n_A - k) + \sum_{j=1}^{n_A+n_B} (j-1)\rho,$$

so when platoon A goes completely before platoon B, we get

$$\sum_{i \in \mathcal{N}} y_i(n_A) = \sigma n_B + \sum_{j=1}^{n_A+n_B} (j-1)\rho,$$

since $\max\{\sigma, a_B - n_A\rho\} = \sigma$ by the assumption on a_B. It is easily seen that we have $\sum y_i(k) > \sum y_i(n_A)$ for $0 < k < n_A$, so in other words, if we decide to put at least one vehicle of platoon A before platoon B, it is always better to put all of them in front. As we will see after this example, this principle holds more generally.

For $k = 0$, so when we schedule platoon A completely after platoon B, the total completion time becomes

$$\sum_{i \in \mathcal{N}} y_i(0) = a_B(n_A + n_B) + \sigma n_A + \sum_{j=1}^{n_A+n_B} (j-1).$$

Comparing this to (2.34a), we conclude that placing B in front is optimal whenever

$$a_B \leq (n_B - n_A)\sigma/(n_A + n_B),$$

which directly generalizes the condition (2.31) that we derived for the case with $n_A = 1$ and $n_B = 2$. (end of example)

The example shows that, when we decide to put one vehicle of a platoon before another platoon, it is always better to put all vehicles of the platoon in front. In other words, whenever a vehicle can be scheduled immediately after its predecessor, this should happen in any optimal schedule. It turns out that this property holds more generally, as stated by the following result.

Theorem 2.2 (Platoon Preservation [36])**.** *If y is an optimal schedule for* (C)*, satisfying $y_{i^*} + \rho \geq a_{j^*}$ for some $(i^*, j^*) \in \mathcal{C}$, then j^* follows immediately after i^*, so $y_{i^*} + \rho = y_{j^*}$.*

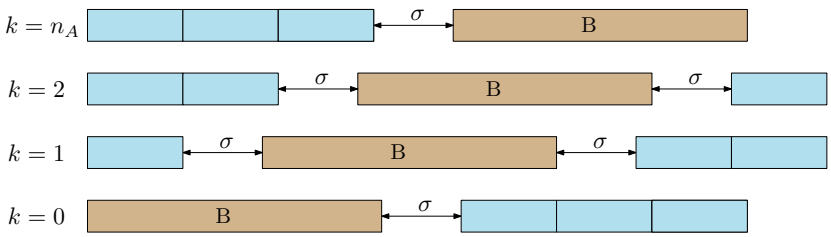


Figure 2.5: Different ways to split platoon A, regarding Example 2.3, assuming equal earliest arrival times $a_A = a_B$ with $n_A = 3$ vehicles in platoon A and some arbitrary number of vehicles n_B in platoon B.

We will now use this result to formulate two more types of cutting planes. First, we try to directly model this necessary condition for optimality. Therefore, we introduce a binary variable $\delta_{ij} \in \{0, 1\}$, for every conjunctive pair $(i, j) \in \mathcal{C}$, which we want to satisfy

$$\delta_{ij} = 0 \iff y_i + \rho < a_j, \tag{2.34a}$$
$$\delta_{ij} = 1 \iff y_i + \rho = a_j. \tag{2.34b}$$

Again, this can be enforced using the big-M method, by adding the constraints

$$y_i + \rho < a_j + \delta_{ij} M, \tag{2.35a}$$
$$y_i + \rho \geq a_j - (1 - \delta_{ij}) M. \tag{2.35b}$$

Now observe that the statement of Theorem 2.2 applied to (i, j) is equivalent to the inequality

$$y_i + \rho \geq y_j - (1 - \delta_{ij}) M. \tag{2.36}$$

We refer to these cutting planes as *necessary conjunctive cutting planes*.

Using the definition of δ_{ij}, we can derive one more type of cutting planes on the disjunctive decision variables γ. Whenever $\delta_{ij} = 1$, Theorem 2.2 implies that we have $i \to k$ and $j \to k$ for every other vehicle $k \in \mathcal{N}$ on a different route $r(k) \neq r(i) = r(j)$, which can be enforced by adding the constraints

$$\delta_{ij} + (1 - \gamma_{ik}) + \gamma_{jk} \leq 2, \tag{2.37a}$$
$$\delta_{ij} + \gamma_{ik} + (1 - \gamma_{jk}) \leq 2. \tag{2.37b}$$

We will refer to these as the *necessary disjunctive cutting planes*.

Runtime benchmark. We conclude this section with an evaluation of the running time of the branch-and-bound approach. Since the running time is primarily determined by the total number of vehicles in the system, we consider problem instances with two routes and report running times as a function of the number of vehicles per route. Problem instances are generated with fixed processing time $\rho = 4$, switch-over time $\sigma = 1$, and randomly generated earliest arrivals times a_i. A detailed discussion of the distribution of a_i that we used is deferred to the end of the next chapter, where a more extensive comparison will be provided.

To illustrate the benefit of each proposed cutting plane, we also include results for the case without any cutting planes. To keep the total computational effort within reasonable limits, we impose a time limit of 60 seconds per instance. Consequently, some care must be taken when interpreting the average running time, because some observations correspond to cases in which the algorithm reaches the time limit.[5] For this benchmark, we used the Gurobi solver version 11.0.2 on a system with a 13th Gen Intel i5-13600K CPU and 32GiB of RAM. Figure 2.6 shows the average running time for the three types of cutting planes. Observe that the necessary conjunctive cutting planes provide the most significant runtime improvement.

[5] In statistics, such observations are called *censored* and there are generally better ways of handling these. However, we do not need such rigorous analysis for the argument here.

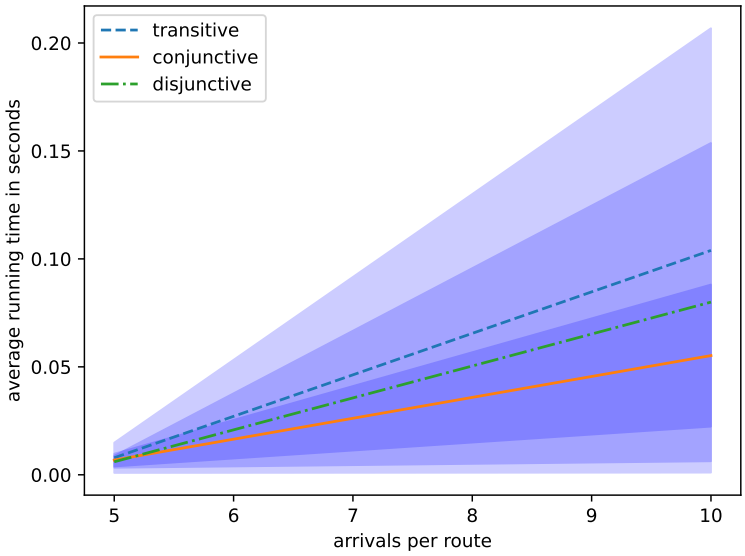


Figure 2.6: [current figure is placeholder] The average running time of the branch-and-cut procedure is plotted as a function of the number of arriving vehicles per route, for each of the three indicated cutting planes. Each average is computed over 20 problem instances. All instances use $\rho = 4$ and $\sigma = 1$. The earliest arrival times for vehicles are generated using the bimodal exponential interarrival time model from Section 3.4 with parameters $p = 0.5, \mu_s = 0.1, \mu_l = 10$. This figure clearly shows that the conjunctive cutting planes provide the most runtime improvement.

2.4 Notes and references

The single intersection model is mostly based on the model described in the PhD thesis of Hult [24, Chapter 3]. Due to the simple rectangular geometry of the vehicles and the fact that routes are straight, it is easy to compute the conflict area in configuration space, for which we provide details in Appendix A. For general curved trajectories, the computation is more involved, see for example [33, Appendix A] and [34, Section 2.2].

The bilevel decomposition and approximation scheme of Section 2.2.2 are further detailed in [26, 25]. In proving that the decomposed problem is equivalent to the original trajectory optimization problem, they state that the lower-level problem has a unique solution. We emphasize that this depends on *strict* convexity of the objective function. For the problem without state constraints, this has been rigorously proven in [20, Theorem 5.1, part (V)], where uniqueness indeed depends on the positive definiteness of the matrix R defining the quadratic component $u^T Ru$ of the running cost function with respect to the control variable.

At various points in this chapter, we included some comments from the optimal control perspective. For a textbook introduction of optimal control theory, we recommend the book of Liberzon [35]. The focal point in their presentation is the Pontryagin maximum principle, which provides neccessary conditions for optimality for optimal control problems. When dealing with state-constrained problems, the most relevant overview of results we could find is the survey of Hartl et al. [21]. We note that, although the theory for dealing with state constraints (à la Pontryagin) is far from complete, in practice, they can be successfully dealt with in numerical approaches, often involving some sort of penalty function.

A solid introduction to the algorithmic foundations for integer programming is provided by the book [8], whose introductory chapter also contains a clear description of the branch-and-cut methodology. For an introduction of integer programming with a focus on solving scheduling problems, we like to refer to [43, Appendix A]. Note that decomposition techniques similar in nature to the decomposition of Section 2.2.2 have a relatively long history in mathematical programming. Well-known examples in the context of mixed-integer linear

programming include Dantzig-Wolfe decomposition and Benders' decomposition.[6] Such techniques have been applied in air traffic scheduling [38] and job-shop problems arising in general traffic scheduling problems [32]. Interestingly, these works also propose an alternative ("noncompact") MILP formulation that does not depend on the big-M trick.

[6]Fun fact: this famous method is named after Jacques Benders [2], who was a professor here at TU/e.

Chapter 3

Scheduling as sequence learning

Most methods that are based on some kind of branch-and-bound tree search are guaranteed to find an optimal solution, but as we illustrated with our benchmark in Section 2.3, without further modifications, the running time still scales very poorly with increased instance sizes. Once we start considering multiple intersections, this scaling issue is expected to become even more prominent, so we first try to obtain fast algorithms for the single intersection crossing time scheduling problem with large numbers of vehicles. Specifically, in this chapter, we are interested in developing heuristics to obtain good schedules, with close to optimal total travel delay, in limited time.

Before we discuss our methodology for solving the crossing time scheduling problem (C), Section 3.1 first provides some background on the general idea of leveraging machine learning techniques to solve combinatorial problems. We introduce the central idea of treating problem instances and solutions as training data points and mention some key works to illustrate the genreal methodology.

Our own machine learning approach hinges on the ability to encode candidate solutions for the crossing time scheduling problem (C) as a sequence of route indices, which is discussed in Section 3.2. The idea is that we only have to decide in which order vehicles are able to cross the intersection, and since the relative route of vehicles on the same route is already fixed, this leaves us with ordering the routes. It is straightforward to show that the corresponding crossing times, given this crossing order, follow as the solution to a linear program. Additionally, we show that partial solutions provide a certain lower bound on the crossing times.

The problem is reduced to a sequence learning problem.

Next, we show in Section 3.3 how to construct a probabilistic sequence model over schedules, by assigning a conditional probability to every candidate solution sequence, conditional on the problem instance. We propose two different parameterizations of the model. One parameterization is based on the Platoon Preservation Theorem 2.2. The other is based on a recurrent neural network encoding of the crossing time lower bounds.

We show how to use randomly generated problem instances to fit the model parameters. There are generally two main approaches here. The most straightforward way is to use an existing solver to compute optimal solutions, then use these as training data points in a supervised learning setting. However, this assumes that we are able to solve the combinatorial problem in the first place. An alternative approach is to try to learn how to construct such optimal sequences by trial-and-error, by evaluating the current model to determine the next route in the sequence, and then updating the model parameters based on an intermediate score, which is best understood in the framework of reinforcement learning.

3.1 Machine learning for combinatorial optimization

Recent years have seen an increased research interest in using modeling techniques and methods from the machine learning (ML) community to solve combinatorial optimization problems (CO). Very generally speaking, we can think of CO problems as being concerned with finding an optimal configuration vector $v \in V \subset \mathbb{R}^n$ from a finite set of candidates V that minimizes some cost function $f : V \to \mathbb{R}$. This problem can simply be written as

$$\min_{v \in V} f(v). \tag{3.1}$$

Since V is assumed to be finite, a straightforward solution is to just evaluate $f(v)$ for all candidates $v \in V$ and then output one configuration with minimal cost. However, in all but trivial problem instances, this is simply not feasible in any reasonable amount of time, which motivates the development of better methods.

In almost all practically relevant CO problems, the function f exhibits some kind of regularity. Therefore, when developing algorithms for CO problems, a good strategy is to look for such structure in the problem that can be exploited algorithmically. This structural insight can be precise, like in Theorem 2.2. More generally, a class of well-known and illustrative examples algorithms that take advantage of well-defined problem structure are branch-and-bound schemes, which can be understood as systematically exploring the search space V by keeping track of a search tree (branching). Simultaneously, these algorithms try to prove that certain parts of the search tree are suboptimal (bounding), such that they can be discarded from the search (pruning). Apart from such well-defined problem structure, algorithms are often based on some hard-to-define intuition about what optimal solutions should look like. It is precisely this kind of intuition that is a good candidate for methods from ML and especially deep learning models can be valuable tools.

Remark 3.1. *We assume that V is finite, which simplifies the current discussion a lot, but note that some typical combinatorial problems have infinite feasible solutions. Furthermore, note that the distinction between* continuous *optimization and* combinatorial *optimization is not always as sharp as it may seem. We could say that a problem is combinatorial if at least one component v_i of the configuration $v \in \mathbb{R}^n$ is restricted to lie in domain of countable size. For example, this is the case in mixed-integer linear programming, where some variables are restriced to be integers. However, it may still happen that a problem that is continuous under this definition can actually be treated as a combinatorial problem. For example, the feasible set of a linear program is a convex polytope, which is certainly not finite set, but Dantzig's simplex algorithm shows that the optimal solution must be one of the finitely many vertices of the convex polytope.*

Motivation. Two main motivations for applying ML for solving CO problems have been identified in the current literature [6]. One approach is to replace certain expensive computations of an existing algorithms such as branch-and-bound with fast approximations provided by some ML model. Please note that, although ML is approximate in nature, this does not necessarily mean that there cannot be any optimality guarantees. Good example of this approach are for example [16], where the branching decision in branch-and-bound is approximated, and the line of work related to [50], where ML models are employed for cutting plane selection. In both examples, as long as the model provides valid branching decisions or valid cutting planes, the algorithm is still guaranteed to obtain an optimal solution, when it exists. Such methods that integrate with existing CO algorithms have been called *joint* methods.

Secondly, when existing algorithms' performance is not satisfactory, we could try to employ ML to explore the space of algorithms to find better algorithms. These types of approaches can be understood as constructing a solution from scratch or in terms of predicting the optimal solution, given the problem instance. Such methods have been called *principal* methods. The method that we will propose in Section 3.3 belongs to this class.

In both cases, the main idea is that *ML can be used to discover certain regularities of the problem automatically.* Therefore, one of the central ideas is to treat problem instances and

their solutions and training data for a machine learning model [6]. This training data is used to fit a machine learning model and the hope is that it captures our intuition in this way.

Algorithm design as optimization problem. The process of designing an algorithm itself may be considered as an optimization problem. Let $i = (V, f) \in \mathcal{I}$ denote an instance of a combinatorial problem from some set $\mathcal{I}$ of problem instances and let $\pi \in \Pi$ denote some algorithm for solving it. We use Π to denote some class of algorithms; we will make this more precise later. Let $m : \mathcal{I} \times \Pi \to \mathbb{R}$ denote some performance measure of the algorithm π applied to problem instance i, which may include aspects such as eventual objective value, bounds on optimality, running time, usage of resources such as memory, etc. Given some instance i, the goal of the algorithm developer can be succintly formulated as

$$\min_{\pi \in \Pi} m(i, \pi). \tag{3.2}$$

However, this single-instance situation is not very typical. More often we have a class of instances $\mathcal{I}$ that we want to consider, for example, the class of all traveling salesman problems. Assume that $m(i, \pi)$ measures the running time of the algorithm, then the classical worst-case analysis leads to the optimization problem

$$\min_{\pi \in \Pi} \max_{i \in \mathcal{I}} m(i, \pi). \tag{3.3}$$

Problem distribution. In principle, we want to devise algorithms that perform as good as possible for all possible instances. However, in practical application, we are often dealing with very similar problems, which we could model as a probability distribution. To illustrat this, imagine a delivery company in Montreal that needs to solve many instances of the Traveling Salesman Problem (TSP) every day [6]. The patterns of these TSPs are not arbitrary but have structure: many customers are downtown, few on top of Mount Royal; the street layout gives something close to ℓ_1 metric (grid-like), etc. The company is not interested in solving all possible TSPs, just their particular kind. There is some true, underlying but unknown probability distribution over the kinds of instances the company will face—call it the problem instance distribution P. This distribution captures the problem instances the company actually cares about, as further illustrated by Figure 3.1. Hence, the company is actually interested in solving the following problem

$$\min_{\pi \in \Pi} \mathbb{E}_{i \sim P} \left[m(i, \pi) \right]. \tag{3.4}$$

The distribution P is generally not known explicitly, so we need to resort to using samples D_{train} (e.g., obtained as historical problem instances) in order to estimate (3.4) and solve instead

$$\min_{\pi \in \Pi} \sum_{i \in D_{\text{train}}} \frac{1}{|D_{\text{train}}|} m(i, \pi). \tag{3.5}$$

Algorithm execution as MDP. To make the optimization outlined above more concrete, we first need to choose a representation for the set of algorithms Π over which we are optimizing. In principle, we could take Π to consist of all C programs, or all Python programs, for example. But in practice, real-world code runs on a complex stack of compilers, interpreters, and hardware optimizations, which makes reasoning about such a space nearly impossible. A more manageable approach is to work with simplified models of computation. A common choice in the ML for CO literature is to represent algorithms as Markov decision processes (MDPs). This perspective naturally connects algorithm optimization to reinforcement learning methods, which many existing works leverage. In what follows, we show how the MDP model arises in this setting and explain why it provides a particularly natural abstraction.

Our starting point is to model the execution of an algorithm as a particular state transition sequence in some deterministc finite-state automaton (DFA) as follows. Let $\mathcal{S}$ be the finite set of states, with $\mathcal{I} \subseteq \mathcal{S}$ denoting the initial states, corresponding to problem instances, and

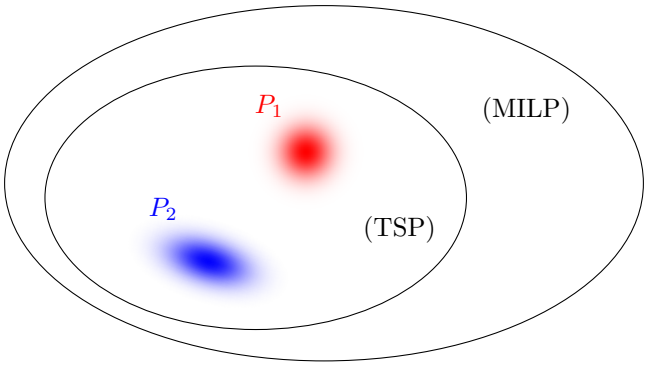


Figure 3.1: Illustration of a typical hierarchical relationship between classes of combinatorial problems: instances of the traveling salesman problem (TSP) with linear objective functions can be understood as specific instances of mixed-integer linear programs (MILP). Furthermore, in certain practical contexts, it might make more sense to consider some distribution of TSP instances (illustrated here as planar Gaussian distributions P_1 and P_2).

$\mathcal{O} \subseteq \mathcal{S}$ denoting the final, absorbing states that represent possible outputs. Once a final state is reached, the simulation terminates. The remaining states $\mathcal{S} \setminus (\mathcal{I} \cup \mathcal{O})$ capture the *internal state* of the algorithm, abstracting the relevant data structures in memory. Let $\mathcal{A}$ denote the finite non-empty set of *actions*.[1] Some transition function $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ specifies how the automaton evolves from one state to the next, given some input symbol. An algorithm is now modeled by a function $\pi : \mathcal{S} \to \mathcal{A}$ that provides the action to be taken from each state. Given some initial state $i \in \mathcal{I}$, we let $o_\pi(i)$ denote the final state that is reached by repeatedly choosing the next action according to π and transitioning to the next state according to p, assuming the process halts rather than cycling indefinitely. In this view, π parameterizes the automaton's execution on instance i, just as program code determines the state transitions of a computer. In this model, the performance of running algorithm π on instance i is mostly determined by the final state reached, so we could write

$$m(i, \pi) = m(o_\pi(i)). \tag{3.6}$$

The evaluation of $m(i, \pi)$ can thus be understood as executing algorithm π on input i and reading off the relevant information from the resulting state. To support this, states may carry auxiliary data about the execution, such as the total number of transitions taken or bounds on the quality of the solution returned.

For algorithm optimization problems such as (3.3) or (3.4), the objective now becomes to identify the function π that yields the best performance. Note that the definition of the intermediate states now becomes a key decision, because fixing a particular state space effectively restricts the optimization to a specific class of algorithms. In this way, the algorithm designer can incorporate prior knowledge or intuition about what kind of algorithm is best suited to the current problem at hand. For example, if states represent permutations of an initial solution, the resulting transition functions naturally model local search algorithms. To make this concrete, consider the traveling salesman problem: any permutation of the vertices corresponds to a valid tour, so the state space consists of all possible vertex permutations. The function π then specifies how the algorithm selects the next permutation from the neighborhood of the current one, capturing the essence of the local search process.

It might be helpful to generalize the DFA model by introducing stochastic transitions as follows. The transition function p becomes a mapping to the probability space over states, which is simply the simplex $\Delta(\mathcal{S})$ over the state space, so we write $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$. In this generalized model, we can use stochastic transitions to model certain parts of the computation that are too complex to model explicitly. This has for example been done in

[1] Note that the terminology "input alphabet" is actually more common for DFAs, because this and similar models are interpreted to "accept" or "reject" certain formal languages within the field of computational complexity. Saying "action space" makes more sense here because we will turn the DFA into an MDP.

previouw works on ML for CO that consider joint methods, so in which some existing solver is augmented by replacing a specific component with a ML model. For example, we might want to take some existing solver based on the branch-and-bound scheme as the basis for our algorithm, but only want to parameterize a specific component to optimize over, such as the cutting plane selection, for example. In this example, the actions $\mathcal{A}$ would correspond, roughly speaking, to selecting some valid cutting plane, upon which the state transition p models the whole of the execution of the branch-and-bound solver until the next time a cutting plane has to be selected.

With stochastic state transitions, the DFA can be understood as a MDP after we define an appropriate (deterministic[2]) reward function $r : \mathcal{S} \to \mathbb{R}$. The logical choice is to define a single final reward, which is the negative performance measure at the final state $r(o_\pi(i)) = -m(o_\pi(i)) = -m(i, \pi)$. From this perspective, function π is a deterministic policy function, which we want to optimize. As is commonly done in the reinforcement learning setting, it might be helpful to generalize π to be a probability distribution over next actions, which can be written as $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, which makes it a stochastic policy. The reasons for doing this are mainly of a technical nature, for example to promote exploration or to make sure the policy gradient exists. However, in some cases it could also make sense to consider randomized algorithms. The reasons for modeling a parameterized algorithm as a MDP can be summarized as follows:

- State transition probabilities $p(s, a)$ can be used to model fixed parts of the algorithm that are too complex and expensive to consider as they are.
- Stochastic policies $\pi(s)$ are allowed for practical reasons. Alternatively, we could interpret action probabilities as modeling the uncertainty about which actions are optimal.
- The final reward models the performance of some algorithm π applied to a problem instance $s_0 \in \mathcal{I}$.

We introduce yet another perspective on the algorithm model that could be helpful in some contexts. Suppose we are given an MDP specified by the tuple $(\mathcal{S}, \mathcal{A}, p, r)$, and some policy π. Let s_0 denote some initial state and let τ denote some particular episode of states and actions, so it is a tuple

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T). \tag{3.7}$$

The probability of this trajectory can be decomposed as follows

$$p(\tau) = \rho_0(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t), \tag{3.8}$$

where $\rho_0(\cdot)$ denotes the distribution of the initial state.

Remark 3.2. *Taking this paradigm to the extreme, one particular research direction that should be mentioned is that of neural turing machines [18], in which neural networks are used to parameterize the execution of a turing machine. It has been demonstrated that simple algorithms like sorting or information retrieval can be learned in this way. Although this is a very powerful idea, it has been observed that such models are very hard to train, which is probably because it does have a very weak prior. Hence, it makes sense to consider parameterizations that are better tailored to specific problems, by incorporating some prior. After we formulate our own methodology, we will come back to this by explaining why our own model can be understood as having a stronger prior.*

Remark 3.3. *Regarding the use of probabilities in the model, observe that the transition probabilities in the MDP do not in model some sort of inherent randomness of the system. The underlying computational system is fully observable and deterministic, so we can always*

[2]In general MDPs, the reward may also be a random function of state-action pairs (s, a), but this does not make much sense in the combinatorial optimization setting.

know the precise state, in principle. This means that there is no so-called aleatoric uncertainy *about the system. However, as we discussed above, it might be helpful to model certain parts of the computation using probability models, which means we are considering so-called* epistemic uncertainty *about the system, which can be generally defined as any uncertainty that can be reduced by knowing more or obtaining more precise information.*

Reference	Problem focus	Contribution / Method
Bengio et al. [6]	General CO problems	Survey on using ML for CO, framing learning to optimize
Mazyavkina et al. [40]	General CO problems	Survey on using ML for CO, with a focus on reinforcement learning
Dai et al. [9]	Graph problems	Deep RL with graph embeddings to construct solutions
Vinyals et al. [54]	TSP	Pointer networks for sequence-based combinatorial problems
Kool et al. [31]	Routing problems	Attention model trained with RL for solving TSP, VRP
Gasse et al. [16]	MILP	Approximate strong branching policy with GNNs in branch-and-bound
Tang et al. [50]	ILP	Attention-based model for cutting plane selection trained with RL

Table 3.1: Key references on machine learning for combinatorial optimization.

Policy optimization. We can identify two main learning paradigms for finding good algorithms π, which also have natural ties to the two main motivations discussed above. Suppose that we have some heavy computation $f(\cdot)$ which we want to replace with a fast approximation $\hat{f}(\cdot)$. A natural strategy is to collect samples of $f(x)$ for a set of inputs $x \in X$ and then consider the supervised learning problem of minimizing some loss function $\ell(f(x), \hat{f}(x))$. We could say that this method is *learning from demonstration*. Please note that this way, it could happen that policies that are not close to the expert perform well nevertheless, because they found some alternative good strategy.

Alternatively, we could try to search the space of possible algorithms more directly, by trying to optimize the expected reward.

Beam search can be understood as searching over execution trajectories of algorithm actions parameterized by probability distributions. In some sense, the beam search procedure itself can also be regarded as just another algorithmic prior on a higher level of abstraction. From this perspective, we could even parameterize some parts of the beam search procedure itself and subject this to learning. Think for example of beam width.

Policy parameterization. The strength of the ML for CO paradigm stems from the ability of high-capacity neural network-based models to capture patters in the problem. Therefore, instead of directly optimizing over some class of algorithms Π, a common strategy is to use a differentiable paramterization of algorithms, so where some real vector $\theta \in \mathbb{R}^n$ is the parameter of policy π_θ. The corresponding algorithm optimization problem with expected cost can then be written as

$$\min_{\theta} \mathbb{E}_{i \sim P}[m(i, \pi_\theta)]. \tag{3.9}$$

These parameters can for example be the weights and biases in a deep neural network. However, using neural networks this brings several difficulties with it. For example, CO problems often involve sets instead of ordered sequences. Therefore, the neural network should be invariant to the ordering of the elements in the input. One early attempt was the pointer network [54], which is a sort of attention mechanism. Later, more general attention mechanisms like the transformer architecture became mainstream.

Generalization. One of the main challenges is obtaining algorithms that generalize to instances not seen during training. For this reason, it makes sense to use a set of validation samples D_{valid} to gauge how well the learned policy generalizes to unseen instances and *unseen internal states*. There are generally two important axes of comparison: larger instances and instances with different structure.

3.2 Sequence representation of optimal schedules

In the rest of this chapter, we will present a concrete methodology to apply the general ML for CO methodology presented in the previous section to the crossing time scheduling problem (C) that we introduced in the previous chapter. To this end, it will be beneficial to work with a compact representation of schedules and the goal of this section is to first show that optimal schedules can actually be represented as a sequence of route indices. This allows us to formulate the scheduling problem in terms of a particularly simple MDP, which is done in Section 3.3.

For ease of reference, let us now restate the MILP reformulation of the original crossing time scheduling problem

$$
\begin{array}{lll}
\min\limits_{y,\gamma} & \sum\limits_{i\in\mathcal{N}} y_i & \\
\text{s.t.} & a_i \le y_i & \text{for all } i \in \mathcal{N}, \\
& y_i + \rho \le y_j & \text{for all } (i,j) \in \mathcal{C}, \\
& \left.\begin{array}{l} y_i + \sigma \le y_j + \gamma_{ij} M \\ y_j + \sigma \le y_i + (1-\gamma_{ij}) M \\ \gamma_{ij} \in \{0,1\} \end{array}\right\} & \text{for all } (i,j) \in \bar{\mathcal{D}}.
\end{array}
\tag{C'}
$$

Observe that this problem has infinitely many feasible solutions, because the decision variables $y_i : i \in \mathcal{N}$, representing the crossing times, are real-valued. We will show that there is a finite representation of optimal solutions. Recall that the binary variables γ_{ij} encode the order in which all vehicles cross the intersection, to which we will simply refer as the *crossing order*. We essentially show that the crossing order contains all the necessary information to encode each optimal solution y. Specifically, after fixing some crossing order by setting binary variables γ_{ij}, we obtain a linear program, which can be shown to have a unique solution whenever it is feasible. However, not all settings of γ_{ij} lead to a feasible linear program, because some assignments correspond to cycles of inequalities. To make the argument more precise, it will be convenient to introduce the *disjunctive graph* representation, which is a common formalism used to encode scheduling problem instances and candidate solutions.

Disjunctive graph. The basic idea is to encode the inequality constraints of (C') using weighted arcs in a directed graph. Each such arc $i \to j$ with weight $w(i,j)$ encodes the inequality $y_i + w(i,j) \le y_j$. Therefore, the vehicle indices $\mathcal{N}$ form the main nodes of the graph. Furthermore, each node i has a dummy node i' with $y_{i'} = 0$ and a directed arc $i' \to i$ with weight $w(i',i) := a_i$ to model the earliest arrival time constraint $a_i \le y_i$. Next, each constraint $y_i + \rho \le y_j$ for a pair $(i,j) \in \mathcal{C}$ is encoded by a so-called *conjunctive* arc $i \to j$ with weight $w(i,j) := \rho$. The remaining constraints encode that exactly one of the constraints $y_i + \sigma \le y_j$ and $y_j + \sigma \le y_i$ must hold for each conflict $\{i,j\} \in \mathcal{D}$. In the disjunctive graph formalism, this corresponds to choosing between the arc $i \to j$ or the arc in the opposite direction $j \to i$; both of these *disjunctive* arcs have weight $w(i,j) := \sigma$. We use $\mathcal{O}$ to denote a *selection* of disjunctive arcs, containing at most one arc for each pair of opposite disjunctive arcs, and we write $G(\mathcal{O})$ to denote the corresponding disjunctive graph. Figure 3.2 shows an example of a disjunctive graph for some small instance of (C'). This simple graph reformulation is widely used in the scheduling literature. For a little bit more background, we refer the reader to the discussion of the job shop scheduling problem in Appendix B.

Instead of setting binary variables γ_{ij} to determine the crossing order, we can now equivalently think about choosing the set of disjunctive arcs $\mathcal{O}$. It will be convenient later to allow partial selections $\mathcal{O}$, in the sense that there may be conflicts for which neither of both arcs is chosen. Such a partial selection essentially encodes a set of possible crossing orders, which may be thought of as a partial solution. In the extreme case of $\mathcal{O} = \varnothing$, we obtain the so-called *empty* disjunctive graph $G_0 := G(\varnothing)$, which can be thought of as encoding an unsolved problem instance. On the other extreme, whenever we choose precisely one disjunctive arcs for each conflict, we say that $\mathcal{O}$ is a *complete selection* and $G(\mathcal{O})$ is the corresponding *complete disjunctive graph*.

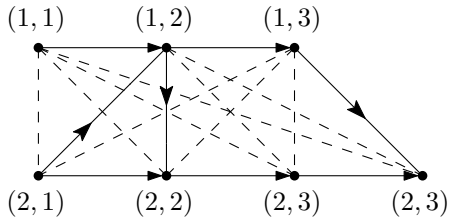


Figure 3.2: Illustration of disjunctive graph for some instance with $R = 2$ routes, and $N = 7$ vehicles in total, with $n_1 = 3$ on the first route and $n_2 = 4$ on the second route. Dummy nodes are not shown for clarity. The horizontal solid arrows represent the conjunctive arcs, the three other solid arrows represent some incomplete choice $\mathcal{O}$ of disjunctive arcs, so the remaining dashed lines represent pairs of disjunctive arcs that are not chosen.

Active schedules. Suppose we fix some selection of disjunctive arcs $\mathcal{O}$, not necessarily complete, and discard all disjunctive inequality constraints whose arcs are not in $\mathcal{O}$, then we obtain the following linear program

$$\begin{aligned} \min_{y} \quad & \sum_{i \in \mathcal{N}} y_i \\ \text{s.t.} \quad & a_i \leq y_i && \text{for all } i \in \mathcal{N}, \\ & y_i + \rho \leq y_j && \text{for all } (i,j) \in \mathcal{C}, \\ & y_i + \sigma \leq y_j && \text{for all } (i,j) \in \mathcal{O}. \end{aligned} \tag{AS}$$

We emphasize that when the selection $\mathcal{O}$ is complete, this linear program corresponds directly to an assignment of binary variables γ_{ij}, so that it is equivalent to (C'). Observe that the set of feasible solutions of this linear program can equivalently be characterized using the disjunctive graph as follows. Let $\mathcal{N}^-(j)$ denote the set of all *in-neighbors* of some node j in graph $G(\mathcal{O})$, which are all nodes $v \in \mathcal{N}$ such that there is some arc $v \to j$. Crossing time schedule y is a feasible solution to (AS) if and only if it satisfies

$$y_j \geq \max_{i \in \mathcal{N}^-(j)} y_i + w(i,j) \quad \text{for all } j \in \mathcal{N}. \tag{3.10}$$

When some feasible schedule y satisfies (3.10) with equality, it is called an *active schedule*. Whenever (AS) is feasible, it it clear from the objective that the optimal solution must be an active schedule.

Next, we investigate when a selection of disjunctive arcs guarantees feasibility. To see why feasibility is not guaranteed per se, consider some selection of disjunctive arcs $\mathcal{O}$ such that the disjunctive graph $G(\mathcal{O})$ contains some cycle

$$i_1 \to i_2 \to \cdots \to i_n \to i_1,$$

then it is easy to see that this corresponds to the chain of inequalities

$$\begin{aligned} y_{i_1} + w(i_1, i_2) &\leq y_{i_2}, \\ y_{i_2} + w(i_2, i_3) &\leq y_{i_3}, \\ &\vdots \\ y_{i_n} + w(i_n, i_1) &\leq y_{i_1}, \end{aligned}$$

which together imply that y_{i_1} must satisfy

$$y_{i_1} + \sum_{m=1}^{n-1} w(i_m, i_m + 1) + w(i_n, i_1) \leq y_{i_1},$$

which is an obvious contradiction when we assume that the processing and switching time are positive, such that all the weights in the above inequality are positive. This shows that

the absence of cycles is necessary for feasibility of (AS). Moreover, given the simple nature of the linear program, it is not surprising that it is also sufficient. Specifically, we show that when $G(\mathcal{O})$ is acyclic, there exists an active schedule, so (AS) is feasible. For brevity, we will say "$\mathcal{O}$ is acyclic" to mean "$G(\mathcal{O})$ is acyclic". We first recall the following elementary property of directed acyclic graphs, whose proof can be found in Appendix E.

Lemma 3.1. *Let $\mathcal{G}$ be some Directed Acyclic Graph (DAG) over nodes V, then there exists some $v \in V$ that has no incoming arcs, which is called* minimal*. Moreover, the nodes V can be arranged in a sequence $v_1, v_2, \ldots, v_{|V|}$ such that if $\mathcal{G}$ contains an arc $v_i \to v_j$ then $i < j$. Such a sequence is called a* topological order*.*

Lemma 3.2. *Let $\mathcal{O}$ be an acyclic selection of disjunctive arcs, so such that $G(\mathcal{O})$ is acyclic, then there exists a unique active schedule $y(\mathcal{O})$.*

Proof. Since $G(\mathcal{O})$ is acyclic, Lemma 3.1 gives us some topological order $v_1, v_2, \ldots, v_{2N}$. Observe that there are exactly N minimal nodes in $G(\mathcal{O})$, which are precisely the N dummy nodes, for which we can just set $y_{v_k} = 0$. Next, we visit the remaining nodes according to the topological order $v_{N+1}, \ldots, v_{2N}$ and for each visited node v, we set

$$y_v := \max_{u \in \mathcal{N}^-(v)} y_u + w(u, v). \tag{3.11}$$

For every such update, note that each in-neighbor $u \in \mathcal{N}^-(v)$ has been visited before, because the existence of the arc $u \to v$ implies that u appears before v in the topological order. Therefore, y_u has already been assigned a value so the right-hand side of (3.11) is well-defined. Hence, we obtain the unique schedule $y(\mathcal{O}) = \{y_i : i \in \mathcal{N}\}$ satisfying (3.10) with equality. □

We emphasize that the lemma above does not require $\mathcal{O}$ to be complete, which we will use in the next section. For now, suppose that $\mathcal{O}$ is complete, then the unique active schedule $y(\mathcal{O})$ is the unique optimal solution to linear program (AS). This means that we can use $\mathcal{O}$ to encode candidate solutions to the original crossing time problem (C). In other words, instead of using $y_i : i \in \mathcal{N}$ as the main decision variables, we could consider the equivalent problem of finding some acyclic complete selection of disjunctive arcs $\mathcal{O}$, for which the corresponding active schedule $y(\mathcal{O})$ following from (AS) is optimal. Therefore, we essentially reduced the infinite number of candidate schedules to a finite number of candidate active schedules. The discussion so far can be concisely summarized as:

Lemma 3.3. *If selection $\mathcal{O}$ is acyclic and complete, then active schedule $y(\mathcal{O})$ is obviously a solution to problem* (C)*. Conversely, if y is an optimal solution to* (C)*, then there exists an acyclic complete selection $\mathcal{O}$ such that $y = y(\mathcal{O})$. Hence, problem* (C) *is equivalent to*

$$\min_{\mathcal{O}} \sum_{i \in \mathcal{N}} y_i(\mathcal{O}) \quad \textit{s.t. } \mathcal{O} \textit{ is acylic and complete.} \tag{3.12}$$

Sequence representation. Although the disjunctive graph representation is a very helpful and general tool, it is less suitable for the sequence modeling techniques that we will introduce in the next section. A more natural representation of the crossing order is to just consider a permutation $\pi = (\pi_1, \ldots, \pi_N)$ of vehicle indices $\mathcal{N}$ that respects the initial ordering of vehicles on each route. By the latter, we mean that for any pair $(r, k), (r, k+m) \in \mathcal{N}$, for some integer $m \geq 1$, of vehicles on the same route, we have that (r, k) appears in π before $(r, k+m)$. We call such permutation π a *vehicle order*. Because the relative order of vehicles on the same route is fixed anyway, it is even simpler to only consider the order of routes from which vehicles cross the intersection. Given some vehicle order π, let the *route order* $\eta(\pi)$ be uniquely defined by $\eta_t(\pi_t) = r(\pi_t)$, for every $t \in \{1, \ldots, N\}$.

Definition 3.1. Let E denote the set of valid route orders, which are all sequences $\eta \in \mathcal{R}^N$ of route indices that contain precisely n_r occurences of symbol r for every route $r \in \mathcal{R}$.

Given some route order $\eta \in E$, it is easy to see that there must be a unique vehicle order π such that $\eta = \eta(\pi)$. Note that this vehicle order can be reconstructed using the following

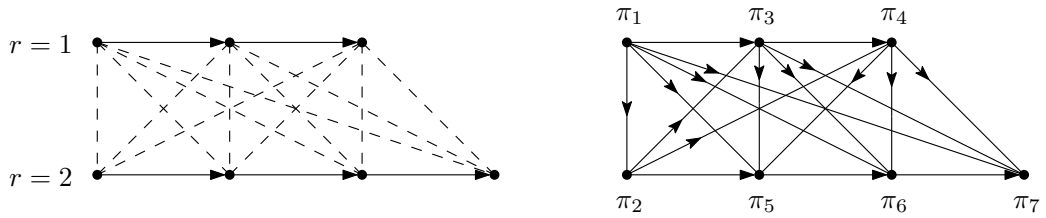


Figure 3.3: Illustration of disjunctive graphs for the same instance as in Figure 3.2, again without dummy nodes. Horizontal arrows are the conjunctive arcs, the rest are disjunctive arcs. The left graph illustrates the *empty* disjunctive graph; the dashed lines indicate an empty selection $\mathcal{O} = \varnothing$. The selection shown in the *acyclic* and *complete* graph on the right corresponds uniquely to the vehicle order $\pi = ((1,1),(2,1),(1,2),(1,3),(2,2),(2,3),(2,4))$ and the route order $\eta = (1,2,1,1,2,2,2)$.

simple procedure. Let $\mathbf{1}\{\cdot\}$ denote the indicator function and consider for each route r the following vehicle counter

$$k_t(r) := \sum_{i=1}^{t} \mathbf{1}\{\eta_i = r\},$$

then $\pi_t = (\eta_t, k_t(\eta_t))$ for every $t \in \{1,\dots,N\}$. This shows that there is a bijection between vehicle orders and route orders, so we can use them interchangeably. Next, we will now show further equivalence to the set of acyclic complete disjunctive graphs, as illustrated by Figure 3.3, which is the main result of this section.

Theorem 3.1. *If $\eta \in E$, then $y^\eta := y(\mathcal{O}(\eta))$ is a solution to* (C)*. Conversely, if y is an optimal solution to problem* (C)*, then there exists a route order η such that $y = y^\eta$. Hence, problem* (C) *is equivalent to the* route ordering problem

$$\min_{\eta \in E} \sum_{i \in \mathcal{N}} y_i^\eta. \tag{RO}$$

Proof. Let Ω denote thet set of all acyclic complete selections $\mathcal{O}$, and let P denote the set of vehicle orders, then we will show that there is a bijection between Ω and P. We already argued that there is a bijection between P and E, so from there, the desired result follows from Lemma 3.3.

Assume that $G(\mathcal{O})$ is an acyclic complete disjunctive graph, then subgraph induced by the non-dummy nodes $\mathcal{N}$ is a DAG, so there must be some topological order π of $\mathcal{N}$ by Lemma 3.1. It is clear that π must respect the ordering of vehicles on routes, due to the conjunctive arcs, so it is clearly a vehicle order.

To show that π is unique, suppose there is another topological order $\pi' \neq \pi$, then there must be at least one pair of indices $i, j \in \mathcal{N}$ that appears in opposite orders in π and π', say, i appears before j in π and j appears before i in π'. It follows that i and j cannot both belong to the same route, because otherwise one of the topological orders would be invalid. Therefore, $r(i) \neq r(j)$, but then we have either $(i,j) \in \mathcal{O}$ or $(j,i) \in \mathcal{O}$, which shows again that one of π or π' must be invalid. Hence, π is the only valid topological order.

Suppose we are given some vehicle order π, then it is straightforward to construct the corresponding disjunctive graph as follows. Starting with an empty selection, we visit the non-dummy nodes in order π and for each node π_k, we just add all disjunctive arcs $\pi_k \to v$ such that $r(v) \neq r(\pi_k)$ and $v \in \mathcal{N}$ has not yet been visited before. This way, we end up with a unique acyclic complete selection $\mathcal{O}(\pi)$. □

3.3 Sequential schedule construction

We will now return to our goal of applying the general methodological framework presented in Section 3.1 to the crossing time scheduling problem. As we mentioned there, the most important decision is the design of the state space, since this essentially determines the class of algorithms we are optimizing over, and which is the main way of incorporating prior knowledge and intuition into the model. We will now show how the findings of the previous section leads to a natural choice of the state space and thus the corresponding algorithmic structure.

The main takeaway of the previous section is that, instead of trying to find an optimal schedule of crossing times y_i directly, we can focus on trying to find the optimal crossing order, for which the crossing times follow immediately by solving the linear program (AS) or using the procedure in the proof of Lemma 3.2. From now on, we will mainly be working with η, because it is in some sense the simplest representation of a schedule and focus on solving (RO). We propose to study a class of *constructive algorithms*, which try to build the optimal schedule by scheduling a single vehicle at each step. Therefore, the intermediate states of the underlying MDP naturally encode *partial schedules*, which can be generally defined as acyclic selections $\mathcal{O}$ that are not yet complete. We essentially propose a method to construct $\mathcal{O}$ in a step-by-step fashion, by adding sets of disjunctive arcs at each step until it is complete.

Suppose that $\mathcal{O}$ is not yet complete and let $y(\mathcal{O})$ be the corresponding active schedule. Observe that adding more arcs to $\mathcal{O}$ can only cause the crossing times of the corresponding active schedule to increase with respect to $y(\mathcal{O})$, because additional constraints are added to (AS). Therefore, the active schedules for incomplete selections can also be interpreted as providing lower bounds on the crossing times for any *completion* of that selection, in the sense of the following corollary of Lemma 3.2.

Lemma 3.4. *Let $\mathcal{O} \subseteq \mathcal{O}^*$ be two acyclic selections of disjunctive arcs and let $y = y(\mathcal{O})$ and $y^* = y(\mathcal{O}^*)$ denote the corresponding active schedules, then $y_i^* \geq y_i$ for all $i \in \mathcal{N}$.*

This leads to the definition of *augmented disjunctive graphs*.

MDP definition. The states are all possible augmented disjunctive graphs.

Let $i \in \mathcal{I}$ denote some problem instance.

Let $\mathcal{G}_0$ denote the initial augmented graph.

The action space consists of the route indices $\mathcal{A} = \mathcal{R}$.

Note that not all actions are feasible for every state; we need to exclude route indices for routes where all vehicles have already been scheduled. Final states are all the possible complete disjunctive graphs and the corresponding final reward is just the negative sum of all crossing times.

$$m(i, \pi) = \sum_{i \in \mathcal{N}} y_i^{\eta} \tag{3.13}$$

The total number of vehicles N in the system is the fixed episode length.

3.3.1 Policy optimization

We will now explain how model parameters can be tuned using some sample of problem instances $X \sim \mathcal{X}$. Given some instance s, let $\eta_\tau(s)$ be the schedule produced by the threshold heuristic. Note that $L(s, \eta^\tau(s))$ is not differentiable with respect to τ, so we cannot use gradient-based optimization methods. However, we only have a single parameter, so we can simply select the value of τ that minimizes the average empirical loss

$$\min_{\tau \geq 0} \sum_{s \in X} L(s, \eta_\tau(s)),$$

which can be computed through a simple grid search. This approach is no longer possible when $p(\eta \,|s)$ is a proper distribution, so we need to something different for the neural parameterization.

Supervised learning. Consider some instance $s \in X$ and let η^* denote some optimal route sequence, which can for example be computed by solving the integer program (C'). For each such optimal schedule, we can compute the sequence $G_0, \eta_1, G_1, \eta_2, \ldots, \eta_N, G_N$. The resulting set of pairs $\{(G_t, \eta_{t+1}) : t = 1, \ldots, N-1\}$ can be used to learn p_θ in a supervised fashion by treating it as a classification task and computing the maximum likelihood estimator $\hat{\theta}$. Interpreting G_t as *states* and η_{t+1} as *actions*, we see that this approach is equivalent to *imitation learning* in the context of finding policies for Markov decision processes from so-called *expert demonstration*. Let Z denote the set of all state-action pairs collected from all training instances X. We make the procedure concrete for the case of two routes $\mathcal{R} = \{1, 2\}$, which is slightly simpler. Let $p_\theta(G_t)$ denote the probability of choosing the first route, then we can use the binary cross entropy loss, given by

$$L_\theta(Z) = -\frac{1}{|Z|} \sum_{(G_t, \eta_{t+1}) \in Z} \mathbb{1}\{\eta_{t+1} = 1\} \log(p_\theta(G_t)) + \mathbb{1}\{\eta_{t+1} = 2\} \log(1 - p_\theta(G_t)),$$

where $\mathbb{1}\{\cdot\}$ denotes the indicator function. Now we can simply rely on some gradient-descent optimization procedure to minimize $L_\theta(Z)$ with respect to θ.

Reinforcement learning. Instead of using state-action pairs as examples to fit the model in a supervised fashion (imitation learning), we can also choose to use the reinforcement learning paradigm, in which the data collection process is guided by some policy.

The reinforcement learning approach depends on the definition of a reward. For each step $G_{t-1} \xrightarrow{\eta_t} G_t$, we can define a corresponding reward, effectively yielding a deterministic Markov decision process. Specifically, we define the reward at step t to be

$$R_t = \sum_{i \in \mathcal{N}} \beta_{t-1}(i) - \beta_t(i).$$

Let the return at step t be defined as

$$\hat{R}_t = \sum_{k=t+1}^{N} R_k.$$

Hence, when the *episode* is done after N steps, the total episodic reward is given by the telescoping sum

$$\hat{R}_0 = \sum_{t=1}^{N} R_t = \sum_{i \in \mathcal{N}} \beta_0(i) - \beta_N(i) = \sum_{i \in \mathcal{N}} a_i - y_i = -L(s, \eta),$$

Therefore, maximizing the episodic reward corresponds to minimizing the scheduling objective, as desired.

Policy-based methods work with an explicit parameterization of the policy. The model parameters are then tuned based on experience, often using some form of (stochastic) gradient

descent to optimize the expected total return. Therefore, the gradient of the expected return plays a central role. The following identity is generally known as the Policy Gradient Theorem:

$$\begin{aligned}
\nabla \mathbb{E}_p \hat{R}_0 &\propto \sum_s \mu_p(s) \sum_a q_p(s,a) \nabla p(a|s,\theta) \\
&= \mathbb{E}_p \left[\sum_a q_p(S_t,a) \nabla p(a|S_t) \right] \\
&= \mathbb{E}_p \left[\sum_a p(a|S_t) q_p(S_t,a) \frac{\nabla p(a|S_t)}{p(a|S_t)} \right] \\
&= \mathbb{E}_p \left[q_p(S_t,A_t) \frac{\nabla p(A_t|S_t)}{p(A_t|S_t)} \right] \\
&= \mathbb{E}_p \left[\hat{R}_t \log \nabla p(A_t|S_t) \right].
\end{aligned}$$

The well-known REINFORCE estimator is a direct application of the Policy Gradient Theorem. At each step t, we update the parameters θ using a gradient ascent update

$$\theta \leftarrow \theta + \alpha \hat{R}_t \nabla \log p_\theta(\eta_t|G_t),$$

with some fixed learning rate α. To reduce variance of the estimator, we can incorporate a so-called *baseline*, which is an estimate of the expected return of the current state. In the context of combinatorial optimization, the value of the baseline may be interpreted as estimating the relative difficulty of an instance?

Beam search. For inference, we would ideally want to calculate the maximum likelihood estimator

$$\arg\max_\eta p(\eta|s),$$

but this is generally very expensive to compute, because this could require $O(R^N)$ evaluations of $p(\eta_t|s,\eta_{1:t-1})$ when we do not make additional structural model assumption. Therefore, one often uses *greedy rollout*, which means that we pick η_t with the highest probability at every step. Other inference strategies have been proposed in the context of modeling combinatorial optimization problems, see for example the "Sampling" and "Active Search" strategies in the seminal paper [5].

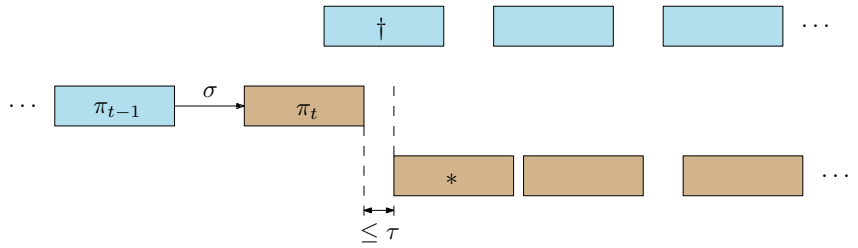


Figure 3.4: Illustration of how the threshold heuristic is evaluated at some intermediate step t to choose the next route η_{t+1}. The top and bottom row contains the unscheduled vehicles from route 1 and route 2, respectively, drawn at their earliest crossing times $\beta_t(i)$. The middel row represents the current partial schedule. Vehicle π_{t-1} is from route 1 and the last scheduled vehicle π_t is from route 2 and the disjunctive constraint for them happens to be tight in this case, illustrated by the arrow. Whenever the indicated distance is smaller than τ, the threshold rule selects vehicle $*$ to be scheduled next. Otherwise, vehicle † will be chosen.

3.3.2 Policy parameterizations

We can consider different ways of parameterizing $p(\eta_{t+1}|s, \eta_{1:t})$. Here, each partial sequence $\eta_{1:t}$ represents some partial schedule, which can equivalenty be defined in terms of the sequence of *scheduled* vehicles $\pi_{1:t}$. However, it is more convenient to define a partial schedule in terms of the *partial disjunctive graph* $G_t = (\mathcal{N}, \mathcal{C} \cup \mathcal{O}_t)$, which is defined by the unique selection $\mathcal{O}_t$ such that for each $i \in \pi_{1:t}$ and each $\{i, j\} \in \mathcal{D}$, we have either $(i, j) \in \mathcal{O}_t$ or $(j, i) \in \mathcal{O}_t$ with $j \in \pi_{1:t}$. To emphasize this parameterization in terms of the partial disjunctive graph, we can alternatively write (**??**) as

$$p(\eta|s) = \prod_{t=1}^{N} p(\eta_t|G_{t-1}).$$

There is a natural extensions of expression (**??**) for partial disjunctive graphs. Given G_t, let the *earliest crossing time* of each vehicle $i \in \mathcal{N}$ be recursively defined as

$$\beta_t(j) = \max\{a_j, \max_{i \in \mathcal{N}_t^-(j)} \beta_t(i) + w(i, j)\},$$

where $\mathcal{N}_t^-(j)$ denotes the set of in-neighbors of node j in G_t. For empty schedules, we have $\beta_0(i) = a_i$ for all i. For complete schedules, we have $\beta_N(i) = y^\eta(i)$ for all i. We have $\beta_t(i) \leq \beta_{t+1}(i)$ for all i, because $\mathcal{O}_t \subset \mathcal{O}_{t+1}$. Hence, β_t can be interpreted as providing the best lower bounds $\beta_t(i) \leq y^\eta(i)$, regardless of how the partial schedule is completed.

Threshold heuristic. We know from Theorem 2.2 that whenever it is possible to schedule a vehicle immediately after its predecessor on the same route, then this must be done in any optimal schedule. Based on this idea, we might think that the same holds true whenever a vehicle can be scheduled *sufficiently* soon after its predecessor. Although this is not true in general, we can define a simple heuristic based on this idea.

In this case, the model does not specify a proper distribution over η, but selects a single candidate by selecting a single next route in each step, so with $\mathbb{1}\{\cdot\}$ denoting the indicator function, we have $p(\eta_{t+1}|G_t) = \mathbb{1}\{\eta_{t+1} = p_\tau(G_t)\}$, selecting the next route at step t as

$$p_\tau(G_t) = \begin{cases} \eta_t & \text{if } \beta_t(\pi_t) + \rho + \tau \geq a_j \text{ and } (\pi_t, j) \in \mathcal{C}, \\ \texttt{next}(\eta_t) & \text{otherwise,} \end{cases}$$

where $\texttt{next}(\eta_t)$ denotes some arbitrary route other than η_t with unscheduled vehicles left. The threshold heuristic is illustrated in Figure 3.4.

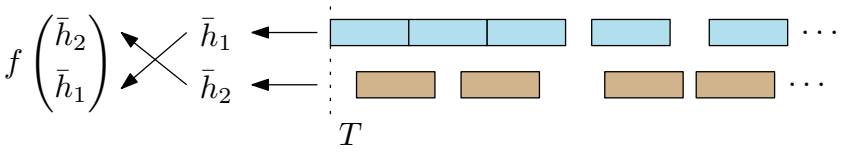


Figure 3.5: Schematic overview of parameterization of the neural heuristic. The distribution over the next route is parameterized as $p_\theta(\eta_{t+1}|G_t)$, which is computed from the current route horizons h_r via embeddings $\bar{h}_r$ and the cyling trick. Observe that the particular cycling shown in the figure corresponds to a situation in which the current route is $\eta_t = 2$.

Neural parameterization. We will now consider a parameterizaton that directly generalizes the threshold heuristic. Instead of looking at the earliest crossing time of the next vehicle in the current lane, we now consider the earliest crossing times of all unscheduled vehicles across lanes. In the following definitions, we drop the step index t to avoid cluttering the notation. For every route r, let π^r denote the sequence of unscheduled vehicles and consider their crossing time lower bounds $\beta(\pi^r) = (\beta(\pi^r_1), \beta(\pi^r_2), \dots)$. Let the minimum crossing time lower bound among unscheduled vehicles be denoted by T, then we call $h_r = \beta(\pi^r) - T = (\beta(\pi^r_1) - T, \beta(\pi^r) - T, \dots)$ the *horizon* of route r.

Next, we define some neural embedding $\bar{h}_r$ of each horizon. Observe that horizons can be variable length. We could fix the length by using padding, but this can be problematic for states that are almost done. Therefore, we employ a recurrent neural network.

The main idea of recurrent neural networks is to share parameters θ across different sequence steps. This way, we obtain a size-agnostic architecture. Given input x_n and internal *state* h_n, produce output o_n and next state h_{n+1}, by some function

$$(o_n, h_{n+1}) = f_\theta(x_n, h_n). \tag{3.14}$$

Each horizon h_r is simply transformed into a fixed-length embedding by feeding it in reverse order through a plain Elman RNN. Generally speaking, the most recent inputs tend to have greater influence on the output of an RNN, which is why we feed the horizon in reverse order, such that those vehicles that are due first are processed last, since we expect those should have the most influence on the decision. These horizon embeddings are arranged into a vector h_t by the following cycling rule. At position k of vector h_t, we put the embedding for route

$$k - \eta_t \bmod |\mathcal{R}|$$

where η_t denotes the last selected route. Using this cycling, we make sure that the embedding of the last selected route is always kept at the same position of the vector. Using some fully connected neural network f_θ and a softmax layer, this global embedding is then finally mapped to a probability distribution as

$$p_\theta(\eta_{t+1}|G_t) = \mathrm{softmax}(f_\theta(h_t)),$$

where θ denotes the parameters of f and of the recurrent neural networks. After θ has been determined, we can apply greedy rollout by simply ignoring routes that have no unscheduled vehicles left and take the argmax of the remaining probabilities.

3.4 Experimental results

The evalutation of model performance is roughly based on two aspects. Of course, the quality of the produced solutions is important. Second, we need to take into account the time that the algorithm requires to compute the solutions. We need to be careful here, because we have both training time as well as inference time for sequence models.

For each route $r \in \mathcal{R}$, we model the sequence of earliest crossing times $a_r = (a_{r1}, a_{r2}, \dots)$ as a stochastic process, to which we refer as the *arrival process*. Recall that constraints (C.2) ensure a safe following distance between successive vehicles on the same route. Therefore, we want the process to satisfy

$$a_{(r,k)} + \rho_{(r,k)} \leq a_{(r,k+1)},$$

for all $k = 1, 2, \dots$. Let the interarrival times be denoted as X_n with cumulative distribution function F and mean μ, assuming it exists. We define the arrival times $A_n = A_{n-1} + X_n + \rho$, for $n \geq 1$ with $A_0 = 0$.

Remark 3.4. *Note that the arrival process may be interpreted as an renewal process with interarrivals times $X_n + \rho$. Let N_t denote the corresponding counting process, i.e., N_t counts the cumulative number of arrivals up to time t, then by the renewal theorem, we obtain the limiting density of arrivals*

$$\mathbb{E}(N_{t+h}) - \mathbb{E}(N_t) \to \frac{h}{\mu + \rho} \quad \text{as } t \to \infty,$$

for $h > 0$. Hence, we refer to the quantity $\lambda := (\mu + \rho)^{-1}$ as the average arrival intensity.

Platoons. In order to model the natural occurrence of platoons, we model the interarrival times X_n as a mixtures of two random variables, one with a small expected value μ_s to model the gap between vehicles within the same platoon and one with a larger expected value μ_l to model the gap between vehicles of different platoons. For example, consider a mixture of two exponentials, such that

$$\begin{aligned} F(x) &= p(1 - e^{-x/\mu_s}) + (1-p)(1 - e^{-x/\mu_l}), \\ \mu &= p\mu_s + (1-p)\mu_l, \end{aligned}$$

assuming $\mu_s < \mu_l$. Observe that the parameter p determines the average length of platoons. Consider two intersecting routes, $\mathcal{R} = \{1, 2\}$, with arrival processes $a_1 = (a_{11}, a_{12}, \dots)$ and $a_2 = (a_{21}, a_{22}, \dots)$, with arrival intensities $\lambda^{(1)} = \lambda^{(2)}$. We keep $\lambda_s = 0.5$ constant, and use

$$\mu_l = \frac{\mu - p\mu_s}{1 - p}$$

to keep the arrival rate constant across arrival distributions.

We study the effect of the problem instance distribution $\mathcal{X}$ by varying the number of routes and number of arrivals per route, distribution of interarrival times, arrival intensity per route and degree of platooning.

Let $N(s)$ denotes the total number of vehicles in instance s. To enable a fair comparison across instances of various sizes, we report the quality of a solution in terms of the average delay per vehicle $L(\eta, s)/N(s)$. Given some problem instance s, let η^* denote the schedule computed using branch-and-bound. We use a fixed time limit of 60 seconds per instance for the branch-and-bound procedure, in order to bound the total analysis time. Therefore, it might be that η^* is not really optimal for some of the larger instances. Given some, possibly suboptimal, schedule η, we define its *optimality gap* as

$$L(s, \eta)/L(s, \eta^*) - 1.$$

For each heuristic, we report the average optimality gap over all test instances.

The performance of the threshold heuristic is evaluated based on optimal solutions obtained using MILP in Table 3.2. With the specific choice $\tau = 0$, the threshold rule is

Table 3.2: Performance evaluation of the branch-and-cut (MILP) approach and the threshold heuristic for different classes of instances with two routes. The first two columns specify the instance class based on the number of vehicles n per route and the type of arrival distribution for each route. These arrival distributions are chosen such that the arrival intensity is the same, only the degree of platooning varies. Performance is measured in terms of $L(s, \eta)/N(s)$, averaged over 100 test instances. The optimality gap is shown in parentheses for the heuristics. The threshold heuristic is fitted based on 100 training instances and the optimal threshold and training time is indicated. For branch-and-cut the average inference time is indicated. Note that we used a time limit of 60 seconds for all the branch-and-cut computations.

n	type	MILP	time	exhaustive (gap)	threshold (gap)	τ_{opt}	time
10	low	5.29	0.05	9.49 (79.45%)	7.78 (47.08%)	0.95	3.79
30	low	8.60	2.01	12.72 (47.88%)	11.31 (31.49%)	2.65	21.04
50	low	11.03	13.78	16.56 (50.20%)	14.60 (32.41%)	1.95	51.09
10	med	4.46	0.06	7.20 (61.32%)	6.39 (43.15%)	2.50	3.79
30	med	6.99	1.88	9.43 (34.97%)	8.96 (28.29%)	1.10	21.14
50	med	8.55	15.11	11.50 (34.40%)	10.77 (25.93%)	1.70	51.23
10	high	4.47	0.06	6.35 (42.04%)	5.70 (27.51%)	1.35	3.81
30	high	6.90	1.90	8.92 (29.19%)	8.58 (24.25%)	1.00	21.16
50	high	7.37	14.99	9.36 (26.94%)	8.88 (20.42%)	0.80	51.45

related to the so-called *exhaustive policy* for polling systems, which is why we consider this case separately. We plot the average objective for the values of τ in the grid search, see Figure **??**.

The neural heuristics is trained for a fixed number of training steps. At regular intervals, we compute the average validation loss and store the current model parameters. At the end of the training, we pick the model parameters with the smallest validation loss. The results are listed in Table 3.3. For the neural heuristic with supervised (imitation) learning, we plot the training and validation loss, see Figure **??**. It can be seen that the model converges very steadily in all cases. For the policy gradient method using REINFORCE with baseline, the training loss with episodic baseline is shown in Figure **??** and for the stepwise baseline in Figure **??**.

Table 3.3: Comparison of neural heuristics with supervised learning and reinforcement learning, based on average delay per vehicle for different classes of instances with two routes. The first two columns specify the instance class based on the number of vehicles n per route and the type of arrival distribution for each route. These arrival distributions are chosen such that the arrival intensity is the same, only the degree of platooning varies. The supervised heuristic is fitted based on 100 train instances and results averaged over 100 test instances. We use two different ways to compute the baseline: single baseline per episode, or baseline for every step during the episode.

n	type	supervised (gap)	time	episodic (gap)	time	stepwise (gap)	time
10	low	5.34 (0.92%)	6.13	5.73 (8.27%)	92.42	5.54 (4.81%)	125.94
30	low	8.70 (1.15%)	9.90	9.81 (14.05%)	290.47	9.17 (6.67%)	782.60
50	low	11.15 (1.15%)	13.99	12.38 (12.32%)	517.16	12.13 (9.99%)	2423.15
10	med	4.53 (1.44%)	5.66	4.85 (8.65%)	94.26	4.66 (4.42%)	128.56
30	med	7.10 (1.62%)	9.75	8.34 (19.42%)	296.29	7.77 (11.22%)	789.75
50	med	8.66 (1.26%)	13.99	10.33 (20.80%)	518.25	10.10 (18.09%)	2434.19
10	high	4.54 (1.50%)	5.97	4.81 (7.49%)	94.93	4.56 (2.01%)	129.00
30	high	7.05 (2.13%)	9.85	7.88 (14.15%)	295.97	8.01 (16.07%)	791.48
50	high	7.52 (1.98%)	14.01	8.91 (20.86%)	518.51	8.41 (14.13%)	2437.07

3.5 Notes and references

The idea of the disjunctive graph is due to [44], but the original technical note does not seem to be publicly accessible.

Lemma 3.5 is very much related to the general notion of an *active schedule* in the scheduling literature, see [43, Definition 2.3.3]. We mention a simpler way of calculating the active schedule $y(\mathcal{O})$ when the disjunctive graph $G(\mathcal{O})$ is complete and acyclic. It is based on a sequential construction, essentially following the path π, being the vehicle order, through the disjunctive graph $G(\mathcal{O})$ and computing y_i for each vehicle $i = \pi_t$ that we visit along the way (proof in Appendix E).

Lemma 3.5. *Let $\mathcal{O}$ be complete and such that $G(\mathcal{O})$ is acyclic, with unique vehicle order π as given by Lemma ??. Suppose that $\sigma > \rho > 0$, then active schedule $y(\mathcal{O}) = \{y_i : i \in \mathcal{N}\}$ is uniquely defined by $y_{\pi_1} = a_{\pi_1}$ and through the recursion*

$$y_j = \max\{a_j, y_i + w(i,j)\}, \tag{3.15}$$

for every pair $i = \pi_t$ and $j = \pi_{t+1}$ with $t = 1, \ldots, N-1$.

3.5.1 Local search

Without relying on systematic search methods like branch-and-bound, an often employed method is to use some kind of local search heuristic. The main idea is that the solution space can be organized based on some measure of similarity. From the current solution, we only move to a neighboring solution if it has a better objective value. Next, give an example of such a neighborhood.

As seen in the previous sections, vehicles of the same route occur mostly in platoons. For example, consider for example the route order $\eta = (0, 1, 1, 0, 0, 1, 1, 1, 0, 0)$. This example has 5 platoons of consecutive vehicles from the same route. The second platoon consists of two vehicles from route 1. The basic idea is to make little changes in these platoons by moving vehicles at the start and end of a platoon to the previous and next platoon of the same route. More precisely, we define the following two types of modifications to a route order. A *right-shift* modification of platoon i moves the last vehicle of this platoon to the next platoon of this route. Similarly, a *left-shift* modification of platoon i moves the first vehicle of this platoon to the previous platoon of this route. We construct the neighborhood of a solution by performing every possible right-shift and left-shift with respect to every platoon in the route order. For illustration purposes, we have listed a full neighborhood for some example route order in Table 3.4.

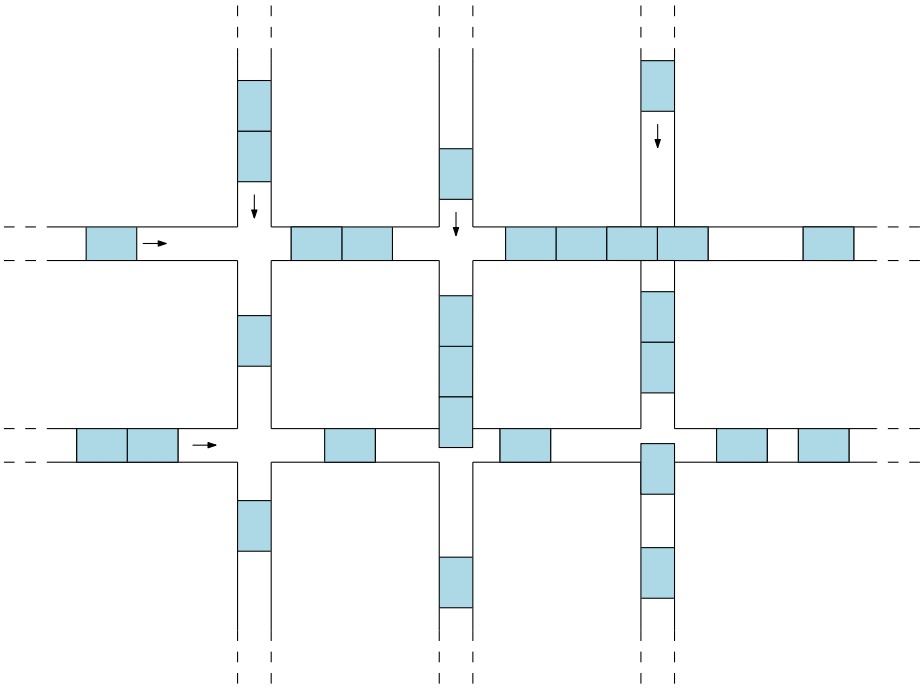
Now using this definition of a neighborhood, we must specify how the search procedure visits these candidates. In each of the following variants, the value of each neighbor is always computed. The most straightforward way is to select the single best candidate in the neighborhood and then continue with this as the current solution and compute its neighborhood. This procedure can be repeated for some fixed number of times. Alternatively, we can select the k best neighboring candidates and then compute the combined neighborhood for all of them. Then in the next step, we again select the k best candidates in this combined neighborhood and repeat. The latter variant is generally known as *beam search*.

Table 3.4: Local search neighborhood of route order $\eta = (0, 1, 1, 0, 0, 1, 1, 1, 0, 0)$ based on the left-shift and right-shift operations applied to every "platoon" in the current order.

platoon id	left-shift	right-shift
1		(1, 1, 0, 0, 0, 1, 1, 1, 0, 0)
2	(1, 0, 1, 0, 0, 1, 1, 1, 0, 0)	(0, 1, 0, 0, 1, 1, 1, 1, 0, 0)
3	(0, 0, 1, 1, 0, 1, 1, 1, 0, 0)	(0, 1, 1, 0, 1, 1, 1, 0, 0, 0)
4	(0, 1, 1, 1, 0, 0, 1, 1, 0, 0)	(0, 1, 1, 0, 0, 1, 1, 0, 0, 1)
5	(0, 1, 1, 0, 0, 0, 1, 1, 1, 0)	

Part II

Networks of Intersections



Chapter 4

Learning to schedule in networks

Due to the second-order boundary conditions $\dot{x}_i(a_i) = \dot{x}_i(b_i) = 1$, the feasibility of the lane planning problem is completely characterized in terms of its schedule times. We will show that it is precisely this property that us to construct a network consisting of individual lanes, connected at intersections. The fact that intersections are shared among multiple lanes naturally gives rise to some collision-avoidance constraints. We will see that the feasibility of finding collision-free trajectories can be stated completely in terms of schedule times, which essentially means that we do not need to worry about vehicle dynamics at all.

Network topology. We will use the lane model to build a simple network model. The network model is based on a directed graph $(\bar{V}, E)$ with nodes $\bar{V}$ and arcs E, which we will use to encode the possible routes. Nodes with no incoming arcs are *entrypoints* and nodes with no outgoing arcs are *exitpoints*. We use V to denote the set of *intersections*, which are nodes with in-degree at least two. Let $\mathcal{R}$ denote the index set for routes, then each $r \in \mathcal{R}$ corresponds to the route

$$\bar{V}_r = (v_r(0), v_r(1), \dots, v_r(m_r), v_r(m_r + 1)),$$

where we require $v_r(0)$ to be an entrypoint and $v_r(m_r + 1)$ to be an exitpoint. Furthermore, we use $V_r = \bar{V}_r \setminus \{v_r(0), v_r(m_r + 1)\}$ to denote the intersections on this route. Let $E_r \subset E$ denote the set of edges that make up V_r. We require that routes are *edge-disjoint*, which is made precise in the following assumption.

Assumption 4.1. *For every distinct routes $p, q \in \mathcal{R}$ such that $p \neq q$, we assume $E_p \neq E_q$.*

This assumption ensures that each route $\bar{V}_r$ can be modeled by connecting a sequence of lanes together, with some *intersection areas* of some fixed size W in between them, see Figure 4.1. Hence, we set the longitudinal start and end position of each lane model as follows. Let $d(v, w)$ denote the length of edge $(v, w) \in E_r$, then we recursively define

$$A_{r1} = 0, \tag{4.1a}$$
$$A_{rk} = B_{r,k-1} + W + L, \tag{4.1b}$$
$$B_{rk} = A_{rk} + d(v_r(k-1), v_r(k)), \tag{4.1c}$$

for each $k \in \{1, \dots, m_r + 1\}$.

Network scheduling. Introduce the global trajectory planning problem by defining the collision-avoidance constraints. Mention that this problem can be solved at once, by using direct transcription. Show that the bilevel formulation decomposes into a (combinatorial) scheduling problem.

Assumption 4.1 ensures that the order of vehicles on each lane is completely determined by the order of vehicles on the corresponding lane.

Instead of schedule times a_i and b_i, we are now going to use crossing times y_i.

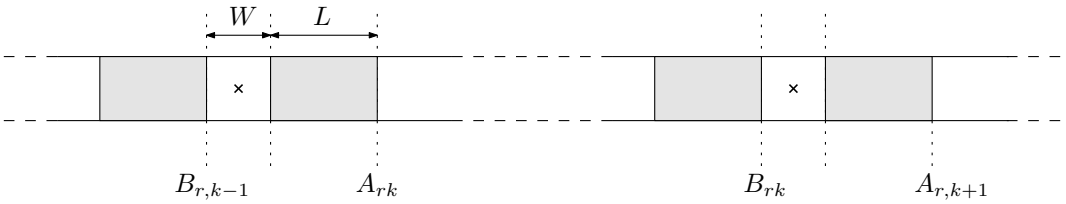


Figure 4.1: Illustration of how the individual lane models are connected to form a route with intersections, marked with a little cross. The four shaded rectangles illustrate four possible vehicle positions. The length of the intersection is W. The longitudinal positions A_{rk} and B_{rk} denote the start and end, respectively, of the kth lane on route r.

Show examples of empty and full disjunctive graphs, accompanied by Gantt charts. Mention that the resulting scheduling problem can be interpreted as an extension of the classical job-shop scheduling problem. Readers unfamiliar with the job shop scheduling problem may appreciate the brief introduction in Appendix B.

Occupancy time slot scheduling

4.1 Notes and references

We emphasize again that the assumption of crossing the intersection at maximum speed is a central feature of our model, because it causes intersections to act as a sort of "checkpoints". The fact that the global trajectory optimization can be stated as a jop-shop-like problem hinges on this assumption, because we essentially "localized" the effects of the vehicle dynamics to individual lanes; the coupling between lanes are through the crossing times. In order to relax this assumption and deal with objectives that takes into account energy or fuel consumption, we need to take make a global trade-off between fast crossing and energy efficiency, for which more advanced optimization schemes are required.

Although our network examples are all grid-like and have perpendicular intersecting lanes, the model proposed in this chapter is more generally applicable. For example, curved roads can be modeled as well, as long we assume that this does not affect the dynamics of the vehicles, in particular the maximum speed.

Chapter 5

Capacitated lanes

There is a difference in how we state the initial conditions of the system. Recall that, in Chapter 2, we defined initial position x_i^0 and velocity v_i^i for each vehicle i. This is also what is usually done when presenting an optimal control problem. Instead, we now consider the time of entry into the system. These two notions are not necessarily equivalent. Furthermore, we need to take special care in handling the domain of the resulting trajectories, since they are now different for each vehicle.

5.1 Model formulation

In our presentation of the isolated intersection model, we relied heavily on the assumption that the intial distance to the intersection was large enough for each vehicle to allow feasible trajectories. Once we extend our model to the multi-intersection situation, it does no longer make sense to ignore the finite length of roads between intersections. In other words, we need to start taking into account the *finite capacity* for vehicles of each lane between intersections. In order to formulate a model for a network of intersections, we first formulate and analyze a model for such a *capacitated lane*. In the next chapter, we will link such lanes together to construct a model representing a network of intersections.

We will propose a model for a one-directional single-lane road of finite length where overtaking is not permitted. The fact that such a lane can be occupied by a limited number of vehicles at the same time, makes the characterization of set of feasible trajectories more involved. Given some lane, consider the set of vehicles that need to travel across this lane as part of their planned route. Suppose that the time of entry to and exit from this lane are fixed for each of these vehicles, then the question is whether there exists a set of trajectories that is safe, i.e., without collisions, and which satisfies these *schedule times*. Loosely speaking, we ask whether there exists an easy way to answer this feasibility question for any set of schedule times. By requiring vehicles to enter and exit the lane at *full speed*, we will show that the answer positive, since the feasibility question is precisely answered by a system of linear inequalities in terms of the schedule times.

Of course, there is generally not a single feasible set of trajectories, so we can consider some different performance critera like we did for the isolated intersection. As we showed previously, the resulting optimal control problem is straightforward to solve using a direct transcription method. Recall the *haste objective*, which seeks to minimize each vehicle's distance to the end of the lane at all times. For this objective, we will show that the optimal solution can be computed much more efficiently. The derivation of this algorithm is a direct byproduct of the feasibility analysis, because the latter will involve the construction of a certain set of upper bounding trajectories, which happen to be optimal under the haste objective.

In the remainder of this introductory section, we will precisely establish the notion of a feasible set of trajectories in a capacitated lane. To provide some initial intuition, we show some examples of feasible trajectories for the haste objective.

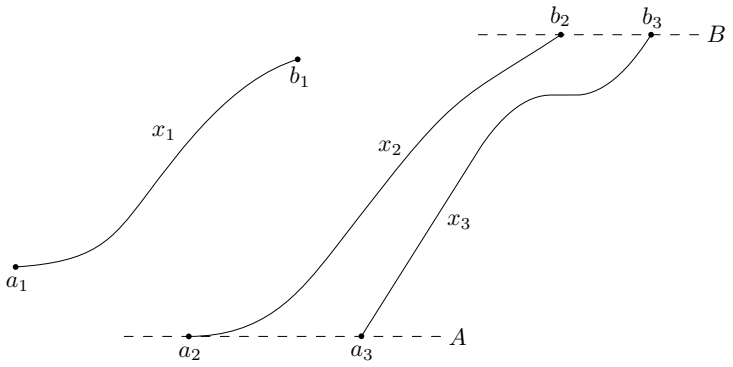


Figure 5.1: Example trajectories $x_1 \in \mathcal{D}[a_1, b_1]$, $x_2 \in D_1[a_2, b_2]$ and $x_3 \in D_2[a_3, b_3]$ for each the three classes of trajectories that are used throughout this chapter (horizontal axis is time, vertical axis is position).

Vehicle trajectories. We only consider the longitudinal position of vehicles on the lane and we assume that speed and acceleration are bounded. Therefore, let $\mathcal{D}[a, b]$ denote the set of valid *trajectories*, which we define to be all continuously differentiable functions $x : [a, b] \to \mathbb{R}$ satisfying the constraints

$$\dot{x}(t) \in [0, 1] \quad \text{and} \quad \ddot{x}(t) \in [-\omega, \bar{\omega}], \quad \text{for all } t \in [a, b], \tag{5.1}$$

for some fixed acceleration bounds $\omega, \bar{\omega} > 0$ and with $\dot{x}$ and $\ddot{x}$ denoting the first and second derivative with respect to time t. Note that the unit speed upper bound is not restrictive, since we can always apply an appropriate scaling of time and the acceleration bounds to arrive at this form. We use A and B to denote the start[1] and end position of the lane. Let $D_1[a, b] \subset \mathcal{D}[a, b]$ denote all trajectories x that satisfy the first-order boundary conditions

$$x(a) = A \;\text{ and }\; x(b) = B \tag{5.2}$$

and additionally satisfy $\dot{x}(a) > 0$ and $\dot{x}(b) > 0$, to avoid the technical difficulties of dealing with vehicles that are waiting at the start or end of the lane. On top of these conditions, let $D_2[a, b] \subset D_1[a, b]$ further induce the second-order boundary conditions

$$\dot{x}(a) = \dot{x}(b) = 1. \tag{5.3}$$

In words, these boundary conditions require that a vehicle arrives to and departs from the lane at predetermined times a and b and do so at full speed. Figure 5.1 shows an example for each of these three classes of trajectories.

Trajectory domains. Function domains will play an important role in the analysis of feasible trajectories. Therefore, we introduce some useful notational conventions. First of all, each of the trajectory classes above can be used with the common convention of allowing $a = -\infty$ or $b = \infty$. For instance, we write $\mathcal{D}(-\infty, \infty)$ to denote the set of trajectories defined on the whole real line. Furthermore, we use $\cdot|_{[a,b]}$ to denote function restriction. For example,

$$(t \mapsto t + 1)|_{[\xi,\infty)}$$

denotes some anonymous function with some restricted domain. Furthermore, given two smooth trajectories $\gamma_1 \in \mathcal{D}[a_1, b_1]$ and $\gamma_2 \in \mathcal{D}[a_2, b_2]$, we write inequality $\gamma_1 \preceq \gamma_2$ to mean

$$\gamma_1(t) \leq \gamma_2(t) \;\text{ for all }\; t \in [a_1, b_1] \cap [a_2, b_2].$$

Whenever the intersection of domains is empty, we say that the above inequality is *void*. The reason for introducing a dedicated symbol is that $\preceq$ is not transitive. To see this, consider the trajectories in Figure 5.1, then $x_1 \preceq x_3$ (void) and $x_3 \preceq x_2$, but clearly $x_1 \not\preceq x_2$.

[1]Note that assuming $A \neq 0$ is convenient later when we start piecing together multiple lanes.

Definition 5.1. Let $L > 0$ denote the *following distance* between consecutive vehicles. Suppose there are N vehicles scheduled to traverse the lane. For each vehicle i, let a_i and b_i denote the *schedule time* for entry and exit, respectively. Assuming that the schedule times are ordered as $a_1 \leq a_2 \leq \cdots \leq a_N$ and $b_1 \leq b_2 \leq \cdots \leq b_N$, then a *feasible solution* consists of a sequence of trajectories $x_1, \ldots, x_N$ such that

$$x_i \in D_2[a_i, b_i] \qquad \text{for each } i \in \{1, \ldots, N\}, \tag{5.4a}$$
$$x_i \preceq x_{i-1} - L \qquad \text{for each } i \in \{2, \ldots, N\}. \tag{5.4b}$$

We will refer to (5.4b) as the *lead vehicle constraints*. For some performance criterion of trajectories, given as a functional $J(x)$ of trajectory x, the *lane planning problem* is to find a feasible solution that maximizes

$$\min \sum_{i=1}^{N} J(x_i). \tag{5.5}$$

We emphasize again that (5.4a) requires vehicles to enter and exit the lane at full speed. The feasibility characterization that we will derive can now be roughly stated as follows. Assuming the system parameters $(\omega, \bar{\omega}, A, B, L)$ to be fixed, with lane length $B - A$ sufficiently large and following distance L sufficiently small, feasibility of the lane planning problem is characterized by a system of linear inequalities in terms of the schedule times a_i and b_i.

Choice of objective. Recall the *haste objective*, which was defined as $J_{\alpha,\beta}$ with $\alpha = -1$ and $\beta = 0$, which we will from now on denote as simply

$$J(x_i) = \int_{a_i}^{b_i} -x_i(t)\,\mathrm{d}t. \tag{5.6}$$

Roughly speaking, this objective seeks to keep all vehicles as close to the end of the lane at all times, but it does not capture energy efficiency in any way. In Section 5.4, we will show that optimal trajectories under the haste objective can be understood as the concatenation of at most four different types of trajectory parts, which we might call *bang-off-bang*. Based on this observation, we present an algorithm to compute optimal trajectories. Generalizing this algorithm to other objectives like $J_{\alpha,\beta}$ with arbitrary parameters α and β, is an interesting topic for further research.

5.2 Single vehicle with arbitrary lead vehicle constraint

Before we analyze the feasibility of the lane planning problem as a whole, we focus on the lead vehicle constraint (5.4b) for a single vehicle $i \geq 2$. This allows us to lighten the notation slightly by dropping the vehicle index i. Instead of $x_{i-1} - L$, we assume we are given some arbitrary *lead vehicle boundary* u and consider the following problem.

Definition 5.2. Let $u \in D_1[c, d]$ and assume we are given two schedule times $a, b \in \mathbb{R}$, then the *single vehicle (feasibility) problem* is to find a trajectory $x \in D_2[a, b]$ such that $x \preceq u$.

5.2.1 Necessary conditions

Suppose we are given some feasible trajectory x for the single vehicle problem. In addition to the given upper bounding trajectory u, we will derive two upper bounding trajectories x^1 and $\hat{x}$ and one lower bounding trajectory $\check{x}$, see Figure 5.2. Using these bounding trajectories, we will formulate four necessary conditions for the single vehicle problem.

Let the *full speed boundary*, denoted x^1, be defined as

$$x^1(t) = A + t - a, \tag{5.7}$$

for all $t \in [a, b]$, then we clearly have $x \preceq x^1$. Observe that $x^1(s) = B$ for $s = a + (B - A)$, which can be interpreted as the earliest time of departure from the lane, so we must have $b \geq a + (B - A)$. This is our first necessary condition.

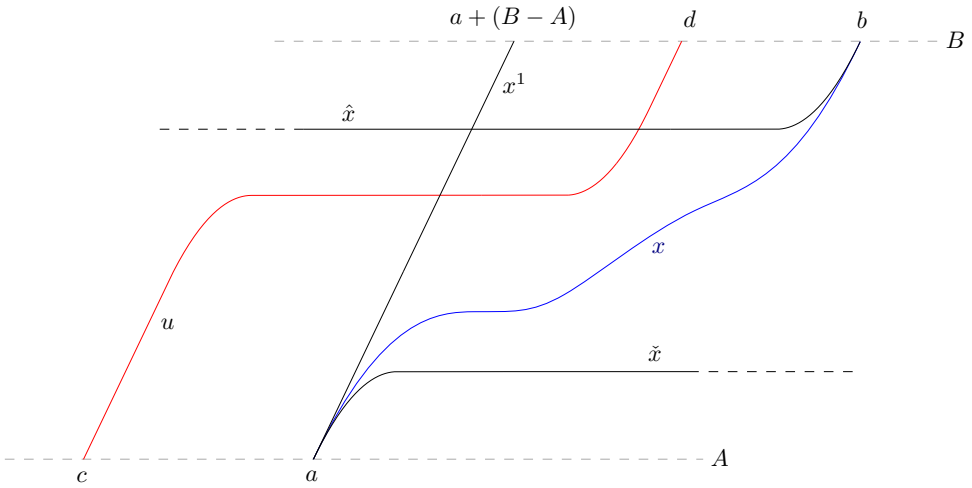


Figure 5.2: Illustration of the four bounding trajectories $u, x^1, \hat{x}, \check{x}$ that bound feasible trajectories from above and below. We also drew an example of a feasible trajectory x in blue. The horizontal axis represents time and the vertical axis corresponds to the position on the lane, so the vertical dashed grey lines correspond to the start and end of the lane.

Lemma 5.1. *If there exists $x \in D_2[a, b]$, then $b - a \geq B - A$.*

Next, since deceleration is at most ω, we have $\dot{x}(t) \geq \dot{x}(a) - \omega(t - a) = 1 - \omega(t - a)$, which we combine with the speed constraint $\dot{x} \geq 0$ to derive $\dot{x}(t) \geq \max\{0, 1 - \omega(t - a)\}$. Hence, we obtain the lower bound

$$x(t) = x(a) + \int_a^t \dot{x}(\tau)\,\mathrm{d}\tau \geq A + \int_a^t \max\{0, 1 - \omega(\tau - a)\}\,\mathrm{d}\tau =: \check{x}(t), \tag{5.8}$$

for all $t \geq a$, so that we have $x \succeq \check{x}$. Analogously, we derive an upper bound from the fact that acceleration is at most $\bar{\omega}$. Observe that we have $\dot{x}(t) + \bar{\omega}(b - t) \geq \dot{x}(b) = 1$, which we combine with the speed constraint $\dot{x}(t) \geq 0$ to derive $\dot{x}(t) \geq \max\{0, 1 - \bar{\omega}(b - t)\}$. Hence, we obtain the upper bound

$$x(t) = x(b) - \int_t^b \dot{x}(\tau)\,\mathrm{d}\tau \leq B - \int_t^b \max\{0, 1 - \bar{\omega}(b - \tau)\}\,\mathrm{d}\tau =: \hat{x}(t), \tag{5.9}$$

for all $t \leq b$, so we have $x \preceq \hat{x}$. We refer to $\check{x}$ and $\hat{x}$ as the *entry boundary* and *exit boundary*, respectively.

Lemma 5.2. *Consider some lead boundary $u \in D_1[c, d]$ and assume $[a, b] \cap [c, d] \neq \varnothing$. If there exists a trajectory $x \in D_2[a, b]$ such that $x \preceq u$, then $a \geq c$ and $b \geq d$ and $u \succeq \check{x}$.*

Proof. Each of these conditions corresponds somehow to one of the bounding trajectories defined above. Suppose $a < c$, then because the domains intersect, we must have $b > c$, but then clearly no x can satisfy $x \preceq u$. When $b < d$, then it is a consequence of $\dot{u}(b) > 0$ that any x will violate $x \preceq u$. To see that the third condition must hold, suppose that $u(\tau) < \check{x}(\tau)$ for some time τ. Since $c \leq a$, this means that u must intersect $\check{a}$ from above. Therefore, any trajectory that satisfies $x \preceq u$ must also intersect $\check{a}$ from above, which contradicts the assumption $x \in D_2[a, b]$. □

Remark 5.1. *The assumption of non-empty domains is required in the previous lemma, because otherwise we include the situation in which x lies completely to the left of u, in which case the stated conditions are obviously not necessary anymore.*

We note that the boundaries $\hat{x}$ and $\check{x}$ can be combined to yield yet another necessary condition. It is straightforward to verify from equations (5.8) and (5.9) that $\hat{x}(t) \geq B - 1/(2\bar{\omega})$ and $\check{x}(t) \leq A + 1/(2\omega)$. Therefore, whenever $B - A < 1/(2\bar{\omega}) + 1/(2\omega)$, these boundaries

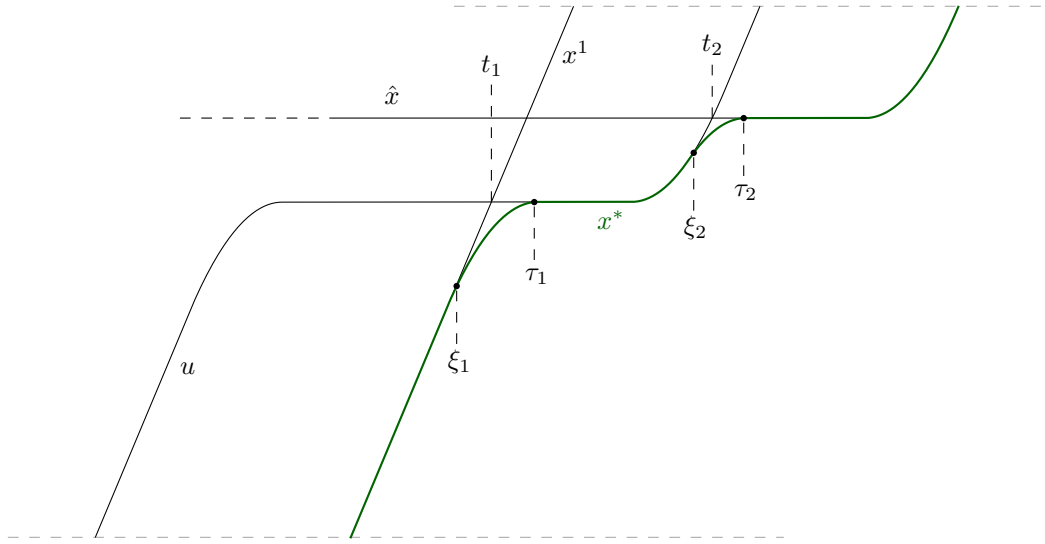


Figure 5.3: The minimum boundary γ, induced by three upper boundaries u, $\hat{x}$ and x^1, is smoothened around time t_1 and t_2, where the derivative is discontinuous, to obtain the smooth optimal trajectory x^*, drawn in green. The times ξ_i and τ_i correspond to the start and end of the connecting deceleration as defined in Section 5.2.4.

intersect for certain values of a and b. Because the exact condition is somewhat cumbersome to characterize, we avoid this case by simply assuming that the lane length is sufficiently large, to keep the analysis simpler.

Assumption 5.1. *The length of the lane satisfies* $B - A \geq 1/(2\omega) + 1/(2\bar{\omega})$.

Observe that $1/(2\omega)$ is precisely the distance required to decelerate from full speed to a standstill. Similarly, $1/(2\bar{\omega})$ is the distance required for a full acceleration. Therefore, we may interpret Assumption 5.1 as requiring enough space in the lane such that there is at least one *waiting position*. We will return to this observation in Section 5.3. (check that we do this)

5.2.2 Sufficient conditions

The goal of the remainder of this section is to prove the following feasibility characterization.

Theorem 5.1 (Feasibility characterization of single vehicle problem)**.** *Given some lead vehicle boundary* $u \in D_1[c, d]$ *and some schedule times* $a, b \in \mathbb{R}$ *such that* $[a, b] \cap [c, d] \neq \varnothing$ *and assuming Assumption 5.1, there exists a solution* $x \in D_2[a, b]$ *satisfying* $x \preceq u$ *if and only if*

(i)	$b - a \geq B - A$,	*(travel constraint)*
(ii)	$a \geq c$,	*(entry order constraint)*
(iii)	$b \geq d$,	*(exit order constraint)*
(iv)	$u \succeq \check{x}$.	*(entry space constraint)*

Note that Lemma 5.1 and Lemma 5.2 already showed necessity of these conditions. Therefore, we will show that, under these conditions, we can always construct a solution γ^* for the single vehicle problem, thereby showing that the four conditions are also sufficient. The particular solution that we will construct also happens to be a smooth upper boundary for all other solutions, in the sense that, for any other feasible solution x we have $x \preceq \gamma^*$. The starting point of the construction is the *minimum boundary* $\gamma : [a, b] \to \mathbb{R}$, defined as

$$\gamma(t) := \min\{u(t), \hat{x}(t), x^1(t)\}. \tag{5.10}$$

Obviously, γ is a valid upper boundary for any other feasible solution, but in general, γ may have a discontinuous derivative at some[2] isolated points in time, in which case $\gamma \notin \mathcal{D}[a,b]$.

Definition 5.3. Let $\mathcal{P}[a,b]$ be the set of functions $\mu : [a,b] \to \mathbb{R}$ for which there is a finite subdivision $a = t_0 < \cdots < t_{n+1} = b$ such that the truncation $\mu|_{[t_i,t_{i+1}]} \in \mathcal{D}[t_i,t_{i+1}]$ is a smooth trajectory, for each $i \in \{0,\ldots,n\}$, and for which the one-sided limits of $\dot{\mu}$ satisfy

$$\dot{\mu}(t_i^-) := \lim_{t \uparrow t_i} \dot{\mu}(t) > \lim_{t \downarrow t_i} \dot{\mu}(t) =: \dot{\mu}(t_i^+), \tag{5.11}$$

for each $i \in \{1,\ldots,n\}$. We refer to such μ as a *piecewise trajectory (with downward bends)*.

Under the conditions of Theorem 5.1, it is not difficult to see from Figure 5.2 that γ satisfies the above definition, so $\gamma \in \mathcal{P}[a,b]$. In other words, γ consists of a number of pieces that are smooth and satisfy the vehicle dynamics, with possibly some sharp bend downwards where these pieces come together. Next, we present a simple procedure to smoothen out this kind of discontinuity by decelerating from the original trajectory somewhat before some t_i, as illustrated in Figure 5.3. We will argue that this procedure can be repeated as many times as necessary to smoothen out every discontinuity.

In Section 5.2.3, we first define a parameterized family of functions to model the deceleration part that we introduce for the smoothing procedure, which is described in Section 5.2.4. We apply this procedure to γ to obtain γ^*, after which it is relatively straightforward to show that γ^* is an upper bound for all other feasible solutions, which is done in Section 5.2.5.

5.2.3 Deceleration boundary

Recall the derivation of $\check{x}$ in equation (5.8) and the discussion preceding it, which we will now generalize a bit. Let $x \in \mathcal{D}[a,b]$ be some smooth trajectory, then observe that $\dot{x}(t) \geq \dot{x}(\xi) - \omega(t-\xi)$ for all $t \in [a,b]$. Combining this with the constraint $\dot{x}(t) \in [0,1]$, this yields

$$\dot{x}(t) \geq \max\{0,\, \min\{1,\, \dot{x}(\xi) - \omega(t-\xi)\}\} =: \{\dot{x}(\xi) - \omega(t-\xi)\}_{[0,1]}, \tag{5.12}$$

where use $\{\cdot\}_{[0,1]}$ as a shorthand for this clipping operation. Hence, for any $t \in [a,b]$, we obtain the following lower bound

$$x(t) = x(\xi) + \int_\xi^t \dot{x}(\tau)\,\mathrm{d}\tau \geq x(\xi) + \int_\xi^t \{\dot{x}(\xi) - \omega(\tau-\xi)\}_{[0,1]}\,\mathrm{d}\tau =: x[\xi](t), \tag{5.13}$$

where we will refer to the right-hand side as the *deceleration boundary* of x at ξ. Observe that this definition indeed generalizes the definition of $\check{x}$, because we have $\check{x} = (x[a])|_{[a,b]}$.

Note that $x[\xi]$ depends on x only through the two real numbers $x(\xi)$ and $\dot{x}(\xi)$. It will be convenient later to rewrite the right-hand side of (5.13) as

$$x^-[p,v,\xi](t) := p + \int_\xi^t \{v - \omega(\tau-\xi)\}_{[0,1]}\,\mathrm{d}\tau, \tag{5.14}$$

such that $x[\xi](t) = x^-[x(\xi),\dot{x}(\xi),\xi](t)$. We can expand the integral in this expression further by carefully handling the clipping operation. Observe that the expression within the clipping operation reaches the bounds 1 and 0 for $\delta_1 := \xi - (1-v)/\omega$ and $\delta_0 := \xi + v/\omega$, respectively. Using this notation, a straightforward calculation shows that

$$x^-[p,v,\xi](t) = p + \begin{cases} (1-v)^2/(2\omega) + (t-\xi) & \text{for } t \leq \delta_1, \\ v(t-\xi) - \omega(t-\xi)^2/2 & \text{for } t \in [\delta_1,\delta_0], \\ v^2/(2\omega) & \text{for } t \geq \delta_0. \end{cases} \tag{5.15}$$

It is easily verified that the three cases above coincide at $t \in \{\delta_1,\delta_0\}$, which justifies the overlaps in the case distinction. Furthermore, since x and $\dot{x}$ are continuous by assumption, it follows that $x[\xi](t) = x^-[x(\xi),\dot{x}(\xi),\xi](t)$ is continuous as a function of either of its arguments.[3] Assuming $0 \leq v \leq 1$, it can be verified that for every $t \in \mathbb{R}$, we have $\ddot{x}^-[p,v,\xi](t) \in \{-\omega,0\}$ and $\dot{x}^-[p,v,\xi](t) \in [0,1]$ due to the clipping operation, so that $x^-[p,v,\xi] \in \mathcal{D}(-\infty,\infty)$.

[2] In fact, it can be shown that, under the necessary conditions, there are at most two of such discontinuities.

[3] Even more, it can be shown that $x[\xi](t)$ is continuous as a function of (ξ,t).

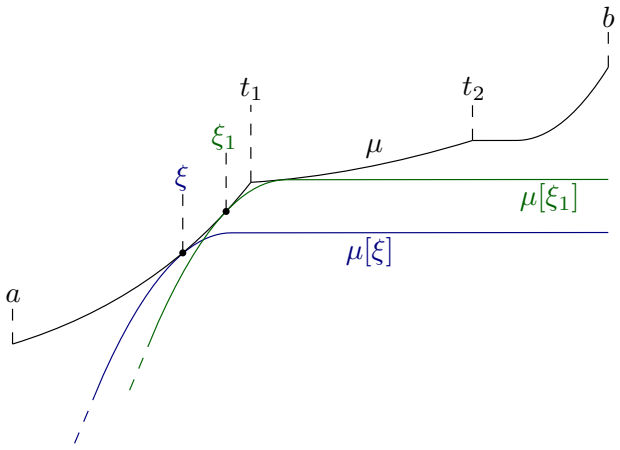


Figure 5.4: Illustration of some piecewise trajectory $\mu \in \mathcal{P}[a,b]$ with a discontinuous derivative at times t_1 and t_2. Furthermore, the figure shows some arbitrary deceleration boundary $\mu[\xi]$ at time ξ in blue and the unique connecting deceleration $\mu[\xi_1]$ to the cover discontinuity at t_1 in green. We truncated the start of both deceleration boundaries for a more compact figure. The careful observer may notice that μ cannot occur as the minimum boundary defined in (5.10), but please note that the class of piecewise trajectories $\mathcal{P}[a,b]$ is just slightly more general than necessary for our current purposes.

Piecewise trajectories. Let $\mu \in \mathcal{P}[a,b]$ be some piecewise trajectory with corresponding subdivision $a = t_0 < \cdots < t_{n+1} = b$ as defined in Definition 5.3. It is straightforward to generalize the definition of a deceleration boundary to μ. Whenever $\xi \in [a,b] \setminus \{t_1, \dots, t_n\}$, we just define $\mu[\xi] := x^-[\mu(\xi), \dot{\mu}(\xi), \xi]$, exactly like we did for x. However, at the points of discontinuity $\xi \in \{t_1, \dots, t_n\}$, the derivative $\dot{\mu}(\xi)$ is not defined, so we choose to use the left-sided limit instead, by defining $\mu[\xi] := x^-[\mu(\xi), \dot{\mu}(\xi^-), \xi]$.

Remark 5.2. *Please note that we cannot just replace x with μ in inequality* (5.13) *to obtain a similar bound for μ on the its full interval $[a,b]$. Instead, we get the following* piecewise lower bounding *property. Consider some interval $I \in \{[a,t_1], (t_1,t_2], \dots, (t_n,b]\}$, then what remains true is that $\xi \in I$ implies $\mu(t) \geq \mu[\xi](t)$ for every $t \in I$.*

5.2.4 Smoothing procedure

Let $\mu \in \mathcal{P}[a,b]$ be some piecewise trajectory and let $a = t_0 < \cdots < t_{n+1} = b$ denote the subdivision as in Definition 5.3. We first show how to smoothen the discontinuity at t_1 and then argue how to repeat this process for the remaining times t_i. Our aim is to choose some time $\xi \in [a,t_1]$ from which the vehicle starts fully decelerating, such that $\mu[\xi] \preceq \mu$ and such that $\mu[\xi]$ touches μ at some time $\tau \in [t_1,b]$ tangentially. We will show there is a unique trajectory $\mu[\xi]$ that satisfies these requirements and refer to it as the *connecting deceleration*, see Figure 5.4 for an example. The construction relies on the following technical assumption.

Assumption 5.2. *Throughout the following discussion, we assume $\mu \succeq \mu[a]$ and $\mu \succeq \mu[b]$.*

Touching. Recall Remark 5.2, which asserts that we have $\mu[\xi](t) \leq \mu(t)$ for every $t \in [a,t_1]$ for any $\xi \in [a,t_1]$. After the discontinuity, so for every $t \in [t_1,b]$, we want $\mu[\xi](t) \leq \mu(t)$ and equality at least somewhere, so we measure the relative position of $\mu[\xi]$ with respect to μ here, by considering

$$d(\xi) := \min_{t \in [t_1,b]} \mu(t) - \mu[\xi](t). \tag{5.16}$$

Since $\mu(t)$ and $\mu[\xi](t)$ are both continuous as a function of t on the interval $[t_1,b]$, this minimum actually exists (extreme value theorem). Furthermore, since d is the minimum of a continuous function over a closed interval, it is continuous as well (see Lemma E.1).

Observe that $d(a) \geq 0$, because $\mu \succeq \mu[a]$ by Assumption 5.2. By definition of t_1, we have $\dot{\mu}(t_1^-) > \dot{\mu}(t_1^+)$, from which it follows that $\mu(t) < \mu[t_1](t)$ for $t \in (t_1, t_1 + \epsilon)$ for some small $\epsilon > 0$, which shows that $d(t_1) < 0$. By the intermediate value theorem, there is $\xi_1 \in [a, t_1)$ such that $d(\xi_1) = 0$. This shows that $\mu[\xi_1]$ touches μ at some time $\tau_1 \in [t_1, b]$.

Uniqueness. It turns out that ξ_1 itself is not necessarily unique, which we explain below. Instead, we are going to show that the connecting deceleration $\mu[\xi_1]$ is unique. More precisely, given any other $\xi \in [a, t_1)$ such that $d(\xi) = 0$, we will show that $\mu[\xi] = \mu[\xi_1]$.

The first step is to establish that the level set

$$X := \{\xi \in [a, t_1) : d(\xi) = 0\} \tag{5.17}$$

is a closed interval. To this end, we show that d is non-increasing on $[a, t_1)$, which together with continuity implies the desired result (see Lemma E.2). To show that d is non-increasing, it suffices to show that $\mu[\xi](t)$ is non-decreasing as a function of ξ, for every $t \in [t_1, b]$. We can do this by computing the partial derivative of $\mu[\xi]$ with respect to ξ and verifying it is non-negativity. Recall the definition of $\mu[\xi]$, based on x^- in equation (5.15). Using similar notation, we write $\delta_1(\xi) = \xi - (1 - \dot{\mu}(\xi))/\omega$ and $\delta_0(\xi) = \xi + \dot{\mu}(\xi)/\omega$ and compute

$$\frac{\partial}{\partial \xi}\mu[\xi](t) = \dot{\mu}(\xi) + \begin{cases} \ddot{\mu}(\xi)(\dot{\mu}(\xi) - 1)/\omega - 1 & \text{for } t \leq \delta_1(\xi), \\ \ddot{\mu}(\xi)(t - \xi) - \dot{\mu}(\xi) + \omega(t - \xi) & \text{for } t \in [\delta_1(\xi), \delta_0(\xi)], \\ \ddot{\mu}(\xi)\dot{\mu}(\xi)/\omega & \text{for } t \geq \delta_0(\xi). \end{cases} \tag{5.18}$$

It is easily verified that the cases match at $t \in \{\delta_1(\xi), \delta_0(\xi)\}$, which justifies the overlaps there. Consider any $\xi \in [a, t_1)$ and $t \in [t_1, b]$, then we always have $\delta_1(\xi) \leq \xi \leq t$, so we only have to verify the second and third case:

$$\frac{\partial}{\partial \xi}\mu[\xi](t) = (\ddot{\mu}(\xi) + \omega)(t - \xi) \geq 0 \qquad \text{for } t \in [\delta_1(\xi), \delta_0(\xi)], \tag{5.19a}$$

$$\frac{\partial}{\partial \xi}\mu[\xi](t) \geq \dot{\mu}(\xi) + (-\omega)\dot{\mu}(\xi)/\omega = 0 \qquad \text{for } t \geq \delta_0(\xi). \tag{5.19b}$$

This concludes the argument for X being a closed interval.

Assuming ξ to be fixed, observe that there is equality in (5.19a) for some $t \in [\delta_1(\xi), \delta_0(\xi)]$ if and only if there is equality in (5.19b) for some other $t' \geq \delta_0(\xi)$. Note that this happens precisely when $\ddot{\mu}(\xi) = -\omega$. Therefore, whenever μ is fully deceleration, so $\dot{\mu}(t) = -\omega$ on some open interval $U \subset (a, t_1)$, we have $(\partial/\partial\xi)\mu[\xi](t) = 0$ for all $t \geq \delta_1(\xi)$. This essentially means that any choice of $\xi \in U$ produces the same trajectory $\mu[\xi]$. Please see Figure 5.5 for an example of this case. This observation is key to the remaining uniqueness argument.

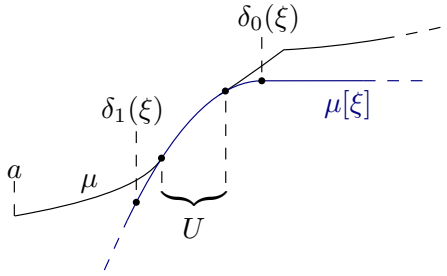


Figure 5.5: Example of a piecewise trajectory μ with a part of full deceleration over some interval U such that any choice of $\xi \in U$ produces the same deceleration boundary $\mu[\xi]$, which naturally coincides with μ on U.

Since X is a closed interval, we may define $\xi_0 = \min X$. Consider any $\xi' \in X$ with $\xi' > \xi_0$, then we show $\mu[\xi'](t) = \mu[\xi_0](t)$ for all $t \in [\xi_0, b]$. For sake of contradiction, suppose there is some $t' \in [\xi_0, b]$ such that $\mu[\xi'](t') > \mu[\xi_0](t')$, then there must be some open interval

$U \subset (\xi_0, \xi')$ such that

$$\frac{\partial}{\partial \xi} \mu[\xi](t') > 0 \text{ for all } \xi \in U. \tag{5.20}$$

However, we argued in the previous paragraph that this actually holds for any $t' \geq \delta_1(\xi)$. In particular, let $t^* \in [t_1, b]$ be such that $\mu(t^*) = \mu[\xi_0](t^*)$, then $t^* \geq t_1 \geq \xi \geq \delta_1(\xi)$, so (5.20) yields $\mu[\xi'](t^*) > \mu[\xi_0](t^*)$, but then $d(\xi') > d(\xi_0) = 0$, so $\xi' \notin X$, a contradiction.

Touching tangentially. It remains to show that μ and $\mu[\xi_0]$ touch tangentially somewhere on $[t_1, b]$. Let $\tau_1 \in [t_1, b]$ be the smallest time such that $\mu(\tau_1) - \mu[\xi_0](\tau_1) = d(\xi_0) = 0$ and consider the following three cases.

First of all, note that $\tau_1 = t_1$ is not possible, because this would require

$$\dot{\mu}(t_1^+) > \dot{\mu}[\xi_0](t_1^+) = \dot{\mu}[\xi_0](t_1), \tag{5.21}$$

but since μ is a piecewise trajectory, we must have $\dot{\mu}(t_1^-) > \dot{\mu}(t_1^+) > \dot{\mu}[\xi_0](t_1)$. This shows that $\mu(t_1 - \epsilon) < \mu[\xi_0](t_1 - \epsilon)$, for some small $\epsilon > 0$, which contradicts $\mu[\xi_0] \preceq \mu$.

Suppose $\tau_1 \in (t_1, b)$, then recall the definition of $d(\xi_0)$ and observe that the usual first-order necessary condition (derivative zero) for local minima requires $\dot{\mu}(\tau_1) = \dot{\mu}[\xi_0](\tau_1)$.

Finally, consider $\tau_1 = b$. Observe that $\dot{\mu}(b) > \dot{\mu}[\xi_0](b)$, would contradict minimality of $\tau_1 = b$. Therefore, suppose $\dot{\mu}(b) < \dot{\mu}[\xi_0](b)$, then $\dot{\mu}[b](b) = \dot{\mu}(b) < \dot{\mu}[\xi_0](b)$, so

$$\dot{\mu}[b](t) \leq \dot{\mu}[\xi_0](t) \text{ for } t \leq b, \tag{5.22}$$

but then $\mu[b](t) > \mu[\xi_0](t)$ for $t < b$. In particular, for $t = \xi_0$, this shows $\mu[b](\xi_0) > \mu[\xi_0](\xi_0) = \mu(\xi_0)$, which contradicts part $\mu[b] \preceq \mu$ of Assumption 5.2.

Repeat for remaining discontinuities. Let us summarize what we have established so far. The times $\xi_0 \in [a, t_1)$ and $\tau_1 \in (t_1, b]$ have been chosen such that

$$\mu[\xi_0](t) \leq \mu(t) \text{ for } t \in [\xi_0, \tau_1], \tag{5.23a}$$

$$\dot{\mu}[\xi_0](\xi_0) = \dot{\mu}(\xi_0) \text{ and } \dot{\mu}[\xi_0](\tau_1) = \dot{\mu}(\tau_1). \tag{5.23b}$$

Instead of ξ_0, it will be convenient later to choose $\xi_1 := \max X$ as the representative of the unique connecting deceleration. We can now use $(\mu[\xi_1])|_{[\xi_1, \tau_1]}$ to replace μ at $[\xi_1, \tau_1]$ to obtain a trajectory without the discontinuity at t_1. More precisely, we define

$$\mu_1(t) = \begin{cases} \mu(t) & \text{for } t \in [a, \xi_1] \cup [\tau_1, b], \\ \mu[\xi_1](t) & \text{for } t \in [\xi_1, \tau_1]. \end{cases} \tag{5.24}$$

From the way we constructed $\mu[\xi_1]$, it follows from (5.23) that we have $\mu_1 \in \mathcal{P}[a, b]$, but without the discontinuity at t_1. Observe that a single connecting deceleration may cover more than one discontinuity, as illustrated in Figure 5.6. Note that we must have $\dot{\mu}_1(a) = \dot{\mu}(a)$ and $\dot{\mu}_1(b) = \dot{\mu}(b)$ by construction. Hence, it is not difficult to see that μ_1 must still satisfy Assumption 5.2, so that we can keep repeating the exact same process, obtaining connecting decelerations $(\xi_2, \tau_2), (\xi_3, \tau_3), \ldots$ and the corresponding piecewise trajectories $\mu_2, \mu_3, \ldots$ to remove any remaining discontinuities until we end up with a smooth trajectory $\mu^* \in \mathcal{D}[a, b]$. We emphasize again that $\dot{\mu}^*(a) = \dot{\mu}(a)$ and $\dot{\mu}^*(b) = \dot{\mu}(b)$.

Proof of Theorem 5.1. Let us now return to the minimum boundary γ defined in (5.10). From Figure 5.2 and the conditions of Theorem 5.1, it is clear that γ must satisfy $\gamma(a) = A$, $\gamma(b) = B$ and $\dot{\gamma}(a) = \dot{\gamma}(b) = 1$, so whenever we have $\gamma \in \mathcal{D}[a, b]$, i.e., γ does not contain discontinuities, we automatically have $\gamma \in D_2[a, b]$ so that γ itself is already a feasible solution. Explain why Assumption 5.2 holds. Otherwise, we perform the smoothing procedure presented above to obtain the smoothed trajectory $\gamma^* \in \mathcal{D}[a, b]$. This completes the proof of Theorem 5.1.

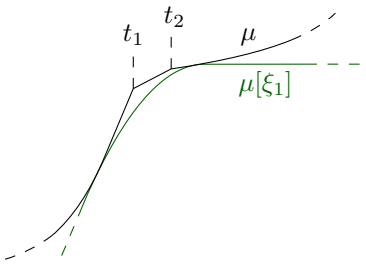


Figure 5.6: Part of a piecewise trajectory μ on which a single connecting deceleration covers the two discontinuities at t_1 and t_2 at once.

5.2.5 Upper boundary solution

As a byproduct of the above analysis, the next lemma shows that the solution γ^* is also an upper boundary for any other feasible trajectory.

Lemma 5.3. *Let $\mu \in \mathcal{P}[a,b]$ be a piecewise trajectory and let $\mu^* \in \mathcal{D}[a,b]$ denote the result after smoothing. All trajectories $x \in \mathcal{D}[a,b]$ that are such that $x \preceq \mu$, must satisfy $x \preceq \mu^*$.*

Proof. Consider some interval (ξ, τ) where we introduced some connecting deceleration boundary. Suppose there exists some $t_d \in (\xi, \tau)$ such that $x(t_d) > \mu(t_d)$. Because $x(\xi) \leq \mu(\xi)$, this means that x must intersect μ at least once in $[\xi, t_d)$, so let $t_c := \sup\{t \in [\xi, t_d) : x(t) = \mu(t)\}$ be the latest time of intersection such that $x(t) \geq \mu(t)$ for all $t \in [t_c, t_d]$. There must be some $t_v \in [t_c, t_d]$ such that $\dot{x}(t_v) > \dot{\mu}(t_v)$, otherwise

$$x(t_d) = x(t_c) + \int_{t_c}^{t_d} \dot{x}(t)\,\mathrm{d}t \leq \mu(t_c) + \int_{t_c}^{t_d} \dot{\mu}(t)\,\mathrm{d}t = \mu(d_t),$$

which contradicts our choice of t_d. Hence, for every $t \in [t_v, \tau]$, we have

$$\dot{x}(t) \geq \dot{x}(t_v) - \omega(t - t_v) > \dot{\mu}(t_v) - \omega(t - t_v) = \dot{\mu}(t).$$

It follows that $x(\tau) > \mu(\tau)$, which contradicts the assumption $x \preceq \mu$. □

Remark 5.3. *The above upper boundary property has the following interesting consequence if we extend the single vehicle problem to an optimal control problem by considering maximizing the haste criterion $J(x)$ defined in (5.6) as optimization objective. In particular, observe that it follows from the above lemma that any other $x \in D_2[a,b]$ satisfying $x \preceq u$ must also satisfy*

$$\int_a^b x(t)\,\mathrm{d}t \leq \int_a^b \gamma^*(t)\,\mathrm{d}t \tag{5.25}$$

Consequently, $x = \gamma^*$ is an optimal solution to the single vehicle optimal control problem

$$\max_{x \in D_2[a,b]} J(x) \quad \textit{such that} \quad x \preceq u. \tag{5.26}$$

5.3 Lane planning feasibility

We will now return to the feasibility of the lane planning problem and show how it decomposes in terms of a sequence of single vehicle feasibility problems. Let us first restate the conditions for feasible solutions of the lane planning problem. Recall that we are given schedule times $a = (a_1, a_2, \ldots, a_N)$ and $b = (b_1, b_2, \ldots, b_N)$, which are assumed to be ordered as $a_1 \leq \cdots \leq a_N$ and $b_1 \leq \cdots \leq b_N$. For brevity, we will write $x \in D_2^N[a,b]$ to denote the vector $x = (x_1, \ldots, x_N)$ of N trajectories $x_i \in D_2[a_i, b_i]$. Assume the system parameters $(\omega, \bar{\omega}, A, B, L)$ to be fixed, then the goal is to find a sequence of trajectories $x \in D_2^N[a,b]$ such that

$$x_i \in D_2[a_i, b_i] \quad \text{for each } i \in \{1, \ldots, N\}, \tag{5.27a}$$
$$x_i \preceq x_{i-1} - L \quad \text{for each } i \in \{2, \ldots, N\}. \tag{5.27b}$$

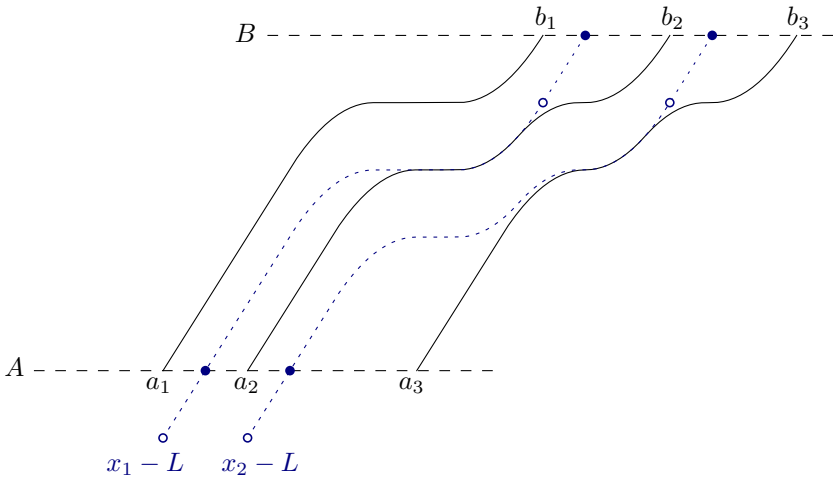


Figure 5.7: Optimal trajectories x_i for three vehicles. The dotted blue trajectories between the little open circles illustrates the safe following constraints (5.27b). The dotted blue trajectories between the solid dots are the following boundaries $\bar{x}_2 \in \bar{D}[\bar{a}_2, \bar{b}_2]$ and $\bar{x}_3 \in \bar{D}[\bar{a}_3, \bar{b}_3]$.

The general idea is to repeat the construction of the previous section for each vehicle to obtain a solution x_i, while using each constructed trajectory as the boudary $u = x_i$ for the next problem of finding $x_{i+1} \preceq u$. We will show that feasibility is equivalent to having the schedule times a_i and b_i satisfy a certain system of inequalities. We will need the following technical assumption regarding the minimum length of the lane, which is very reasonable to assume in practice.

Assumption 5.3. *Assume that vehicle lengths are limited by $L < B - A$.*

Now consider the safe following constraints (5.27b). We show how transform these into equivalent upper boundaries $\bar{x}_i \in \bar{D}_1[\bar{a}_i, \bar{b}_i]$ for each $i \in \{2, \dots, N\}$, such we can apply Theorem 5.1. It becomes clear from Figure 5.7 that inequality (5.27b) only applies on some subinterval $I_i \subset [a_{i-1}, b_{i-1}]$. However, as the figure suggests, we can easily truncate and extend these boundaries as necessary. For some $y \in \mathcal{D}[\alpha, \beta]$, we define the inverse at some position p in its range to be

$$y^{-1}(p) = \inf\{t \in [\alpha, \beta] : y(t) = p\}. \tag{5.28}$$

Given some trajectory $u \in D_1[c, d]$, we define its *downshift*

$$\bar{u}(t) = \begin{cases} u(t) - L & \text{for } t \in [u^{-1}(A + L), d], \\ B - L + t - d & \text{for } t \in [d, d + L]. \end{cases} \tag{5.29}$$

For ease of reference, we denote the endpoints of its domain as $\bar{a} := u^{-1}(A+L)$ and $\bar{b} := d+L$.

Lemma 5.4 (Boundary extension)**.** *Consider some trajectory $u \in \mathcal{D}[c, d]$ such that $u(d) \geq A$. If $x \in \mathcal{D}[a, b]$ is such that $x(a) = A$ and $x \preceq u$, then it satisfies $x \preceq (u(d) + t - d)|_{[d,\infty)}$, which may be interpreted as extending the upper boundary u to the right at full speed.*

Proof. If $b < d$, then $x \preceq (\cdot)|_{[d,\infty)}$ is always void and the statement is trivially true. Assume $b \geq d$ and consider an arbitrary $t \geq d$. Suppose $a \leq d$, then we have $x(t) \leq x(d) + t - d \leq u(d) + t - d$. Suppose $a > d$, then we have $x(t) \leq x(a) + t - a = A + t - a \leq u(d) + t - d$. □

Lemma 5.5 (Downshift boundary equivalence)**.** *For each $u \in D_2[c, d]$, the downshift trajectory satisfies $\bar{u} \in D_1[\bar{a}, \bar{b}]$. For each $x \in D[a, b]$ such that $a \geq c$ and $b \geq d$, we have $x \preceq u - L$ if and only if $x \preceq \bar{u}$.*

Proof. The two cases in the definition of $\bar{u}$ coincide, so that $\bar{u} \in \mathcal{D}$. Furthermore, it is easily verified that $\bar{u}(\bar{a}) = A$ and $\bar{u}(\bar{b}) = B$, so the first claim follows.

Suppose $x \preceq u - L$, and suppose there exists some $t \in [a,b] \cap [c,d]$. If $t \in [\bar{a}, d]$, then $x(t) \leq u(t) - L = \bar{u}(t)$ by definition. If $t \in [d, \bar{b}]$, then apply Lemma 5.4 to $u - L$ (using Assumption 5.3 for $u(d) - L \geq A$) to obtain $x \leq (\tau \mapsto u(d) - L + \tau - d)|_{[d,\infty)} = (\tau \mapsto B - L + \tau - d)|_{[d,\infty)}$, so that $x(t) \leq B - L + t - d = \bar{u}(t)$.

For the other direction, suppose $x \preceq \bar{u}$. First of all, since $u(c) = A$ and $u(\bar{a}) = A + L > A$ and u is non-decreasing, we have $c < \bar{a}$. Suppose $c \leq a < \bar{a}$, then since $b \geq d \geq \bar{a}$ and $\dot{x}(a) = 1$, we must have $x(\bar{a}) > x(a) = A = \bar{u}(\bar{a})$, contradicting the initial assumption. Hence, $a \geq \bar{a}$, so any $t \in [a,b] \cap [c,d]$ satisfies $t \in [\bar{a}, d]$, but then $x(t) \leq \bar{u}(t) = u(t) - L$ by definition. □

The following lemma summarizes what we have established so far.

Lemma 5.6. *The following four statements are equivalent:*

(C0) The lane planning problem is feasible.

(C1) There exists $x \in D_2^N[a,b]$ such that $x_i \preceq x_{i-1} - L$ for all $i \in \{2, \dots, N\}$.

(C2) There exists $x \in D_2^N[a,b]$ such that $x_i \preceq \bar{x}_{i-1}$ for all $i \in \{2, \dots, N\}$.

(C3) There exists $x \in D_2^N[a,b]$ such that $b_i - a_i \geq B - A$ for all $i \in \{1, \dots, N\}$; and

- *(i)* $b_i \geq \bar{b}_{i-1}$, *(exit order constraint)*
- *(ii)* $a_i \geq \bar{a}_{i-1}$, *(entry order constraint)*
- *(iii)* $\bar{x}_{i-1} \succeq \check{x}_i$, *(entry space constraint)*

for all $i \in \{2, \dots, N\}$.

Proof. Of course, (C0) and (C1) are equivalent by definition of the lane planning problem. Note that equivalence of (C1) and (C2) is handled by Lemma 5.5. Equivalence of (C2) and (C3) follows from a straightforward application of Theorem 5.1 by setting $x = x_i$ and $u = \bar{x}_{i-1}$ for each $i \in \{2, \dots, N\}$. □

Simpler conditions. The goal of the remainder of this section is to get rid of the entry space constraints and replace them with an inequality constraints in terms of schedule times. Moreover, we want to get rid of the dependence of $\bar{a}_{i-1}$ on $\bar{x}_{i-1}$, such that we obtain equivalent statements in a_{i-1}. Note that the exit order constraint is already in the desired form, because we have $\bar{b}_{i-1} = b_{i-1} + L$. More specifically, we will thus show that the statements of Lemma 5.6 above are further equivalent to:

(C4) We have $b_i - a_i \geq B - A$ for all $i \in \{1, \dots, N\}$; and

- (i*) $b_i \geq b_{i-1} + L$,
- (ii*) $a_i \geq a_{i-1} + L$,

for all $i \in \{2, \dots, N\}$; and

- (c*) $a_i \geq \check{a}_i(a,b)$, (entry time constraint)

for all $i \in \{n, \dots, N\}$,

for some $n \geq 2$ and where $\check{a}_i(a,b)$ denotes some expression in terms of schedule times. Consequently, $\max\{a_{i-1} + L, \check{a}_i(a,b)\}$ can be interpreted as the earliest possible time of arrival to the lane for vehicle i.

Before we are able to prove this equivalence, we will need some better understanding of the smoothing procedure, so we will derive explicit formulas for finding the touching times ξ and τ to obtain optimal trajectories under the hast objective.

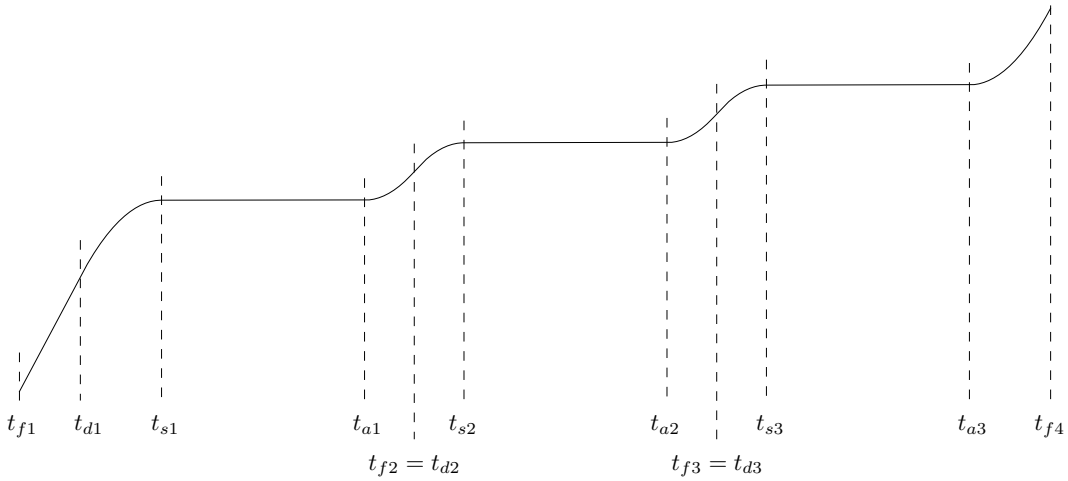


Figure 5.8: Example of an alternating vehicle trajectory with its defining time intervals. The particular shape of this trajectory is due to two leading vehicles, which causes the two start-stop *bumps* around the times where these leading vehicles depart from the lane.

5.4 Optimal solution for haste objective

Due to the recursive nature of the problem, we will see that optimal trajectories for the haste objective (minimizing $J_{\alpha,\beta}$ with $\alpha = -1, \beta = 0$) possess a particularly simple structure, which enables a simple computation.

Definition 5.4. Let a trajectory $\gamma \in \mathcal{D}[a,b]$ be called *alternating* if for all $t \in [a,b]$, we have

$$\ddot{\gamma}(t) \in \{-\omega, 0, \bar{\omega}\} \quad \text{and} \quad \ddot{\gamma}(t) = 0 \implies \dot{\gamma}(t) \in \{0,1\}. \tag{5.30}$$

We now argue that each vehicle's optimal trajectory x_i is alternating. First, consider $x_1 = x^*(a_1, b_1, \varnothing)$, which is constructed by joining $x^1[x_1]$ and $\hat{x}[x_1]$ together by smoothing. Observe that both boundaries are alternating by definition. Let $\gamma_1(t) = \min\{x^1[x_1](t), \hat{x}[x_1](t)\}$ be the minimum boundary, then it is clear that the smoothened $x_1 = \gamma_1^*$ must also be alternating, because we only added a part of deceleration at some interval $[\xi, \tau]$, which clearly satisfies $\ddot{\gamma}_1^*(t) = -\omega$ for $t \in [\xi, \tau]$. Assume that x_{i-1} is alternating, we can similarly argue that x_i is alternating. Again, let $\gamma_i(t) = \min\{\bar{x}[x_{i-1}], \hat{x}[x_i](t), x^1[x_i](t)\}$ be the minimum boundary. After adding the required decelerations for smoothing, it is clear that $x_i = \gamma_i^*$ must also be alternating.

Observe that an alternating trajectory $\gamma \in \mathcal{D}[a,b]$ can be described as a sequence of four types of consecutive repeating phases, see Figure 5.8 for an example. In general, there exists a partition of $[a,b]$, denoted by

$$a = t_{f1} \le t_{d1} \le t_{s1} \le t_{a1} \le t_{f2} \le t_{d2} \le t_{s2} \le t_{a2} \le \cdots \le t_{f,n+1} = b,$$

such that we have the consecutive *alternating intervals*

$$\begin{aligned} F_i &:= [t_{f,i}, t_{d,i}] \quad \text{(full speed)}, & S_i &:= [t_{s,i}, t_{a,i}] \quad \text{(stopped)}, \\ D_i &:= [t_{d,i}, t_{s,i}] \quad \text{(deceleration)}, & A_i &:= [t_{a,i}, t_{f,i+1}] \quad \text{(acceleration)}, \end{aligned}$$

such that on these intervals, γ satisfies

$$\begin{aligned} \dot{\gamma}(t) &= 1 \quad \text{for } t \in F_i, & \dot{\gamma}(t) &= 0 \quad \text{for } t \in S_i, \\ \ddot{\gamma}(t) &= -\omega \quad \text{for } t \in D_i, & \ddot{\gamma}(t) &= \bar{\omega} \quad \text{for } t \in A_i. \end{aligned}$$

Partial trajectories. Next, we will define parameterized functions x^f, x^d, x^s, x^a to describe alternating trajectory γ on each of these alternating intervals. Given some initial position

$p \in [A, B]$, velocity $v \in [0, 1]$, start and end times a and b such that $a \leq b$ and $v + \bar{\omega}(b-a) \leq 1$, we define the acceleration trajectory $x^a[p, v, a, b] : [a, b] \to \mathbb{R}$ by setting

$$x^a[p, v, a, b](\tau) := p + v(\tau - a) + \bar{\omega}(\tau - a)^2/2, \tag{5.31a}$$
$$\dot{x}^a[p, v, a, b](\tau) := v + \bar{\omega}(\tau - a). \tag{5.31b}$$

Similarly, for p, v, a, b satisfying $a \leq b$ and $v - \omega(b - a) \geq 0$, let the deceleration trajectory $x^d[p, v, a, b] : [a, b] \to \mathbb{R}$ be defined as

$$x^d[p, v, a, b](\tau) := p + v(\tau - a) - \omega(\tau - a)^2/2, \tag{5.32a}$$
$$\dot{x}^d[p, v, a, b](\tau) := v - \omega(\tau - a). \tag{5.32b}$$

One may notice that x^d is essentially the same as the deceleration boundary x^-, which we defined in Section 5.2.3. However, note that the condition $v - \omega(b - a) \geq 0$ restricts the domain such that we do not need the clipping operation. Furthermore, the parameterization of x^d will be more convenient in the next section.

We use the following notation for trajectories with constant minimum or maximum speed. We write $x^s[p, a, b](\tau) \equiv p$, with domain $[a, b]$, to model a stopped vehicle and let $x^f[p, a, b](\tau) = (p + \tau - a, 1)$ model a vehicle that drives at full speed, also with domain $[a, b]$.

5.4.1 Connecting partial trajectories

It can be shown that the smoothing procedure introduces a part of deceleration only between the four pairs of partial trajectories

$$x^a \to x^a, \qquad x^a \to x^s, \qquad x^f \to x^a, \qquad x^f \to x^s.$$

We will use these results to characterize optimal trajectories for our optimal control problem.

Lemma 5.7 ($x^f \to x^s$)**.** *Let $x^f[p, a, b]$ and $x^s[q, c, d]$ be two trajectories. Considering τ_1 and τ_2 as variables in the equation*

$$x^d[x^f[p, a, b](\tau_1), \tau_1, \tau_2](\tau_2) = x^s[q, c, d](\tau_2),$$

it has solution $\tau_2 = q - p + a + 1/2\omega$ and $\tau_1 = \tau_2 - 1/\omega$, whenever $\tau_1 \in [a, b]$ and $\tau_2 \in [c, d]$.

Proof. The expanded system of state equations is given by

$$\begin{cases} p + \tau_1 - a + (\tau_2 - \tau_1) - \omega(\tau_2 - \tau_1)^2/2 = q, \\ 1 - \omega(\tau_2 - \tau_1) = 0. \end{cases}$$

The second equation yields $\tau_2 - \tau_1 = 1/\omega$, which after substituting back in the first equation yields $p - a + \tau_2 - 1/2\omega - q = 0$, from which the stated solution follows. □

To keep the expressions for the case of joining $x^f \to x^a$ a little bit simpler, we first consider a full line joining to a acceleration trajectory of full length $1/\omega$.

Lemma 5.8. *Consider some full acceleration trajectory $x^a[(p, 0), a, a + 1/\omega]$ and the line through $(\lambda, 0)$ with slope 1. Whenever λ, which can be interpreted as a time epoch, satisfies $\lambda \in [a - p - 1/2\omega, a - p + 1/2\omega]$, then the equation*

$$x^+[(p, 0), a, a + 1/\omega](\tau) = x^d[(q, 1), q + \lambda, q + \lambda + 1/\omega](\tau),$$

with τ and q considered as variables, has a unique solution

$$\tau = a + 1/\omega - \sqrt{\frac{a - p + 1/2\omega - \lambda}{\omega}},$$
$$q = 2\tau - a - 1/\omega - \lambda,$$

so the joining deceleration is given by $x^d[(q, 1), q + \lambda, \tau]$

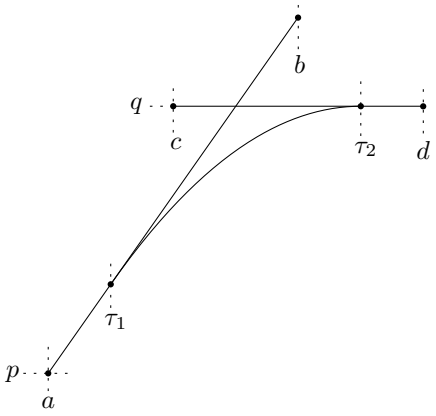


Figure 5.9: $x^f \to x^s$

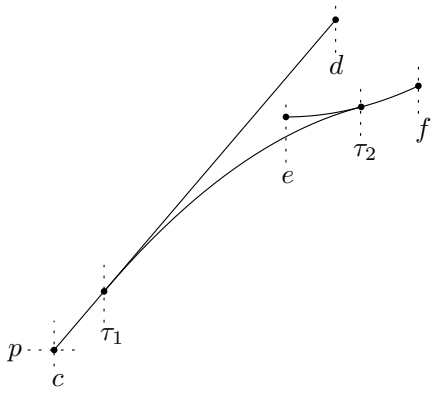


Figure 5.10: $x^f \to x^a$

Proof. First of all, the expanded system of state equations is given by

$$\begin{cases} p + \omega(\tau - a)^2/2 = q + (\tau - q - \lambda) - \omega(\tau - q - \lambda)^2/2, \\ \omega(\tau - a) = 1 - \omega(\tau - q - \lambda). \end{cases}$$

We use the second equation to express q in terms of τ, which yields

$$q = 2\tau - 1/\omega - a - \lambda,$$

which we substitute in the first equation to derive the equation

$$\omega\tau^2 - 2(1 + \omega a)\tau + \omega a^2 + a + p + \lambda + 1/2\omega = 0.$$

This is a quadratic equation in τ, with solutions

$$\tau = a + 1/\omega \pm \sqrt{\frac{a - p + 1/2\omega - \lambda}{\omega}},$$

of which only the smallest one is valid, because $\tau \leq a + 1/\omega$. Furthermore, we see that τ is defined as a real number when

$$a - p + 1/2\omega - \lambda \geq 0 \iff \lambda \leq a - p + 1/2\omega.$$

The other requirement is that $\tau \geq a$, which is equivalent to

$$1/\omega \geq \sqrt{\frac{a - p + 1/2\omega - \lambda}{\omega}} \iff \lambda \geq a - p - 1/2\omega.$$

□

Lemma 5.9 ($x^f \to x^a$)**.** *Consider partial trajectories $x^f[p, c, d]$ and $x^a[x, e, f]$.*

Proof. First of all, observe that $x^f[p, c, d]$ lies on the line with slope 1 through $(\lambda, 0) := (c-p, 0)$ and $x^a[x, e, f]$ lies on the full acceleration curve $x^a[(x_1 - x_2^2/(2\omega), 0), e - x_2/\omega, e - x_2/\omega + 1/\omega]$, see Figure 5.10. Now apply Lemma 5.8 to $p = x_1 - x_2^2/(2\omega)$, $a = e - x_2/\omega$ and $\lambda = c - p$ yields some solutions τ and q. Let $\tau_2 := \tau$ and let τ_1 denote the time where the line and x^a join, given by $\tau_1 = \lambda + q$. Now we simply check whether this solution is also feasible for the smaller trajectories. We must have $\tau_1 \in [c, d]$ and $\tau_2 \in [e, f]$. □

Lemma 5.10. *Consider the acceleration trajectory $x^a[(p, 0), a, b]$ and the horizontal line through $(0, q)$. Let $\tau_1 = a + \sqrt{(q - p)/\omega}$ and $\tau_2 = a + 2\sqrt{(q - p)/\omega}$. If τ_1 satisfies $\tau_1 \in [a, b]$, then both trajectories are joined by deceleration trajectory $x^d[x^a[(p, 0), a, b](\tau_1), \tau_1, \tau_2]$*

Proof. Consider the following equation

$$x^d[x^a[(p,0),a,b](\tau_1),\tau_1,\tau_2](\tau_2) = (q,0).$$

The expanded system of state equations is given by

$$\begin{cases} p + \omega(\tau_1 - a)^2/2 + (\omega(\tau_1 - a))(\tau_2 - \tau_1) - \omega(\tau_2 - \tau_1)^2/2 = q, \\ \omega(\tau_1 - a) - \omega(\tau_2 - \tau_1) = 0. \end{cases}$$

From the second equation, we derive $\tau_1 - a = \tau_2 - \tau_1$. Plugging this back in the first equation yields the quadratic equation $p + \omega(\tau_1 - a)^2 = q$ with solutions $\tau_1 = a \pm \sqrt{(q-p)/\omega}$, of which only the larger one is valid. Finally, the second equation gives $\tau_2 = 2\tau_1 - a$. □

Lemma 5.11 ($x^a \to x^s$)**.** *Consider partial trajectories $x^a[x,c,d]$ and $x^s[q,e,f]$.*

Proof. Observe that $x^a[x,c,d]$ lies on the full acceleration curve $x^a[(x_1 - x_2^2/(2\omega),0), c - x_2/\omega, c - x_2/\omega + 1/\omega]$. Hence, we can apply Lemma 5.10 with $p = x_1 - x_2^2/(2\omega)$, $a = c - x_2/\omega$, which yields some solutions τ_1 and τ_2, which are feasible solutions if $\tau_1 \in [c,d]$ and $\tau_2 \in [e,f]$. □

Lemma 5.12. *Consider full acceleration trajectories $x^a[(p,0),a,b]$ and $x^a[(q,0),c,d]$.*

Proof. Consider the equation

$$x^d[x^a[(p,0),a,b](\tau_1),\tau_1,\tau_2](\tau_2) = x^a[(q,0),c,d](\tau_2),$$

expanded to the system of equations

$$\begin{cases} p + \omega(\tau_1 - a)^2/2 + \omega(\tau_1 - a)(\tau_2 - \tau_1) - \omega(\tau_2 - \tau_1)^2/2 = q + \omega(\tau_2 - c)^2/2, \\ \omega(\tau_1 - a) + \omega(\tau_2 - \tau_1) = \omega(\tau_2 - c). \end{cases}$$

□

Lemma 5.13 ($x^a \to x^a$)**.** *Consider partial trajectories $x^a[x,a,b]$ and $x^a[y,c,d]$.*

Proof. □

5.4.2 Algorithm

Put everything together into pseudocode.

Algorithm 1 Computing connecting deceleration for alternating trajectories.

Let i such that I_i is the latest such that $t_1 < I_i$.
Let j such that I_j is the earliest such that $t_1 > I_j$.

5.5 Feasibility characterization

Waiting positions. Waiting capacity is given by

$$C = \left\lfloor \frac{B - A - 1/(2\bar{\omega}) - 1/(2\omega)}{L} \right\rfloor + 1, \tag{5.33}$$

where the brackets indicate the floor function.

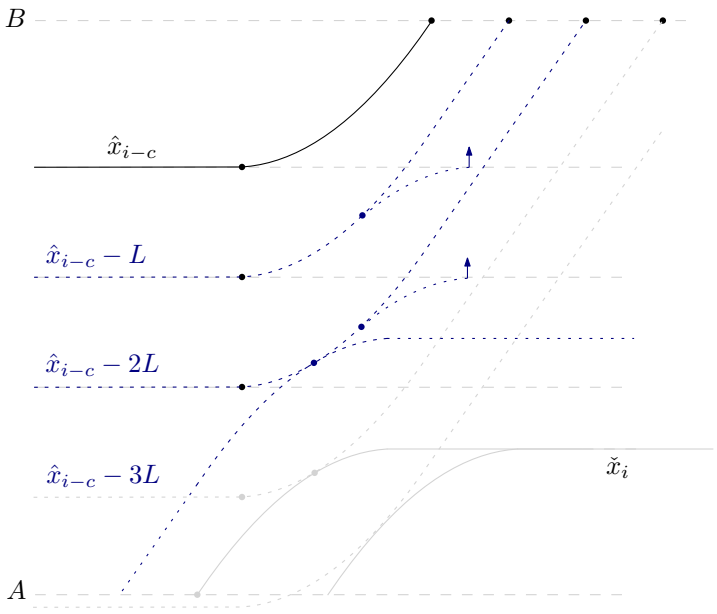


Figure 5.11: Earliest arrival due to entry order constraint and entry space constraint.

5.6 Notes and references

The analysis of feasibility conditions in this chapter is very much related to the proof of the safety guarantee in [41], see their Lemma IV.4 with relatively long proof in appendix. They study the online situation, in which new vehicles arrive to the system at later times, for which they show that the rescheduling policy is safe, in the sense that there are still feasible and collision-free trajectories for the vehicles whose crossing times got updated. We do not study such an online setting, but we take into account the fact that lanes are of finite length.

We emphasize that the assumption $\dot{x}(a) = \dot{x}(b) = 1$ was mainly made for convenience. There are different ways of relaxing the state constraints on the speed that could be studied. The interesting question is whether feasibility can still be easily characterized. Instead of fixing the speed to be maximal at entry and exit, we could require, for example, that the speed is bounded from below, i.e., $\eta \leq \dot{x}(a) \leq 1$ and $\eta \leq \dot{x}(b) \leq 1$, for some $\eta > 0$. The motivation for studying this relaxation is that this might lead to more energy efficient trajectories, whenever full speed crossing is not strictly necessary.

Chapter 6

Conclusion and discussion

- new insights from current work
- possible impact of this work
- limitations of current work
- recommendations on further work

Contributions.

- Explicit trajectory calculation
  - For haste objective
  - For general arrival/crossing times (cf. Timmerman, Joshi)

Beyond proper decomposition. We would like to drop the assumption of full speed crossing and allow more general performance measures. In this general setting, the problem does not properly decompose, so the upper- and lower-level problems must be considered simultaneously. We recognize that feasibility of the lower-level problem is a complicating factor, because it puts difficult-to-formalize constraints on the upper-level combinatorial problem. We wonder whether it is possible to use ML to approximate these constraints somehow. In other words, can we approximate the space of feasible solutions somehow, with some guarantees? The other complicating factor is the fact that the optimization objective is now a non-trivial function of the upper-level decisions variables. However, this is a very natural candidate for approximation, and something similar has already been done before [retrieve this paper].

Bibliography

[1] Ch. 10 - Trajectory Optimization. https://underactuated.mit.edu/trajopt.html.

[2] Karen Aardal, Cor Hurkens, and Jan Karel Lenstra. Jacques Benders and his decomposition algorithm. *Operations Research Letters*, 63:107361, November 2025. ISSN 0167-6377. doi: 10.1016/j.orl.2025.107361.

[3] Brandon Amos and J. Zico Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks, December 2021.

[4] Tsz-Chiu Au and Peter Stone. Motion planning algorithms for autonomous intersection management. In *Proceedings of the 1st AAAI Conference on Bridging the Gap Between Task and Motion Planning*, AAAIWS'10-01, pages 2–9. AAAI Press, January 2010.

[5] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning, January 2017.

[6] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon, March 2020.

[7] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP optimization suite 9.0. Technical Report, Optimization Online, February 2024.

[8] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer Programming*, volume 271 of *Graduate Texts in Mathematics*. Springer International Publishing, Cham, 2014. ISBN 978-3-319-11007-3 978-3-319-11008-0. doi: 10.1007/978-3-319-11008-0.

[9] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning Combinatorial Optimization Algorithms over Graphs, February 2018.

[10] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998. ISSN 0377-2217. doi: 10.1016/S0377-2217(97)00019-2.

[11] A. V. Dmitruk and A. M. Kaganovich. Maximum principle for optimal control problems with intermediate constraints. *Computational Mathematics and Modeling*, 22 (2):180–215, April 2011. ISSN 1046-283X, 1573-837X. doi: 10.1007/s10598-011-9096-8.

[12] Priya L. Donti, David Rolnick, and J. Zico Kolter. DC3: A learning method for optimization with hard constraints, April 2021.

[13] K. Dresner and P. Stone. Multiagent traffic management: A reservation-based intersection control mechanism. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.*, pages 530–537, July 2004.

[14] K. Dresner and P. Stone. A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research*, 31:591–656, March 2008. ISSN 1076-9757. doi: 10.1613/jair.2502.

[15] Ding-Zhu Du, Panos M. Pardalos, Xiaodong Hu, and Weili Wu. *Introduction to Combinatorial Optimization*, volume 196 of *Springer Optimization and Its Applications*. Springer International Publishing, Cham, 2022. ISBN 978-3-031-10594-4 978-3-031-10596-8. doi: 10.1007/978-3-031-10596-8.

[16] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact Combinatorial Optimization with Graph Convolutional Neural Networks, October 2019.

[17] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Annals of Discrete Mathematics*, volume 5 of *Discrete Optimization II*, pages 287–326. Elsevier, January 1979. doi: 10.1016/S0167-5060(08)70356-X.

[18] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines, December 2014.

[19] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2024.

[20] Lanshan Han, M. Kanat Camlibel, Jong-Shi Pang, and W. P. Maurice H. Heemels. A unified numerical scheme for linear-quadratic optimal control problems with joint control and state constraints. *Optimization Methods and Software*, 27(4-5):761–799, October 2012. ISSN 1055-6788. doi: 10.1080/10556788.2011.593624.

[21] Richard F. Hartl, Suresh P. Sethi, and Raymond G. Vickson. A Survey of the Maximum Principles for Optimal Control Problems with State Constraints. *SIAM Review*, 37(2):181–218, 1995. ISSN 0036-1445.

[22] Matthew Hausknecht, Tsz-Chiu Au, and Peter Stone. Autonomous Intersection Management: Multi-intersection optimization. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4581–4586, September 2011. doi: 10.1109/IROS.2011.6094668.

[23] Shan Huang, Adel W. Sadek, and Yunjie Zhao. Assessing the Mobility and Environmental Benefits of Reservation-Based Intelligent Intersections Using an Integrated Simulator. *IEEE Transactions on Intelligent Transportation Systems*, 13(3): 1201–1214, September 2012. ISSN 1558-0016. doi: 10.1109/TITS.2012.2186442.

[24] Robert Hult. *Optimization Based Coordination Strategies for Connected and Autonomous Vehicles*. Chalmers University of Technology, Göteborg, 2019. ISBN 978-91-7905-108-2.

[25] Robert Hult, Gabriel R Campos, Paolo Falcone, and Henk Wymeersch. Technical Report: Approximate solution to the optimal coordination problem for autonomous vehicles at intersections.

[26] Robert Hult, Gabriel R. Campos, Paolo Falcone, and Henk Wymeersch. An approximate solution to the optimal coordination problem for autonomous vehicles at intersections. In *2015 American Control Conference (ACC)*, pages 763–768, Chicago, IL, USA, July 2015. IEEE. ISBN 978-1-4799-8684-2. doi: 10.1109/ACC.2015.7170826.

[27] Purva Joshi, Marko Boon, and Sem Borst. Trajectories and Platoon-forming Algorithm for Intersections with Heterogeneous Autonomous Traffic. *ACM Journal on Autonomous Transportation Systems*, 2(3):1–32, September 2025. ISSN 2833-0528. doi: 10.1145/3701042.

[28] Matthew Kelly. An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation. *SIAM Review*, 59(4):849–904, January 2017. ISSN 0036-1445. doi: 10.1137/16M1062569.

[29] Mohammad Khayatian, Yingyan Lou, Mohammadreza Mehrabian, and Aviral Shirvastava. Crossroads+: A Time-aware Approach for Intersection Management of Connected Autonomous Vehicles. *ACM Transactions on Cyber-Physical Systems*, 4(2): 1–28, April 2020. ISSN 2378-962X, 2378-9638. doi: 10.1145/3364182.

[30] Mohammad Khayatian, Mohammadreza Mehrabian, Edward Andert, Rachel Dedinsky, Sarthake Choudhary, Yingyan Lou, and Aviral Shirvastava. A Survey on Intersection Management of Connected Autonomous Vehicles. *ACM Transactions on Cyber-Physical Systems*, 4(4):1–27, October 2020. ISSN 2378-962X, 2378-9638. doi: 10.1145/3407903.

[31] Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems!, February 2019.

[32] Leonardo Lamorgese and Carlo Mannino. A Noncompact Formulation for Job-Shop Scheduling Problems in Traffic Management. *Operations Research*, 67(6):1586–1609, November 2019. ISSN 0030-364X. doi: 10.1287/opre.2018.1837.

[33] Michael W. Levin and David Rey. Conflict-point formulation of intersection control for autonomous vehicles. *Transportation Research Part C: Emerging Technologies*, 85: 528–547, December 2017. ISSN 0968-090X. doi: 10.1016/j.trc.2017.09.025.

[34] Zhenning Li, Qiong Wu, Hao Yu, Cong Chen, Guohui Zhang, Zong Z. Tian, and Panos D. Prevedouros. Temporal-spatial dimension extension-based intersection control formulation for connected and autonomous vehicle systems. *Transportation Research Part C: Emerging Technologies*, 104:234–248, July 2019. ISSN 0968-090X. doi: 10.1016/j.trc.2019.05.003.

[35] Daniel Liberzon. *Calculus of Variations and Optimal Control Theory.*

[36] Matthijs Limpens. *Online Platoon Forming Algorithms for Automated Vehicles: A More Efficient Approach.* Bachelor, Eindhoven University of Technology, September 2023.

[37] Andrea Lodi and Giulia Zarpellon. On learning and branching: A survey. *TOP*, 25(2): 207–236, July 2017. ISSN 1863-8279. doi: 10.1007/s11750-017-0451-6.

[38] Carlo Mannino and Giorgio Sartor. The Path&Cycle Formulation for the Hotspot Problem in Air Traffic Management. page 11 pages, 2018. doi: 10.4230/OASICS.ATMOS.2018.14.

[39] Stefano Mariani, Giacomo Cabri, and Franco Zambonelli. Coordination of Autonomous Vehicles: Taxonomy and Survey. *ACM Computing Surveys*, 54(1):1–33, January 2022. ISSN 0360-0300, 1557-7341. doi: 10.1145/3431231.

[40] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement Learning for Combinatorial Optimization: A Survey, December 2020.

[41] David Miculescu and Sertac Karaman. Polling-systems-based Autonomous Vehicle Coordination in Traffic Intersections with No Traffic Signals, July 2016.

[42] Youngjae Min, Anoopkumar Sonar, and Navid Azizan. Hard-Constrained Neural Networks with Universal Approximation Guarantees, October 2024.

[43] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer International Publishing, Cham, 2016. ISBN 978-3-319-26578-0 978-3-319-26580-3. doi: 10.1007/978-3-319-26580-3.

[44] Bernard Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. Note D.S. 9, SEMA (Société d'Économie et de Mathématiques Appliquées), Paris, 1964.

[45] Igor G. Smit, Jianan Zhou, Robbert Reijnen, Yaoxin Wu, Jian Chen, Cong Zhang, Zaharah Bukhsh, Wim Nuijten, and Yingqian Zhang. Graph Neural Networks for Job Shop Scheduling Problems: A Survey, 2024.

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

[47] Éric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[48] Pavankumar Tallapragada and Jorge Cortes. Distributed control of vehicle strings under finite-time and safety specifications, July 2017.

[49] Pavankumar Tallapragada and Jorge Cortés. Hierarchical-distributed optimized coordination of intersection traffic, January 2017.

[50] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement Learning for Integer Programming: Learning to Cut, July 2020.

[51] Pierre Tassel, Martin Gebser, and Konstantin Schekotihin. A Reinforcement Learning Environment For Job-Shop Scheduling, April 2021.

[52] Pierre Tassel, Martin Gebser, and Konstantin Schekotihin. An End-to-End Reinforcement Learning Approach for Job-Shop Scheduling Problems Based on Constraint Programming, June 2023.

[53] R. W. Timmerman and M. A. A. Boon. Platoon forming algorithms for intelligent street intersections. *Transportmetrica A: Transport Science*, 17(3):278–307, February 2021. ISSN 2324-9935, 2324-9943. doi: 10.1080/23249935.2019.1692962.

[54] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks, January 2017.

[55] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning, October 2020.

[56] Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep Reinforcement Learning Guided Improvement Heuristic for Job Shop Scheduling, February 2024.

[57] Weiming Zhao, Ronghui Liu, and Dong Ngoduy. A bilevel programming model for autonomous intersection control and trajectory planning. *Transportmetrica A: Transport Science*, January 2021. ISSN 2324-9935.

Appendix

Appendix A

Feasible configurations for single intersection model

We present a way to derive the feasible configurations of the two routes that intersect at some arbitrary angle, as shown in Figure 2.1. Assume that $\alpha < \pi/2$ is the acute angle between the two intersections. Furthermore, we consider uniform rectangular vehicle geometries with $L_i \equiv L$ and $W_i \equiv W$, but the analysis is easily extended to arbitrary dimensions. We skip a thorough derivation of the following expressions, but we note that it is based on the type of the distances illustrated in Figure A.1. Roughly speaking, we encode the part of the intersection that vehicle i occupies in terms of the other vehicle's x_j coordinates, by defining the following upper and lower limit positions

$$u(x_i) := \begin{cases} -\infty & \text{if } x_i \leq B \text{ or } x_i - L \geq E, \\ B + (x_i - E)/\cos(\alpha) & \text{if } x_i \in (E, E + c\,], \\ E + (x_i - E) \cdot \cos(\alpha) & \text{if } x_i \in [E + c, E), \\ E & \text{if } x_i \geq E \text{ and } x_i - L < E, \end{cases} \tag{A.1}$$

$$l(x_i) := \begin{cases} B & \text{if } x_i - L \leq E \text{ and } x_i > E, \\ B + (x_i - L - E)/\cos(\alpha) & \text{if } x_i - L \in (E, E - c\,], \\ E + (x_i - L - E) \cdot \cos(\alpha) & \text{if } x_i - L \in [E - c, E), \\ \infty & \text{if } x_i - L \geq E \text{ or } x_i \leq E. \end{cases} \tag{A.2}$$

With these definitions, in order for the intersection to be free for vehicle j, position x_i must satisfy either $x_i < l(x_j)$ or $x_i - L > u(x_j)$ and x_j must satisfy either $x_j < l(x_i)$ or $x_j - L > u(x_i)$. Hence, these two pairs of equations completely determine the set of feasible

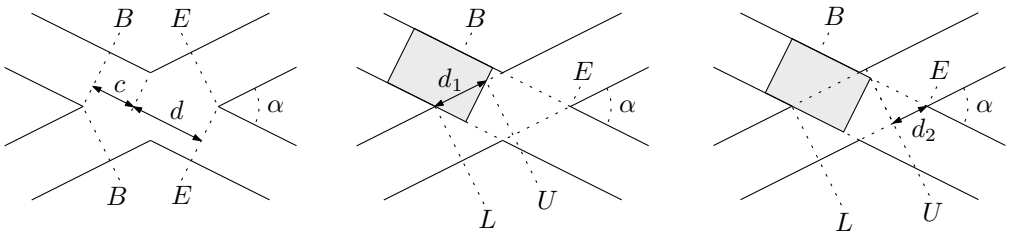


Figure A.1: Sketches to derive the feasible configurations of two vehicles in the intersecting routes model. Using some elementary trigonometry, the distances in the first figure can be shown to be $c = W/\tan(\alpha)$ and $d = W/\sin(\alpha)$. Furthermore, observe that we have $(x_i - B)/d_1 = \cos(\alpha)$ for $x_i \in (B, B + c]$, as shown in the middle figures and $d_2/(E - x_i) = \cos(\alpha)$ for $x_i \in [B + c, E)$, as shown in the right figure. These two types of distances can be used to derive the full characterization.

configurations, which can now be written as

$$\mathcal{X}_{ij} = \{(x_i, x_j) \in \mathbb{R}^j : [\, x_i - L, x_i \,] \cap [\, l(x_j), u(x_j)] = \varnothing \tag{A.3}$$

$$\text{and } [\, x_j - L, x_j] \cap [\, l(x_i), u(x_i) \,] = \varnothing\}. \tag{A.4}$$

In case the routes intersect at a right angle $\alpha = \pi/2$, the situation is much simpler and the two limiting positions are simply given by

$$(l(x_i), u(x_i)) = \begin{cases} (B, E) & \text{if } (x_i - L, x_i) \cap (B, E) \neq \varnothing, \\ (\infty, -\infty) & \text{otherwise,} \end{cases} \tag{A.5}$$

such that the set of feasible configurations is simply given by

$$\mathcal{X}_{ij} = \mathbb{R}^2 \setminus [B, E + L]^2. \tag{A.6}$$

Appendix B

Job shop scheduling

The job shop model provides a mathematical framework to study systems where a given set of—possibly distinct—facilities must be shared among a number of heterogeneous tasks over time. We begin by providing a fairly general definition of this model and then present a small example for a specific problem. Next, we introduce the disjunctive graph, which is a standard auxiliary representation of both problem instances and solutions. Finally, we briefly discuss simple heuristics and illustrate how job shop problems can be approached within the mixed-integer programming framework. For a comprehensive textbook treatment of job shop scheduling, we refer the reader to [43, Chapter 7].

General definition. Originally motivated by production planning problems, the job shop model is phrased in terms of a set of n jobs that require to be processed on a set of m machines. Each machine can process at most one job at the same time. We use the pair of indices (i, j) to identify the operation that machine i performs on job j, which takes a fixed amount of time $p(i, j)$. Each job j visits all machines[1] following a predetermined machine sequence, which may be different among jobs. Let $\mathcal{N}$ denote the set of all operations, then the general Job Shop Scheduling Problem (JSSP) is to determine a schedule $y = \{y(i, j) : (i, j) \in \mathcal{N}\}$ of starting times such that some objective function $J(y)$ is minimized. Variants of this basic problem can be obtained by specifying a concrete objective function and by introducing additional constraints, which we will both illustrate in the following example.

Example B.1. Let s_j and e_j denote the first and last machine that job j vists, respectively. For each job j, we define a so-called release date $r(j)$ by requiring that $y(s_j, j) \geq r(j)$. As objective function, we consider the so-called makespan $J(y) := \max_j y(e_j, j) + p(e_j, j)$, which we aim to minimize. The resulting problem is known as $Jm|r_j|C_{\max}$ in the commonly used three-field classification notation [17], see also [43, Chapter 2]. Now consider a specific problem instance with $m = 3$ machines and $n = 2$ jobs. We specify the order in which jobs visit machines by providing the corresponding ordering of operations, which we choose to be $(1, 1) \to (2, 1) \to (3, 1)$ and $(3, 2) \to (2, 2) \to (1, 2)$. Using matrix notation $r(j) \equiv r_j$ and $p(i, j) \equiv p_{ij}$, the release dates and processing times are given by

$$r = \begin{pmatrix} 1 & 0 \end{pmatrix}, \qquad p = \begin{pmatrix} 2 & 1 \\ 1 & 3 \\ 4 & 1 \end{pmatrix}.$$

For this problem, Figure B.1 shows an optimal schedule y^* with makespan $J(y^*) = 8$.

Disjunctive graph. A commonly used representation of job shop problems is through their disjunctive graph, which is a directed graph with vertices $\mathcal{N}$ corresponding to the operations and two types of arcs. The conjunctive arcs $\mathcal{C}$ are used to encode the predetermined machine sequence of each job. Each such arc $(i, j) \to (k, j)$ encodes that job j should first be processed

[1] When some job j requires only processing on a proper subset of the machines, observe that we can simply assume that $p(i, j) = 0$ for each machine i that is not involved.

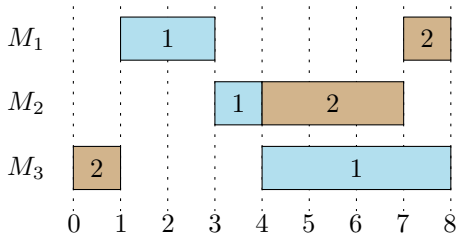


Figure B.1: Example of an optimal schedule for Example B.1, shown as a Gantt chart. Each row M_i corresponds to machine i and each block numbered j on this row represents the operation (i, j). The dashed lines indicate unit time steps. Note that machine 2 is kept idle, while operation $(2, 2)$ could have already been scheduled at time 1. Furthermore, for this particular instance, it can be checked that this is the unique optimal schedule.

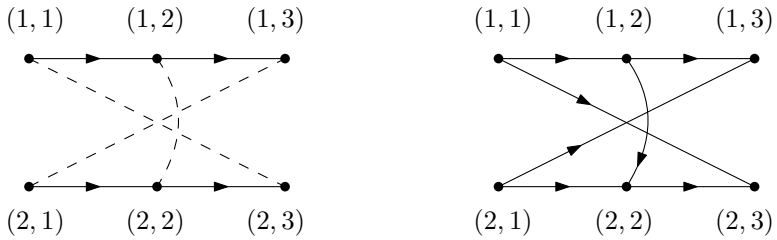


Figure B.2: Illustration of disjunctive graphs for Example B.1. Horizontal arrows represent conjunctive arcs. We used dashed lines to for the pairs of disjunctive arcs as dashed lines. The left graph corresponds to an empty selection $\mathcal{O} = \varnothing$ while the right graph shows the selection $\mathcal{O}$ that corresponds to the optimal schedule of Figure B.1.

on machine i before it is processed on machine k. When two distinct jobs j_1 and j_2 both require processing on the same machine i, we say that they are conflicting. The disjunctive arcs $\mathcal{D}$ are used to encode the possible choices of resolving such conflicts, by deciding which of j_1 or j_2 visits i first. More specifically, let j_1 and j_2 be conflicting on some machine i, then the nodes (i, j_1) and (i, j_2) are connected by two arcs in opposite directions.

The disjunctive graph can also be used to encode (partial) solutions as follows. It can be shown that each feasible solution corresponds to a selection $\mathcal{O}$ of exactly one disjunctive arc from each pair such that the induced graph $(\mathcal{N}, \mathcal{C} \cup \mathcal{O})$ is acyclic [43]. More precisely, consider two conflicting operations (i, j_1) and (i, j_2), then $\mathcal{O}$ contains either $(i, j_1) \to (i, j_2)$ or $(i, j_1) \to (i, j_2)$. To illustrate this, the empty and complete disjunctive graphs for the instance in Example B.1 are shown in Figure B.2.

Solution methods. Most job shop problems are very hard to solve. For example, the class of problems $Jm|r_j|C_{\max}$ considered in Example B.1 is known to be NP-hard [17], even without release dates, which is denoted $Jm||C_{\max}$. As a consequence, much effort has gone into developing good heurstics. A type of heuristic that is often considered is to apply a so-called *dispatching rule* in order to build a schedule in a step-by-step fashion. At each step, the rule chooses some job from all jobs with remaining unscheduled operations and schedules this next operation at the earliest time possible, given the current schedule.

A more principled way of solving job shop problems relies on the mathematical programming framework. We illustrate this for the problem $Jm|r_j|C_{\max}$ of Example B.1. Using the

notation of the disjunctive graph, the problem can be concisely stated as

$$
\begin{array}{rll}
\min\limits_{y} & J(y) & \\
\text{such that} & y(s_j, j) \leq r(j) & \text{for each job } j, \\
& y(i,j) + p(i,j) \leq y(r,k) & \text{for each conjunction } (i,j) \to (r,k) \in \mathcal{C}, \\
& \left.\begin{array}{l} y(i,j) + p(i,j) \leq y(i,k) \\ \qquad \text{or (not both)} \\ y(i,k) + p(i,k) \leq y(i,j) \end{array}\right\} & \text{for each disjunction } (i,j) \leftrightarrow (i,k) \in \mathcal{D}, \\
& y(i,j) \in \mathbb{R} & \text{for each operation } (i,j) \in \mathcal{N}.
\end{array}
$$

Note that this is almost an mixed-integer linear program (MILP). Let $M > 0$ be some sufficiently large number and introduce a binary decision variable $b_{(i,j)\leftrightarrow(i,k)} \in \{0,1\}$ for each pair of disjunctive arcs, then the pair of disjunctive constraint can be rewritten to

$$
\begin{aligned}
y(i,j) + p(i,j) &\leq y(i,k) + M b_{(i,j)\leftrightarrow(i,k)}, \\
y(i,k) + p(i,k) &\leq y(i,j) + M(1 - b_{(i,j)\leftrightarrow(i,k)}),
\end{aligned}
$$

which is generally referred to as the *big-M method*. The resulting MILP can be solved by any off-the-shelf solver.

Appendix C

Neural combinatorial optimization

This section introduces the idea of applying a Machine Learning (ML) perspective on Combinatorial Optimization (CO) problems, which has gained a lot of attention[1] recently. One of the key ideas in this line of research is to treat problem instances as data points and to use machine learning methods to approximately map them to corresponding optimal solutions [6].

Algorithm execution as MDPs. It is very natural to see the sequential decision-making process of any optimization algorithm in terms of the Markov Decision Process (MDP) framework, where the environment corresponds to the internal state of the algorithm. From this perspective, two main learning regimes can be distinguished. Methods like those based on the branch-and-bound framework are often computationally too expensive for practical purposes, so *learning to imitate* the decisions taken in these exact algorithms might provide us with fast approximations. In this approach, the ML model's performance is measured in terms of how similar the produced decisions are to the demonstrations provided by the expert. On the other hand, some problems do not even allow efficient exact methods, so it is interesting to study solution methods that *learn from experience*. An interesting feature of this direction is that it enables the algorithm to implicitly learn to exploit the hidden structure of the problems we want to solve.

Because neural networks are commonly used as encoder in these ML models for CO, we will refer to this new field as *Neural Combinatorial Optimization* (NCO). A wide range of classical combinatorial optimization problems has already been considered in this framework, so we briefly discuss the taxonomy used in the survey [40]. One distinguishing feature is whether existing off-the-shelf solvers are used or not. On the one hand, *principal* methods are based on a parameterized algorithm that is tuned to directly map instances to solutions, while *joint* methods integrate with existing off-the-shelf solvers in some way (see the survey [37] on integration with the branch-and-bound framework). An illustrative example of the latter category are the use of ML models for the branching heuristic or the selection of cutting planes in branch-and-cut algorithms [50]. The class of principal methods can be further divided into *construction* heuristics, which produce complete solutions by repeatedly extending partial solutions, and *improvement* heuristics, which aim at iteratively improving the current solution with some tunable search procedure.

Constraint satisfaction. A major challenge in NCO is constraint satisfaction. For example, solutions produced by neural construction policies need to satisfy the constraints of the original combinatorial problem. To this end, neural network components have been designed whose outputs satisfy some specific type of constraint, for example being a permutation of the input [54]. Constraints can also be enforced by the factorization of the mapping into repeated application of some policy. For example, in methods for the classical traveling salesman

[1] Pun not intended: a lot of recent works rely on neural attention architectures.

problem, a policy is defined that repeatedly selects the next node to visit. The constraint that nodes may only be visited once can be easily enforced by ignoring the visited nodes and taking the argmax among the model's probabilities for unvisited nodes.

Instead of enforcing constraints by developing some tailored model architecture, like construction and improvement heuristics, general methodologies have recently been explored for the problem of constraint satisfaction in neural networks. For example, the DC3 framework [12] employs two differentiable processes, completion and correction, to solve any violations of equality or inequality constraints, respectively. The more recent HardNet framework [42] uses a closed-form projection to map to feasible solutions under affine constraints and relies on a differentiable convex optimization solver (e.g., OptNet [3]) when general convex constraints are considered.

Neural job shop scheduling Various NCO methods have already been studied for the Job Shop Scheduling Problem (JSSP) with makespan objective, for which we now highlight some works that illustrate some of the above classes of methods. A lot of the policies used in these works rely on some graph neural network architecture, which is why the survey [45] provides an overview based on this distinguishing feature.

Dispatching rules. A very natural approach to model JSSP in terms of an MDP is taken in [51], where a dispatching heuristic is defined in an environment based on discrete scheduling time steps. Every available job corresponds to a valid action and there is a so-called No-Op action to skip to the next time step. States are encoded by some manually designed features. They consider the makespan objective by proposing a dense reward based on how much idle time is introduced compared to the processing time of the job that is dispatched. In some situation, some action can be proved to be always optimal ("non-final prioritization"), in which case the policy is forced to take this action. Additionally, the authors design some rules for when the No-Op action is not allowed in order to prevent unnecessary idling of machines. The proposed method is evaluated on the widely used Taillard [47] and Demirkol [10] benchmarks, for which performance is compared to static dispatching rules and a constraint programming (CP) solver, which is considered cutting-edge.

From a scheduling theory perspective [43], it can be shown that optimal schedules are completely characterized by the order of operations for regular objectives (non-decreasing functions of the completion times). The start times are computed from this order by a so-called *placement rule*, so considering discrete time steps introduces unnecessary model redundancy.

The seminal "Learning to Dispatch" (L2D) paper [55] proposes a construction heuristic for JSSP with makespan objective. Their method is based on a dispatching policy that is parameterized in terms of a graph neural network encoding of the disjunctive graph belonging to a partial solution. Again, each action corresponds to choosing for which job the next operation is dispatched. The rewards are based on how much the lower bound on the makespan changes between successive states. They use a Graph Isomorphism Network (GIN) architecture to parameterize both an actor and critic, which are trained using the Proximal Policy Optimization (PPO) algorithm. Using the Taillard and Demirkol benchmarks, they show that their model is able to generalize well to larger instances. As we already alluded to above, this way of modeling the environment is better suited to JSSP with regular objectives, because it does not explicitly determine starting times. They use a dispatching mechanism based on finding the earliest starting time of a job, even before already scheduled jobs, see their Figure 2. By doing this, they introduce symmetry in the environment: after operations O_{11}, O_{21}, O_{31} have been scheduled, both action sequences O_{22}, O_{32} and O_{32}, O_{22} lead to exactly the same state S_5 shown in their Figure 2. In this particular example, this means that it is impossible to have $O_{11} \to O_{22} \to O_{32}$. In general, it is not clear whether the resulting restricted policy is still sufficiently powerful, in the sense that an optimal operation order can always be constructed.

Guided local search. Recently, the authors of L2D investigated an improvement heuristic for JSSP [56] with makespan objective. This method is based on selecting a solution within

the well-known N_5 neighborhood, which has been used in previous local search heuristics. It is still not clear whether their resulting policy is complete, in the sense that any operation order can be achieved by a sequence of neighborhood moves. The reward is defined in terms of how much the solution improves relative to the best solution seen so far (the "incumbent" solution). The policy is parameterized using a GIN architecture designed to capture the topological ordering of operations encoded in the disjunctive graph of solutions. They propose a custom n-step variant of the REINFORCE algorithm in order to deal with the sparse reward signal and long trajectories. To compute the starting times based on the operation order, they propose a dynamic programming algorithm, in terms of a message-passing scheme, as a more efficient alternative to the classical recursive critical path method. Our proposal for efficiently updating the current starting time lower bounds in partial solutions can also be understood as a similar message-passing scheme, but where only some messages are necessary.

Joint method. An example of a joint method is given in [52], where the environment is stated in terms of a Constraint Programming (CP) formulation. This allows the method to be trained using demonstration from an off-the-shelf CP solver.

Appendix D

Reinforcement learning

For machine learning problems where data-collection is restricted in some way, the supervised learning paradigm, i.e., learning from labeled examples, is sometimes no longer appropriate or feasible. Very generally, the reinforcement learning paradigm can viewed as a generalization of supervised learning in which the data collection and selection process is not fixed anymore. The classical perspective is that of an *agent* that tries to maximize some cumulative *rewward* signal when interacting in some *environment*, which is formalized by the Markov Decision Process (MDP) model. We refer the reader to [46] for the commonly cited textbook introduction to RL from this perspective.

Problem definition. Consider finite sets of states $\mathcal{S}$ and actions $\mathcal{A}$. Given some current state s, the agent sends some action a to the environment, upon which it responds by providing some scalar reward signal r and transitions to state s', which happens with probablity $p(s', r|s, a)$. By fixing a policy π, which is a function $\pi(a|s)$ that gives the probablity of the agent choosing action a in state s, we obtain the induced *state Markov chain* with transition probabilities

$$\Pr(s \to s') = \sum_a \sum_r \pi(a|s) p(s', r|s, a).$$

Given some initial state distribution $h(s)$, we sample $S_0 \sim h(s)$ and use $S_0, S_1, S_2, \ldots$ to denote some sample trajectory. Moreover, we can also consider a more fine-grained Markov chain by considering the sequence of states, actions and rewards

$$S_0, A_1, R_1, S_1, A_2, R_2, S_2, \ldots,$$

in which which the state Markov chain is naturally embedded. Such a sample trajectory is also referred to as an *episode*. Let the corresponding *return* at step t be defined as

$$G_t = \sum_{k=t+1}^{\infty} R_k.$$

By marking a subset of states as being *final states*, we can consider finite episodes

$$S_0, A_1, R_1, S_1, A_2, R_2, S_2, \ldots S_N,$$

by using the convention that final states return zero reward and transition to themselves almost surely. For finite episodes, the goal is to find a policy π that maximizes the expected return $\mathbb{E}[G_0]$.

Solution methods. Most classical methods to find such an optimal policy π can be categorized as either being value-based or policy-based. Value-based can be generally understood as producing some estimate $v(s)$ for the expected return $\mathbb{E}[G_0|S_0 = s]$. The optimal policy is then parameterized in terms of these estimates $v(s)$. In contrast, policy-based methods use a

more direct parameterization of the policy space and often rely on some kind of gradient-based optimization. Specifically, let π_θ be some policy with parameters θ, then we aim to apply the gradient descent updating

$$\theta \leftarrow \theta - \alpha \nabla \mathbb{E}[G_0]$$

where α is referred to as the learning rate. However, in almost all interesting situations, it is infeasible to compute the gradient directly.

Induced Markov chain. For some fixed policy π and initial state distribution h, we consider the underlying *induced Markov chain* over states. Because we are working with finite episodes, the induced state process is a Markov chain with absorbing states. We want to analyze how often states are visited on average, over multiple episodes. To better understand what *on average* means here, imagine that we link together separate episodes to create a regular Markov chain without absorbing states, in the following way: from each final state, we introduce state transitions to the initial states according to distribution h, see also Figure D.1. Furthermore, we will write $S_t^{(i)}$ to denote the state at step t of episode i.

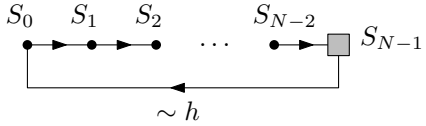


Figure D.1: Illustration of the induced Markov chain when dealing with finite episodes. The next state after the final state, indicated as the grey rectangle, is sampled according to initial state distribution h.

D.1 Stationary distribution for finite episodes.

Consider an absorbing Markov chain with transition matrix

$$P_{xy} = \sum_a \pi(a|x) p(y|x,a).$$

There are t transient states and r absorbing states, so P can be written as

$$P = \begin{pmatrix} Q & R \\ \mathbf{0} & I_r \end{pmatrix},$$

where Q is a t-by-t matrix, R is a nonzero t-by-r matrix, I_r is the r-by-r identify matrix and $\mathbf{0}$ is the zero matrix. Observe that $(Q^k)_{xs}$ is the probability of reaching state s in k steps without being absorbed, starting from state x. Hence, the expected number of visits to state s without being absorbed, starting from state x, is given by

$$\eta(s|x) := \sum_{k=0}^{\infty} (Q^k)_{xs}.$$

Writing this in matrix form $N_{xs} = \eta(s|x)$, we can use the following property of this so-called Neumann series, to obtain

$$N = \sum_{k=0}^{\infty} Q^k = (I_t - Q)^{-1}.$$

Now we can derive two equivalent equations

$$N = (I_t - Q)^{-1} \iff \begin{cases} N(I_t - Q) = I_t \iff N = I_t + NQ, & \text{or} \\ (I_t - Q)N = I_t \iff N = I_t + QN. \end{cases}$$

Expanding the first equation in terms of matrix entries $N_{xs} = \eta(s|x)$ gives

$$\begin{aligned}\eta(s|x) &= \mathbb{1}\{x = s\} + \sum_y \eta(y|x) Q_{ys} \\ &= \mathbb{1}\{x = s\} + \sum_y \eta(y|x) \sum_a \pi(a|y) p(y|x, a)\end{aligned}$$

and similarly, the second equation gives

$$\begin{aligned}\eta(s|x) &= \mathbb{1}\{x = s\} + \sum_y Q_{xy} \eta(s|y) \\ &= \mathbb{1}\{x = s\} + \sum_a \pi(a|x) \sum_y p(y|x, a) \eta(s|y)\end{aligned}$$

Now since the initial state is chosen according to distribution h, the expected number of visits $\eta(s)$ to state s in some episode is given by

$$\eta(s) = \sum_x h(x) \eta(s|x),$$

or written in matrix form $\eta = hN$, where η and h are row vectors. Therefore, we can also work with the equations

$$\begin{cases} hN = h + hNQ, & \text{or} \\ hN = h + hQN, \end{cases}$$

which are generally called *balance equations*. By writing the first variant as $\eta = h + \eta Q$ and expanding the matrix multiplication, we obtain

$$\eta(s) = h(s) + \sum_y \eta(y) \sum_a \pi(a|y) p(s|y, a).$$

Through appropriate normalization of the expected number of visits, we obtain the average fraction of time spent in state s, given by

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}.$$

Sampling. Suppose we have some function $f : \mathcal{S} \to \mathbb{R}$ over states and we are interested in estimating $\mathbb{E}_{S_t^{(i)} \sim \mu}[f(S_t^{(i)})]$. We can just take random samples of $S_t^{(i)}$, by sampling initial state $S_0^{(i)} \sim h$ and then *rolling out* π to obtain

$$\tau^{(i)} = (S_0^{(i)}, A_0^{(i)}, R_1^{(i)}, S_1^{(i)}, A_1^{(i)}, R_2^{(i)}, S_2^{(i)}, \ldots, S_{N^{(i)}-1}^{(i)}) \sim \pi(\tau^{(i)} | S_0^{(i)}),$$

where $N^{(i)}$ denotes the total number of states visited in this episode. Given M such episode samples, we compute the estimate as

$$\mathbb{E}_{S_t^{(i)} \sim \mu}[f(S_t^{(i)})] \approx \left(\sum_{i=1}^{M} \sum_{t=0}^{N^{(i)}-1} f(S_t^{(i)}) \right) / \left(\sum_{i=1}^{M} N^{(i)} \right).$$

Observe that the analysis of the induced Markov chain can be extended to explicitly include actions and rewards as part of the state and derive the stationary distribution of the resulting Markov chain. However, we do not need this distribution explicitly in practice, because we can again use episode samples $\tau^{(i)}$. To keep notation concise, we will from now on denote this type of expectation as $\mathbb{E}_{\tau \sim h, \pi}[f(\tau)]$ and omit episode superscripts. Using this new notation, note that the average episode length is given by

$$\mathbb{E}_{h,\pi}[N] = \sum_{s'} \eta(s').$$

D.2 Policy gradient estimation

Let $v_{\pi_\theta} = \mathbb{E}_{h,\pi_\theta}[G_0]$ denote the expected episodic reward under policy π, where G_t is called the reward-to-go at step t, which is defined as

$$G_t := \sum_{k=t+1}^{\infty} R_k.$$

The main idea of policy gradient methods is to update the policy parameters θ in the direction that increases the expected episodic reward the most. This means that the policy parameters are updated as

$$\theta_{k+1} = \theta_k + \alpha \nabla v_{\pi_\theta},$$

where α is the learning rate and the gradient is with respect to θ. Instead of trying to derive or compute the gradient exactly, we often use some statistical estimate based on sampled episode. The basic policy gradient algorithm is to repeat the three steps

1. `sample` M `episodes` $\tau^{(1)}, \ldots, \tau^{(M)}$ `following` π_θ,
2. `compute gradient estimate` $\widehat{\nabla v_{\pi_\theta}}(\tau^{(1)}, \ldots, \tau^{(M)})$,
3. `update` $\theta \leftarrow \theta + \alpha \widehat{\nabla v_{\pi_\theta}}$.

REINFORCE estimator. We will now present the fundamental policy gradient theorem, which essentially provides a function f such that

$$\nabla v_{\pi_\theta} = \mathbb{E}_{\tau \sim h,\pi_\theta}[f(\tau)],$$

which allows us to estimate the policy gradient using episode samples. To align with the notation of [46], we write $\Pr(x \to s, k, \pi) := (Q^k)_{xs}$, for the probability of reaching state s in k steps under policy π, starting from state some x, so that the expected number of visits can also be written as

$$\eta(s) = \sum_x h(x) \sum_{k=0}^{\infty} \Pr(x \to s, k, \pi)$$

As proven in the chapter on policy gradient methods in [46], the gradient of the value function for a fixed initial state s_0 with respect to the parameters is given by

$$\nabla v_\pi(s_0) = \sum_s \sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \sum_a q_\pi(s,a) \nabla \pi(a|s). \tag{D.1}$$

When choosing the initial state s_0 according to some distribution $h(s_0)$, we verify that the final result is still the same as in [46]:

$$\begin{aligned}
\nabla v_\pi :=& \nabla \mathbb{E}_{s_0 \sim h}[v_\pi(s_0)] && \text{(D.2a)}\\
=& \sum_{s_0} h(s_0) \sum_s \sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \sum_a q_\pi(s,a) \nabla \pi(a|s) && \text{(D.2b)}\\
=& \sum_s \eta(s) \sum_a q_\pi(s,a) \nabla \pi(a|s) && \text{(D.2c)}\\
=& \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s) && \text{(D.2d)}\\
\propto& \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s), && \text{(D.2e)}
\end{aligned}$$

where the constant of proportionality is just the average episode length. Because we do not know μ or q_π explicitly, we would like to estimate ∇v_π based on samples. If we sample

episodes according to h and π as explained above, we encounter states according to μ, so we have

$$\nabla v_\pi \propto \mathbb{E}_{h,\pi}\left[\sum_a q_\pi(S_t,a)\nabla\pi(a|S_t)\right] \tag{D.3a}$$

$$= \mathbb{E}_{h,\pi}\left[\sum_a \pi(a|S_t)q_\pi(S_t,a)\frac{\nabla\pi(a|S_t)}{\pi(a|S_t)}\right] \tag{D.3b}$$

$$= \mathbb{E}_{h,\pi}\left[q_\pi(S_t,A_t)\frac{\nabla\pi(A_t|S_t)}{\pi(A_t|S_t)}\right] \tag{D.3c}$$

$$= \mathbb{E}_{h,\pi}\left[G_t\nabla\log\pi(A_t|S_t)\right]. \tag{D.3d}$$

Baseline. Let $b(s)$ be some function of the state s only, then we have for any $s\in\mathcal{S}$

$$\sum_a b(s)\nabla\pi(a|s) = b(s)\nabla\sum_a \pi(a|s) = b(s)\nabla 1 = 0. \tag{D.4}$$

This yields the so-called REINFORCE estimate with *baseline*

$$\nabla v_\pi \propto \sum_s \mu(s)\sum_a (q_\pi(s,a)+b(s))\nabla\pi(a|s) \tag{D.5a}$$

$$= \mathbb{E}_{h,\pi}\left[\big(q_\pi(S_t,A_t)+b(S_t)\big)\nabla\log\pi(A_t|S_t)\right] \tag{D.5b}$$

$$= \mathbb{E}_{h,\pi}\left[\big(G_t+b(S_t)\big)\nabla\log\pi(A_t|S_t)\right]. \tag{D.5c}$$

Although estimates (D.3d) and (D.5c) are both equivalent in terms of their expected value, they may differ in higher moments, which is why an appropriate choice of b can make a lot of difference in how well the policy gradient algorithm converges to an optimal policy. As a specific baseline, consider the expected cumulative sum of rewards up to step the current step t, defined as

$$b(s) = \mathbb{E}_{h,\pi}\left[\sum_{k=1}^{t} R_k \Big| S_t = s\right], \tag{D.6}$$

then observe that

$$q_\pi(s,a)+b(s) = \mathbb{E}_{h,\pi}\left[\sum_{k=t+1}^{\infty} R_k\Big|S_t=s,A_t=a\right] + \mathbb{E}_{h,\pi}\left[\sum_{k=1}^{t} R_k\Big|S_t=s\right] \tag{D.7a}$$

$$= \mathbb{E}_{h,\pi}\left[\sum_{k=1}^{\infty} R_k\Big|S_t=s,A_t=a\right] \tag{D.7b}$$

$$= \mathbb{E}_{h,\pi}[G_0|S_t=s,A_t=a], \tag{D.7c}$$

which is just the expected total episodic reward. Now define function f to be

$$f(s,a) := (q_\pi(s,a)+b(s))\nabla\log\pi(a|s) = \mathbb{E}_{h,\pi}\left[G_0|S_t=s,A_t=a\right]\nabla\log\pi(a|s) \tag{D.8a}$$

$$= \mathbb{E}_{h,\pi}\left[G_0\nabla\log\pi(a|s)|S_t=s,A_t=a\right], \tag{D.8b}$$

then applying the law of total expectation yields

$$\nabla v_\pi \propto \mathbb{E}_{h,\pi}[f(S_t,A_t)] = \mathbb{E}_{h,\pi}\left[G_0\nabla\log\pi(A_t|S_t)\right]. \tag{D.9}$$

Appendix E

Miscellaneous

Lemma 3.1. *Let $\mathcal{G}$ be some Directed Acyclic Graph (DAG) over nodes V, then there exists some $v \in V$ that has no incoming arcs, which is called* minimal. *Moreover, the nodes V can be arranged in a sequence $v_1, v_2, \ldots, v_{|V|}$ such that if $\mathcal{G}$ contains an arc $v_i \to v_j$ then $i < j$. Such a sequence is called a* topological order.

Proof. For sake of contradiction, suppose there is no such minimal node, so every $v \in V$ has an incoming arc. Pick some arbitrary $v_0 \in V$, then there must exist $v_1 \in V$ such that $v_1 \to v_0$. Again, v_1 must have an incoming, so we can pick $v_2 \in V$ such that $v_2 \to v_1$. We can continue picking such predecessor as long as we want, obtaining a sequence $v_0, v_1, \ldots, v_n$. Since there are only finitely many nodes, if we take the length n of this sequence large enough, we must eventually pick a node twice, say $v_k = v_m$ for some $m < k \leq n$, but then we have $v_k \to v_{k-1} \to \cdots \to v_m = v_k$, which shows $\mathcal{G}$ has a cycle.

To show that $\mathcal{G}$ has a topological order, consider the following procedure. Starting with $\mathcal{G}_0 := \mathcal{G}$ we select some minimal node v_1. We remove v_1 and all its outgoing edges from the graph to obtain a new graph $\mathcal{G}_1$, which is still a DAG, so there must be some minimal node v_2. We can repeat this procedure until $\mathcal{G}_N$ is an empty graph to obtain a sequence $v_1, \ldots, v_N$. Suppose $v_i \to v_j$ in $\mathcal{G}$, then v_i must appear earlier in the sequence than v_j, because otherwise v_i was not a minimal element at the time it was picked. □

Lemma 3.5. *Let $\mathcal{O}$ be complete and such that $G(\mathcal{O})$ is acyclic, with unique vehicle order π as given by Lemma* **??**. *Suppose that $\sigma > \rho > 0$, then active schedule $y(\mathcal{O}) = \{y_i : i \in \mathcal{N}\}$ is uniquely defined by $y_{\pi_1} = a_{\pi_1}$ and through the recursion*

$$y_j = \max\{a_j, y_i + w(i,j)\}, \tag{3.15}$$

for every pair $i = \pi_t$ and $j = \pi_{t+1}$ with $t = 1, \ldots, N-1$.

Proof. We prove that y is a feasible solution to (AS) by showing that it satisfies (3.10). First of all, since π_1 is a minimal node of $G(\mathcal{O})$, we have $\mathcal{N}^-(\pi_1) = \varnothing$, so feasibility condition (3.10) is trivially satisfied with equality by the definition $y_{\pi_1} = a_{\pi_1}$. We proceed inductively by showing that, for any further $i = \pi_t$ and $j = \pi_{t+1}$, we have

$$y_i + w(i,j) = \max_{v \in \mathcal{N}^-(j)} y_v + w(v,j) \tag{E.1}$$

so that the definition of y_j through (3.15) also satisfies (3.10) with equality. Hence, uniqueness of $y(\mathcal{O}) = y$ simply follows from the fact that y_j satisfies (3.10) with equality for every $j \in \mathcal{N}$.

For the inductive step, we show $y_i + w(i,j) \geq y_v + w(v,j)$ for any $v \in \mathcal{N}^-(j) \setminus \{i\}$ for the cases illustrated in Figure E.1. Suppose i and j are connected via a disjunctive arc $(i,j) \in \mathcal{C}$, then any other in-neighbor $v \in \mathcal{N}^-(j) \setminus \{i\}$ must belong to a different route, so that $(v,j) \in \mathcal{O}$. Because i and j are on the same route and i is the immediate predecessor of j in the topological order, we must also have $(v,i) \in \mathcal{O}$. Therefore, we have

$$y_i + w(i,j) = y_i + \rho \geq y_v + \sigma + \rho > y_v + \sigma = y_v + w(v,j).$$

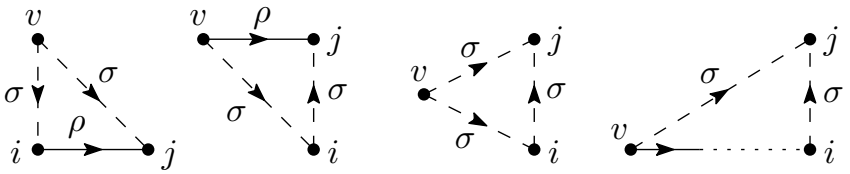


Figure E.1: Sketch of the four cases distinguished in the proof of Lemma 3.5. Arc weights are indicated by σ and ρ. Conjunctive arcs $\mathcal{C}$ are drawn with solid lines and disjunctive arcs $\mathcal{O}$ are drawn with dashed lines. The dotted line in the rightmost figure represent a chain of conjunctive arcs.

Otherwise, i and j are connected by a disjunctive arc $(i,j) \in \mathcal{O}$. Let $v \in \mathcal{N}^-(j) \setminus \{i\}$, then if v is on the same route as j, they are connected by a conjunctive arc $(v,j) \in \mathcal{C}$, so we must have $(v,i) \in \mathcal{O}$, again because i is the immediate predecessor of j. Hence, we have

$$y_i + w(i,j) = y_i + \sigma \geq y_v + 2\sigma > y_v + \rho = y_v + w(v,j).$$

If $(v,j) \in \mathcal{O}$ with $r(v) \neq r(i)$, then it follows that $(v,i) \in \mathcal{O}$, so that

$$y_i + w(i,j) = y_i + \sigma \geq y_v + 2\sigma > y_v + \sigma = y_v + w(v,j).$$

If $(v,j) \in \mathcal{O}$ with $r(v) = r(i)$, then there is a path of conjunctive arcs between v and i, from which it follows that $y_i \geq y_v + \rho$, so that

$$y_i + w(i,j) = y_i + \sigma \geq y_v + \sigma + \rho > y_v + w(v,j). \qquad \square$$

Lemma E.1. *Let $f : X \times Y \to \mathbb{R}$ be some continuous function. If Y is compact, then the function $g : X \to \mathbb{R}$, defined as $g(x) = \inf\{f(x,y) : y \in Y\}$, is also continuous.*

Lemma E.2. *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be continuous and $y \in \mathbb{R}^m$, then the level set $N := f^{-1}(\{y\})$ is a closed subset of $\mathbb{R}^n$.*

Proof. For any $y' \neq y$, there exists an open neighborhood $M(y')$ such that $y \notin M(y')$. The preimage $f^{-1}(M(y'))$ is open by continuity. Therefore, the complement $N^c = \{x : f(x) \neq y\} = \cup_{y' \neq y} f^{-1}(\{y'\}) = \cup_{y' \neq y} f^{-1}(M(y'))$ is open. $\square$

The following definition might be helpful in deriving the buffer constraint....

Acceleration boundary. Before we present the decomposition, we first define an auxiliary upper boundary. Similar to how we generalized the entry boundary $\check{x}$ to the deceleration boundary in Section 5.2.3, we now generalize the exit boundary $\hat{x}$ to obtain the *acceleration boundary*. Because the derivation is completely analogous, we will only present the resulting expressions. Let $x \in \mathcal{D}[a,b]$ be some smooth trajectory, then the acceleration boundary $x^+[\xi]$ of x at some $\xi \in [a,b]$ is defined as the right-hand side of the inequality

$$x(t) \leq x(\xi) + \int_\xi^t \{\dot{x}(\xi) + \bar{\omega}(\tau - \xi)\}_{[0,1]} \,\mathrm{d}\tau =: x^+[\xi](t), \tag{E.2}$$

which holds for every $t \in [a,b]$. Observe that the exit boundary can now be written as the restricted acceleration boundary $\hat{x} = (x^+[b])|_{[a,b]}$. Similar to definition (5.14), we define

$$x^+[p,v,\xi](t) := p + \int_\xi^t \{v + \bar{\omega}(\tau - \xi)\}_{[0,1]} \,\mathrm{d}\tau, \tag{E.3}$$

such that $x^+[\xi](t) = x^+[x(\xi), \dot{x}(\xi), \xi](t)$ and similar to (5.15), we calculate

$$x^+[p,v,\xi](t) = p + \begin{cases} \dots & \text{for } t \leq \bar{\delta}_0, \\ \dots & \text{for } t \in [\bar{\delta}_0, \bar{\delta}_1], \\ \dots & \text{for } t \geq \bar{\delta}_1, \end{cases} \tag{E.4}$$

with $\bar{\delta}_0 :=$ and $\bar{\delta}_1 :=$.

Recall the definition of $\hat{x}$ in equation (5.9). By carefully handling the $\max\{\cdot\}$, we can expand this expression as

$$\hat{x}(t) = \begin{cases} B - b + t + \bar{\omega}(b-t)^2/2 & \text{for } t \geq b - 1/\bar{\omega}, \\ B - 1/(2\bar{\omega}) & \text{for } t \leq b - 1/\bar{\omega}. \end{cases} \tag{E.5}$$