Efficient and Provably Safe Coordinated Intersection Crossing of Autonomous Vehicles

Opportunities of machine learning for combinatorial problems in network-wide motion planning

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

Contents

Chapter 1

Introduction

1.1 Context and motivation

The rapid advances in autonomous driving and wireless communication technologies present unprecedented opportunities to transform modern road traffic systems [51]. One promising direction involves the *coordination among groups of vehicles*, referring to the dynamic management of interactions both among vehicles and between vehicles and infrastructure, to coordinate behaviors such as speed, routing, and maneuvers in ways that promote safety, efficiency, and smooth traffic flow [46].

Autonomous intersections. To illustrate this concept, consider an intersection where only autonomous vehicles are permitted. Instead of relying on conventional traffic lights to regulate the safe passage of vehicles, an intelligent coordination system could enable vehicles to communicate both with one another and with the surrounding infrastructure to determine crossing sequences well in advance [12, 19, 33]. Such anticipatory coordination would allow vehicles to continuously adjust their speeds, thereby minimizing the need for complete stops at the intersection. Experiments in controlled environments have already demonstrated the feasibility of this idea [29, 34].

The relevance of this approach becomes even more pronounced when considering networks of interconnected intersections. In such systems, the optimization of one intersection cannot be treated in isolation, as traffic flow at any given junction directly influences neighboring ones. Coordinated strategies could extend across the entire network, allowing vehicles to plan optimal trajectories over multiple intersections.

Research landscape The concept of network-wide coordination offers a rich terrain for research and technological innovation [51, 76] as well as policy development, involving questions on how to adopt [50] autonomous vehicle for safe [25] and accessible [15] transportation systems. The current literature reflects this breadth through diverse modeling choices and conflicting requirements that raise challenging trade-offs and design questions. We highlight four active research themes:

- *Performance objectives and user preferences.* From the perspective of promoting traffic throughput, minimizing travel delay is a very natural objective. However, this conflicts with the goals of limiting fuel consumption and promoting passenger comfort; there is growing interest in eco-driving approaches [21]. In multi-user systems, fairness and economic implications are also central—for example, some works [55, 58] propose auction-based systems where users bid for priority service at intersections.
- *Decentralized control and communication.* A key design issue is how to organize communication and control in the systems. Centralized coordination with full observability is nearly impossible in practice due to noisy sensors, communication delays [75], data loss or inaccurate information [73], and scalability challenges. Most proposals assume decentralized architectures [76], where low-level control is handled by individual vehicles while a central entity focuses on higher-level coordination.

- *Graceful adoption of autonomous driving.* Fully autonomous traffic without human drivers is not expected in the foreseeable future. Consequently, significant effort has gone into studying mixed traffic systems [39]. Interestingly, even low penetration rates can bring large efficiency benefits, as autonomous vehicles can influence human driver behavior [74], thereby reducing congestion.
- *Robustness to uncertainty.* Uncertainty and unobservability are fundamental in all formulations of coordination. Major sources include fluctuating mobility demand throughout the day, noisy sensors, partial observability, and the need to handle dangerous situations [77], mechanical failures, software faults, cyberattacks, and dynamic prioritization of emergency vehicles.

1.2 Problem scope and model assumptions

While acknowledging the breadth of research on autonomous intersection coordination, we focus on a simplified formulation that exposes the fundamental challenges of large-scale coordination, studying an idealized traffic model that captures only the essential components.

At the heart of any traffic coordination system lies a fundamental tension: safety requirements and efficiency objectives are inherently in conflict, yet both are essential goals of coordination. Safety demands maintaining sufficient separation between vehicles and avoiding simultaneous occupation of conflict zones. Efficiency requires minimizing delays and maximizing throughput, which naturally pushes toward tighter spacing and more aggressive intersection usage.

This conflict represents the irreducible core of the coordination problem: every system must resolve this tension. Traditional traffic lights resolve it conservatively, sacrificing efficiency for guaranteed safety. Human drivers resolve it through continuous negotiation and risk assessment. Autonomous coordination offers the potential for a more sophisticated resolution of this trade-off, making it a promising technology.

Safe and efficient motion planning. We formulate the coordination problem as a trajectory-optimization problem subject to collision-avoidance constraints. Since our focus is on high-level coordination, we assume that low-level steering and control are handled such that vehicles can perfectly follow planned trajectories. The coordination task is thus to steer the traffic system from an initial to a desired configuration as efficiently as possible, subject to safety constraints.

To better understand the inherent difficulties of coordination, we consider an idealized setting with centralized control, no human drivers, and perfect information. The problem is formulated in an *offline* setting: all vehicles, their routes, and their objectives are known in advance, and we do not consider random arrivals, reoptimization, or rolling-horizon control. We refer to this setting as the *safe and efficient motion planning problem.*

Motivation for idealized setting. This abstraction sacrifices realism for clarity, isolating the computational core of the coordination problem and enabling concrete, interpretable results. We make this choice for two reasons:

First, even in this idealized setting, the problem exhibits interesting computational challenges arising from the inherent conflict between safety and efficiency. Any method that succeeds here provides an upper bound on what can be achieved in real systems with decentralized control, communication delays, noisy information, and mixed traffic.

Second, our idealized model fits certain controlled environments where many complications of urban traffic can be avoided. Autonomous vehicles are already being deployed in confined sites such as ports, mines, and quarries [35], as well as warehouses, manufacturing facilities, and indoor farming installations. In these settings, assumptions of full autonomy and centralized control are reasonable, making our methods immediately applicable.

1.3 Challenges and opportunities

Safe trajectory optimization. Trajectory optimization is infinite-dimensional: we are looking for a set of functions describing vehicle motion over time. Moreover, the potentially severe consequences make collision avoidance a *hard constraint*, distinguishing traffic coordination from optimization problems where constraints can be relaxed or penalized.

A common approach is *direct transcription*—discretizing time to obtain a finite-dimensional program—for which hard constraint satisfaction is straightforward to achieve. However, this method becomes computationally expensive as the number of vehicles, intersections, and time steps grows. When vehicle dynamics are simple and explicitly known, analytical solutions can sometimes be derived, but enforcing hard safety constraints makes this notoriously difficult. For more complex or uncertain dynamics, learning-based approaches can be used to approximate the dynamics or control policies, but ensuring safety and interpretability remains a major challenge.

Crossing order optimization. Trajectory optimization is inherently non-convex because collision-avoidance constraints impose an implicit vehicle crossing decision at intersections. Rather than relying on fixed traffic signal phases, these ordering decisions can, in principle, be optimized. Even at a single intersection, the number of feasible crossing sequences grows quickly with the number of vehicles. In a network, decisions at one intersection directly influence arrivals at others, coupling the problem across the system. This coupling, combined with the combinatorial growth of possible sequences in larger networks, makes exact optimization intractable. Real-time requirements further constrain computation, motivating the development of fast heuristic methods.

Joint combinatorial-continuous optimization. The combinatorial aspect of trajectory optimization cannot be considered in isolation, since the feasibility and cost of any crossing sequence depend non-trivially on the continuous-time dynamics and interactions of the vehicles. Simultaneously optimizing both the combinatorial and continuous components—what we refer to as *joint optimization*—poses major algorithmic challenges. While heuristics have been proposed in the literature, often assuming decentralized control and communication, approaches that optimize both levels in a unified framework, with clearly identifiable trade-offs, remain relatively scarce.

Learning from problem structure. The coordination problems discussed above exhibit regularities that can potentially be exploited algorithmically. Traditional approaches rely on manually designed heuristics to capture such structure, but recent advances in machine learning for combinatorial optimization suggest an alternative: treating problem instances as data and learning to recognize patterns associated with good solutions.

A key insight from this literature is that practical applications rarely require optimal solutions for all conceivable instances. Instead, problem instances are typically drawn from a limited distribution with predictable characteristics—such as recurring traffic patterns, common vehicle routes, or structured networks. By focusing on expected performance over these distributions, rather than worst-case guarantees, heuristics can be learned that are both efficient and effective.

1.4 Related work

The survey [46] classifies works mainly on the level of organization. Moreover, apart from vehicle motion around intersections and other conflict spaces, they also review 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 [33], where works are classified, first on level of organization, but they also consider vehicle dynamics, conflict detection and scheduling policy.

The general goal of coordination has been approached from a wide range of perspectives, which has given rise to diverse concrete problem formulations and methodologies. The level of organization at which coordination takes place is a very typical distinguishing feature among existing works [46]. A well-known example of *local coordination* is platooning of vehicles, in which consecutive vehicles on the same lane try to keep close together while maintaining similar speeds, with the aim of lowering energy consumption by reducing aerodynamic resistance. It has been shown that platooning can also result in a more efficient use of intersections [48, 62, 69]. On a larger scale, *global coordination* methods like dynamic route optimization have been proposed to reduce travel delay for all vehicles in the network [56].

Apart from the level of organization, coordination problems can have many more modeling aspects. For example, one may think of heterogeneous vehicles—in terms of physical differences or functionality—different models of centralized/decentralized communication between vehicles or with the infrastructure, under different guarantees on reliability; complex road topology, curved lanes affecting the maximum safe speed, merging lanes; implications of mixing human traffic with fully autonomous traffic.

1.4.1 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 [12, 13] 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 [3]. Other works consider speed profile recommendations and more fine-grained prioritization of requests by the intersection controller [26]. The model has also been extended to networks of intersections, where dynamic route optimization has been considered as well [24]. Later works propose more precise methods for conflict detection than the original cell-based approach [38, 41]. Finally, communication delays play a major role in practice, so a time-aware extension of AIM has been introduced in [34].

1.4.2 Optimizing the crossing order

In traffic coordination, combinatorial problems arise naturally from ordering constraints—deciding which vehicle should move first across shared resources such as intersections, merge areas, 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.

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 [38], 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 [41], but they formulate the reservation assignment problem as a non-linear optimization problem, which is solved using a tabu search heuristic.

The policy in [65] 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 platoons. All vehicles in a platoon are required to cross the intersection together, while maintaining feasibility with respect to safe trajectories. Next, crossing times are assigned to platoons while avoiding collisions. Based on this schedule, a local vehicular control method [64] is used that guarantees safety to reach the assigned crossing times.

The work [48] 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 [69].

Finally, we note that crossing order decisions become particularly interesting when vehicles with heterogeneous dimensions and dynamics are considered, which has been investigated in [31]. 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.

1.4.3 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 [28] 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 [80] 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.

Safe reinforcement learning. The decision variables in this kind of problem represent the individual vehicle's trajectories, which are functions of time. The study of optimizing over functions traces back to the calculus of variations, which has evolved into the modern theory of optimal control [42]. The latest manifestation of this line of research is found within the roots of the popular field of reinforcement learning, which grew out of the dynamic programming approach to optimal control [4, 6, 53, 61].

An alternative line of research explores *safe reinforcement learning* as a data-driven approach to motion planning under safety constraints. Instead of solving large-scale optimization problems explicitly, such methods learn control policies, while embedding safety

Ref.	Key Paradigm	Control Setting	Safety	Objective	Notes / Limitations
[12]	Reservation of grid cells for trajectory in intersection	Multiagent online	Non-collision	Delay minimization	Simplified vehicle model; global heuristic
[64, 65]	Hierarchical control	Multiagent; online feedback control	Failure recovery	Combination of travel delay and fuel consumption	Robust to communication failures
[38]	Mixed-integer linear programming	Centralized; rolling horizon optimization	Non-collision	Travel delay	Optimization of intersection entry speed
[28, 30]	Approximation algorithm based on bilevel decomposition	Centralized offline optimization	Non-collision	Integral over function of distance and acceleration	Single vehicle per lane
[80]	Bilevel decomposition with platoon-based scheduling	Centralized rolling horizon optimization	Non-collision	Travel delay	Optimization of intersection entry speed
[48]	Provable feasible trajectories under random arrivals	Centralized; reoptimization triggered by arrivals	Non-collision under uncertainty	Travel delay	Inspired by polling models; extensive proof of trajectory feasibility
[69]	Explicit solution of lower-level optimal control	Centralized online	Non-collision	Travel delay or minimal acceleration	No crossing order optimization; performance analysis using polling theory

Table 1.1: Comparison of approaches to autonomous intersection coordination.

considerations directly into the learning process—either through constrained optimization formulations, shielded policy updates, or model-based safety layers. These approaches offer the potential to scale to high-dimensional systems where classical trajectory optimization becomes intractable. However, guaranteeing strict safety remains an open challenge.

1.4.4 Machine learning for combinatorial optimization

A part of our work is inspired by recent developments in applying machine learning methods to combinatorial optimization problems. This emerging field seeks to leverage the pattern-recognition capabilities of machine learning to either improve existing algorithms or discover entirely new solution strategies.

Core methodology. Combinatorial optimization problems seek the best option from a finite set of discrete structures—such as routes through graphs, optimal subsets, or machine schedules. While theoretically we could evaluate every candidate solution, this brute-force approach quickly becomes infeasible. Traditional algorithms exploit problem structure to search more efficiently, using techniques like branch-and-bound for systematic exploration with provable guarantees, or local search heuristics based on intuitions about optimal solutions.

The machine learning perspective treats algorithm development itself as an optimization problem. Given some class of problem instances and a distribution P over these instances, the goal is to find an algorithm that performs well on average when instances are sampled from P. This contrasts with the classical worst-case analysis, which seeks algorithms that perform well on all possible instances.

Two main approaches have emerged in the literature, as surveyed by Bengio et al. [5] and Mazyavkina et al. [47]:

1. *Joint methods* replace expensive components of existing algorithms with fast learned approximations. For example, machine learning models have been used to approximate branching decisions in branch-and-bound [16] and to select cutting planes in integer programming [66]. Crucially, learned approximations need not compromise optimality guarantees if they produce valid decisions (e.g., valid cuts or feasible branches).

2. *Principal methods* use machine learning to explore the space of possible algorithms and discover new solution strategies. This can be understood as learning to construct solutions from scratch or as predicting optimal solutions given problem instances. Our methodology belongs to this class.

Markov decision processes as algorithmic models A key technical innovation in this literature is modeling algorithms as policies for Markov decision processes (MDPs). In this framework, states represent the internal state of the algorithm—including data structures, partial solutions, and relevant problem information. Actions correspond to algorithmic decisions (e.g., which variable to branch on, which node to visit next). State transitions capture

Reference	Problem focus	Contribution / Method
Bengio et al. [5]	General CO problems	Survey on using ML for CO, framing the learning problem
Mazyavkina et al. [47]	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. [72]	TSP	Pointer networks for sequence-based combinatorial problems
Kool et al. [36]	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. [66]	ILP	Attention-based model for cutting plane selection trained with RL
Graves et al. [18]	General computation	Neural turing machine as a differentiable model of universal computation

Table 1.2: Some key references on machine learning for combinatorial optimization.

how these decisions update the algorithm's internal state, and rewards measure solution quality or progress toward a solution.

This MDP perspective naturally connects to reinforcement learning methods, which have been successfully applied to various combinatorial problems. Notable examples include pointer networks for sequence problems like TSP [72], graph neural networks with reinforcement learning for general graph problems [9], and attention mechanisms trained with reinforcement learning for routing problems [36].

For our crossing time scheduling problem, this framework is natural: each state represents a partial schedule (which vehicles have been assigned crossing times), actions correspond to selecting which vehicle or route to schedule next, and rewards capture how the schedule quality evolves.

Relevance to intersection scheduling. Our crossing time scheduling problem shares structural similarities with classical job shop scheduling, where jobs (vehicles) must be processed by machines (intersections) in predetermined sequences. Recent work has successfully applied neural reinforcement learning methods to job shop problems, demonstrating that learned heuristics can compete with or outperform traditional approaches. These successes motivated our investigation of similar techniques for intersection coordination.

However, our problem has distinctive features: the natural occurrence of platoons (vehicles traveling close together), the network structure when multiple intersections are considered, and the constraint that vehicles must eventually cross at full speed. These characteristics require adaptations of existing methods, which we detail in Chapter 4.

The fundamental premise underlying our approach is that the distribution of problem instances encountered at any particular intersection exhibits learnable structure. By training on representative instances—whether generated from models or collected from real traffic data—we can potentially discover coordination strategies specifically tuned to local conditions, rather than relying on worst-case guarantees that may be overly conservative for the traffic patterns actually observed.

1.5 Research questions and outline

From the start of this project, the overarching goal has been to develop tractable optimization models and algorithms for the coordination of autonomous 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 our project can be summarized into three main categories:

(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.

Chapter 2

Motion planning in fully autonomous intersections

Given the controlled environment sketched in the introduction, this chapter will further detail our mathematical optimization approach to traffic management of autonomous vehicles.

Autonomous intersection model. In Section 2.1, we model a single isolated intersection, illustrated in Figure 2.1, in which vehicles, equipped with perfect communication and steering capabilities, operate under the supervision of a central traffic manager. The model is based on the essential trade-offs inherent to all transportation systems. Specifically, the model is general enough to capture the following three fundamental and conflicting objectives.

- *Guaranteeing safety* remains the primary design requirement in any transportation system, has highest priority and is almost non-negotiable in practice. We say *almost*, because real systems are too complex to strive for zero chance of failure. In mathematical models, there is no such excuse. In our model, safety is defined simply as the absence of vehicle collisions.
- *Minimization of delay* is one of the central measures of efficiency and service quality. It is directly related[1] to the *capacity* of the system—in terms of the maximum number of vehicles—and to the *throughput*—in terms of the number of completed journeys or trips per second.
- *Energy efficiency*, which contributes both to sustainability goals and to the reduction of economic operational costs, is a desirable secondary goal. It also indirectly contributes to passenger comfort, because energy minimization results in smooth trajectories with controlled acceleration and deceleration.

Trajectory optimization. The task of safely and efficiently coordinating the motion of autonomous vehicles through our single-intersection model, can be formulated mathematically as *trajectory optimization*, which is precisely defined in Section 2.1.4. Perhaps the most natural way of solving this kind of problems depends of discretization of the time domain. In Section 2.2, we will illustrate a widely applicable method known as *direct transcription*, which is relatively straightforward to implement and guarantees to find an optimal solution. The price we pay for the generality of this method is the fact that its running time scales very poorly, in the sense that finding an optimal solution becomes intractable relatively quickly, when considering larger numbers of vehicles in the system or making the time discretization finer. This motivates the development of more specialized algorithms that leverage the structure of the particular problem at hand.

[1] In stylized queueing models of traffic, the relationship between these quantities is given by Little's law.

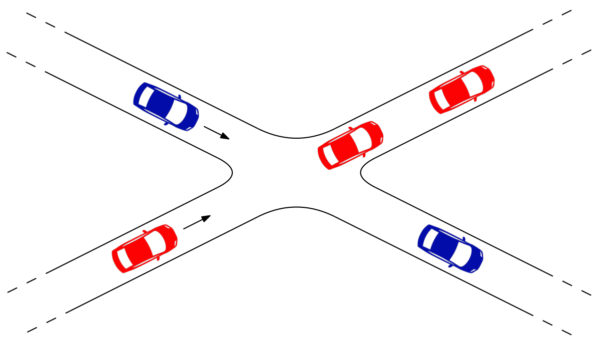


Figure 2.1: Stylized model of a single intersection with two conflicting flows of vehicles that are not allowed to turn at the intersection, so we can assume—without loss of generality—that vehicles cross the intersection in straight lines.

"Who goes first?" We conclude this chapter by discussing the most salient structural property of the trajectory optimization problem, which is a direct consequence of requirements on safety. Indeed, in order to avoid collision between vehicles from opposing lanes, we need to fix a particular *crossing order* among the vehicles in the system. This observation leads to the following natural reformulation of the problem. First, we allocate each vehicle to an *occupancy time slot*, being the time reserved for this vehicle to cross the intersection. Second, we try to assign each vehicle a trajectory satisfying its reservation.

It is natural to ask to what extent this decomposition simplifies the formulation by facilitating a two-step procedure: can we first optimize the crossing order as a separate stage, and then compute the corresponding vehicle trajectories afterwards? In general, the answer is negative. Nevertheless, the decomposition serves as a useful conceptual tool that can guide algorithm development, so we introduce the decomposition mathematically in Section 2.3.1 and we discuss one interesting example of an approximation algorithm from the literature in Section 2.3.2.

Delay minimization. Mathematicians are naturally inclined to seek the most general possible formulation. In Section 2.3.3, however, we deliberately focus on a more specific setting that captures the essential features of the problem, while enabling us to take full advantage of the decomposition. We make the following two assumptions:

- Vehicles drive, and remain driving, at *full speed* from the moment they first enter the intersection area.
- The optimization objective is stated purely in terms of *travel delay*; we do not take into account any further kind of property related to the smoothness or *shape* of the vehicles trajectories.

Under these assumption, the decomposition is proper. Consequently, the infinite-dimensional trajectory optimization problem reduces to a finite-dimensional *scheduling problem*, in terms of finding feasible sets of intersection occupancy time slots, while minimizing the total travel delay. Subsequent chapters are devoted to solving this reduced problem.

2.1 The single intersection model

To keep the model simple, 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, such 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 objective functions. The ingredients here will be used to formulate the trajectory optimization problem in the next section.

2.1.1 Collision-free vehicle positions

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 safe configurations of two vehicles, each on its own route.

Consider two routes intersecting at some angle α, as illustrated in Figure 2.2, 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 configurations (x_i, x_j) for which these two vehicles are not in a collision, in the sense that their corresponding translated open rigid bodies do not intersect. In general, the set of all safe configurations is defined as

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

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

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 2.2. 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, 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:

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

where $(a, b) := \{c \in \mathbb{R} : a < c < b\}$ denote the open interval and $\times$ is the cartesian product. 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 2.3. In the right-most example, where the intersections make a right angle $\alpha = \pi/2$, observe that there is actually equality in (2.2). To keep the presentation simple, we will make the following assumptions.

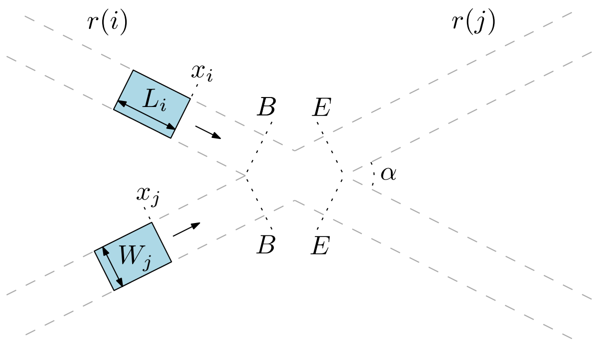


Figure 2.2: Example of two vehicle routes intersecting at some angle α. The indicated positions $x_i = B$ and $x_i = E$ for each vehicle i are chosen such that whenever the vehicle position x_i satisfies either $x_i \leq B$ or $x_i - L_i \geq E$, then the intersection area is not occupied by this vehicle at all and the other vehicle can take any position without causing a collision.

Assumption 2.1. 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 $\mathcal{E}^2 = \mathcal{E} \times \mathcal{E}$ provides a safe way to approximate all intersection-occupying vehicle positions.

Let us now proceed to considering arbitrary numbers of vehicles. We will use the following notation to identify routes and the vehicles associated with them.

Definition 2.1. Let the routes be identified by indices $\mathcal{R} := \{1, \dots, R\}$. 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

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

Given vehicle index $i = (r, k) \in \mathcal{N}$, we use the auxiliary notation $r(i) = r$ and $k(i) = k$. To identify the set of vehicles on route r, we use the shorthand notation

$$\mathcal{N}_r := \{i \in \mathcal{N} : r(i) = r\}. \tag{2.4}$$

Define the set of unordered *conflict* pairs

$$\mathcal{D} := \{\{i, j\} \in \mathcal{N} : r(i) \neq r(j)\}, \tag{2.5}$$

and the set of *consecutive* or *conjunctive* pairs

$$\mathcal{C} := \{(i, j) \in \mathcal{N} : r(i) = r(j) \text{ and } k(i) + 1 = k(j)\}. \tag{2.6}$$

Recall that $\mathcal{E}$ denotes the interval of intersection-occupying positions for each vehicle in the system. Therefore, imposing the *safe crossing constraints*

$$(x_i, x_j) \notin \mathcal{E}^2 \tag{2.7}$$

for each conflict $\{i, j\} \in \mathcal{D}$ guarantees the absense of collisions between vehicles from conflicting routes. Next, we prevent collisions between consecutive vehicles by further imposing the *safe headway constraints*

$$x_i - x_j \geq L, \tag{2.8}$$

for each conjunctive pair $(i, j) \in \mathcal{C}$. Obviously, 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.

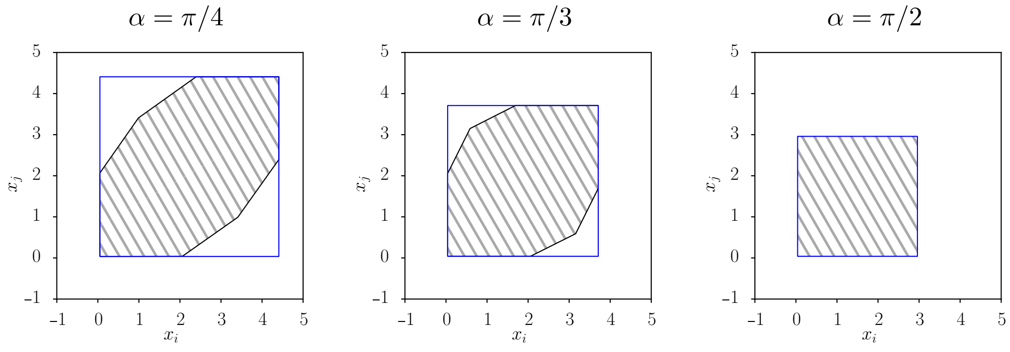


Figure 2.3: For three different intersection angles and fixed vehicle dimensions $W_i = W_j = 1$ and $L_i = L_j = 2$, we plotted the region $\mathcal{X}_{ij}^C$ in configuration space corresponding to collisions as the area marked in grey. The blue square shows the approximation of $\mathcal{X}_{ij}^C$ given by the left-hand side of (2.2). Because we assume a rectangular vehicle geometry, these figures are relatively straightforward to compute, which we briefly explain in Appendix A.

2.1.2 Motion dynamics

Now that we established the static features of the model, we will introduce the dynamical behavior of the system. In other words, we describe how the position of vehicles are allowed change as a function of time. Let $x_i(t)$ denote the position of vehicle i at time t and let $\dot{x}_i(t)$ and $\ddot{x}_i(t)$ denote its speed and acceleration, respectively. Consider the bounds

$$0 \leq \dot{x}_i(t) \leq \bar{v}, \tag{2.9a}$$

$$-\omega \leq \ddot{x}_i(t) \leq \bar{\omega}, \tag{2.9b}$$

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 (2.9).

Let the joint initial state of the system be denoted by

$$s := \{s_i = (x_i^0, v_i^0) : i \in \mathcal{N}\}.$$

The existence of trajectories satisfying (2.7),(2.8) and (2.9) generally depends on the initial state s of vehicles in some non-trivial manner. To give a rough idea, we derive some simple sufficient conditions. We exclude initial collisions at time $t = 0$ by requiring

$$x_i^0 - x_j^0 \geq L \quad \text{for all} \quad (i, j) \in \mathcal{C}. \tag{2.10}$$

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

$$x_{(r,1)}^0 \leq B - \bar{v}^2/(2\omega) - \bar{v}^2/(2\bar{\omega}), \tag{2.11}$$

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. From now, we will assume:

Assumption 2.2. The initial states $s = \{s_i = (x_i^0, v_i^0) : i \in \mathcal{N}\}$ satisfy (2.10) and (2.11).

2.1.3 Performance criteria

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

$$J_{\alpha,\beta}(x_i) = \int_0^{t_f} \alpha x_i(t) + \beta |\ddot{x}_i(t)| \, \mathrm{d}t. \tag{2.12}$$

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. In some sense, minimizing $J_{\alpha,\beta}$ with $\alpha = -1$ and $\beta = 0$ can be interpreted as modeling the kind of *hasty* human driving behavior we are all too familiar with.

To provide another example of a sensible criterion, consider cost functionals of the form

$$J_{v_d}(x_i) = \int_0^{t_f} (v_d - \dot{x}_i(t))^2 + (\ddot{x}_i(t))^2 \, \mathrm{d}t, \tag{2.13}$$

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. Furthermore, note that the occurance of $(\cdot)^2$ is generally prefered over $|\cdot|$ in practical applications, because the former has nicer smoothness properties; the absolute value is not differentiable at zero.

2.1.4 Trajectory optimization

Given the above three main ingredients, we are now ready to formulate the trajectory optimization problem in terms of finding vehicle trajectories x_i that are

safe $\rightarrow$ x_i satisfy the safety constraints of Section 2.1.1 at all times;

admissible $\rightarrow$ x_i satisfy the dynamic motion constraints of Section 2.1.2;

efficient $\rightarrow$ x_i are optimal w.r.t. some criterium of Section 2.1.3.

Assume the system parameters $(L, \mathcal{E}, \bar{v}, \omega, \bar{\omega}, t_f)$ to be fixed, where $t_f > 0$ denotes some final simulation time. We write $x := \{x_i : i \in \mathcal{N}\}$ to denote the *joint position* of vehicles, and $x(t)$ the corresponding *joint trajectory*. Consider the following optimization problem over x.

$$
\boxed{
\begin{aligned}
J(s) := \min_{x} \;& \sum_{i \in \mathcal{N}} J(x_i) && && \text{(T)} \\
\text{such that } \;& x_i \in D(s_i) && \text{for all } i \in \mathcal{N}, && \text{(T.1)} \\
& x_i(t) - x_j(t) \geq L && \text{for all } (i,j) \in \mathcal{C}, && \text{(T.2)} \\
& (x_i(t), x_j(t)) \notin \mathcal{E}^2 && \text{for all } \{i,j\} \in \mathcal{D}. && \text{(T.3)}
\end{aligned}
}
$$

Since the decision variable x is a set of continuous-time trajectories, we are dealing with an infinite-dimensional optimization problem. Furthermore, the problem is inherently non-convex—observe that the set of feasible configurations $\mathcal{X}_{ij}$ for two vehicles is already non-convex. This non-convexity is due to the fact that we must decide which of the vehicles crosses the intersection first. In the remainder of this chapter, we will discuss ways of dealing with this non-convexity.

2.2 Direct transcription

The above trajectory optimization problem (T) can be solved by employing a method known as *direct transcription* [32, 70]. The key idea of direct transcription is to reformulate the continuous-time optimal control problem into a finite-dimensional optimization problem by discretizing time. At each discrete time step, 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 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 time discretization grid. Let K denote the number of time steps and let Δt denote the time step size, then we define

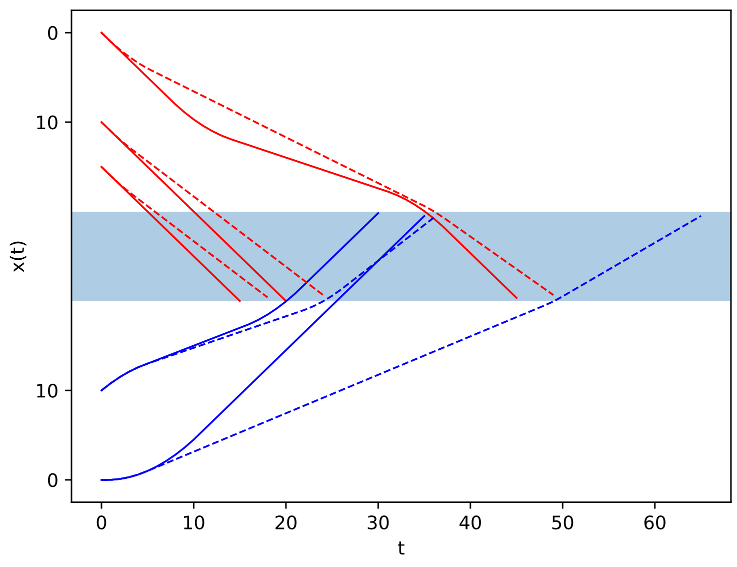
$$\mathbb{T} := \{0, \Delta t, \ldots, K\Delta t\}.$$

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

$$
\begin{aligned}
x_i(t + \Delta t) &= x_i(t) + v_i(t)\Delta t, \\
v_i(t + \Delta t) &= v_i(t) + u_i(t)\Delta t,
\end{aligned}
$$

for each $t \in \mathbb{T}\backslash\{K\Delta t\}$ 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

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



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
K	$=$	50
Δt	$=$	1

Figure 2.4: 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. Trajectories are not drawn beyond this region.

Encoding of the safe crossing constraints needs some additional attention, because they represent a binary "disjunctive" decision. Following the approach in [28], 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:

$$x_i(t) \leq B + \delta_i(t)M, \tag{2.14a}$$

$$x_i(t) \geq E + L - \gamma_i(t)M, \tag{2.14b}$$

$$\delta_i(t) + \delta_j(t) + \gamma_i(t) + \gamma_j(t) \leq 3, \tag{2.14c}$$

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* (2.14a), since M is chosen sufficiently large such that the inequality is trivially true. Analogously, setting $\gamma_i(t) = 1$ deactivates (2.14b). Hence, the constraint (2.14c) can be interpreted as limiting the number of deactivations to three, which is equivalent to requiring at least one of the following four inequalities to hold:

$$x_i(t) \leq B, \quad x_j(t) \leq B, \quad x_i(t) \geq E + L, \quad x_j(t) \geq E + L,$$

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 2.4.

Remark. *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 use a spline interpolation technique. We refer to [70] for a friendly introduction to trajectory optimization in general and [32] for a comprehensive tutorial on the direct collocation methods, which also contains a concise overview of related numerical methods (ibid., Section 9).*

2.3 Occupancy time slots and trajectories

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

$$\tau(x_i) := \min\{t \in [0, t_f] : x_i(t) = B\}, \tag{2.15a}$$
$$\xi(x_i) := \min\{t \in [0, t_f] : x_i(t) = E + L\}. \tag{2.15b}$$

Observe that $\tau(x_i)$ and $\xi(x_i)$ specify the times $t \in (\tau(x_i), \xi(x_i))$ when vehicle i occupies the intersection[2]. Hence, we will refer to $(\tau(x_i), \xi(x_i))$ as the *occupancy time slot* of vehicle i.

Recall the encoding of the collision constraints using binary variables in (2.14). Similarly, observe that the collision-avoidance constraints

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

are completely equivalent to the constraints

$$\big(\tau(x_i), \xi(x_i)\big) \cap \big(\tau(x_j), \xi(x_j)\big) = \varnothing \quad \text{for all } \{i, j\} \in \mathcal{D}. \tag{2.16}$$

2.3.1 Bilevel decomposition

The main idea of the decomposition is to make these crossing 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 crossing 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 (2.16), the trajectory optimization problem essentially reduces to solving a separate lower-level problem for each route.

Before we make this precise, let us first introduce some 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.

Lower-level problem. 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*

$$F(y_r, z_r, s_r) := \min_{x_r} \sum_{i \in \mathcal{N}_r} J(x_i) \tag{L}$$
$$\text{s.t.} \quad x_i \in D(s_i) \quad \text{for all } i \in \mathcal{N}_r, \tag{L.1}$$
$$x_i(t) - x_j(t) \geq L \quad \text{for all } (i, j) \in \mathcal{C} \cap \mathcal{N}_r^2, \tag{L.2}$$
$$x_i(y_i) = B \quad \text{for all } i \in \mathcal{N}_r, \tag{L.3}$$
$$x_i(z_i) = E + L \quad \text{for all } i \in \mathcal{N}_r. \tag{L.4}$$

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 $(y_r, z_r) \in \mathcal{S}(s_r)$ to denote the set of occupancy time slots for which (L) is feasible, which we call *feasible route schedules*.

Upper-level problem. The upper-level problem is now to find a set of occupancy time slots satisfying (2.16), 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*

$$U(s) := \min_{y,z} \sum_{r \in \mathcal{R}} F(y_r, z_r, s_r) \tag{U}$$
$$\text{s.t.} \quad (y_i, z_i) \cap (y_j, z_j) = \varnothing \quad \text{for all } \{i, j\} \in \mathcal{D}, \tag{U.1}$$
$$(y_r, z_r) \in \mathcal{S}(s_r) \quad \text{for all } r \in \mathcal{R}. \tag{U.2}$$

[2]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, we use the convention that "$(\infty, \infty) = \varnothing$".

Forced crossing. Without further assumptions, problem (U) is almost equivalent to the original problem (T). As we already noted when defining cost functionals, there is a subtle issue in the fact that a feasible solution x of (T) does not necessarily cross the intersection. To avoid this, we explicitly require the crossing and exit times to be finite, which yields variant (T′). With this technical assumption, we briefly sketch how validity of the decomposition can be established rigorously.

$$
\begin{aligned}
T'(s) := \min_{x} \quad & \sum_{i \in \mathcal{N}} J(x_i) && && \text{(T}'\text{)} \\
\text{s.t.} \quad & x_i \in D(s_i) && \text{for all } i \in \mathcal{N}, && \text{(T.1)} \\
& x_i(t) - x_j(t) \geq L && \text{for all } (i,j) \in \mathcal{C}, && \text{(T.2)} \\
& (x_i(t), x_j(t)) \notin \mathcal{E}^2 && \text{for all } \{i,j\} \in \mathcal{D}, && \text{(T.3)} \\
& \tau(x_i) < \infty && \text{for all } i \in \mathcal{N}, && \text{(T}'\text{.4)} \\
& \xi(x_i) < \infty && \text{for all } i \in \mathcal{N}. && \text{(T}'\text{.5)}
\end{aligned}
$$

Theorem 2.1 ([30, Theorem 1])**.** *Assume problem* (T′) *is feasible and an optimal solution exists, then it is equivalent to the decomposed problem* (U)*.*

Proof (sketch). Given some initial state s, let $\mathcal{X}(s)$ denote the feasible trajectories of (T′) and let $\mathcal{S}(s)$ denote the feasible schedules of (U). We argue that each $(y,z) \in \mathcal{S}(s)$ can be transformed into a feasible solution $x \in \mathcal{X}$ and vice versa.

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

$$\sum_{r \in \mathcal{R}} F(y_r^*, z_r^*, s_r) = \sum_{i \in \mathcal{N}} J(x_i^*) \geq T'(s). \tag{2.17}$$

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. Hence,

$$\sum_{i \in \mathcal{N}} J(x_i') = \sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{N}_r} J(x_i') \geq \sum_{r \in \mathcal{R}} F(y_r', z_r', s_r) \geq U(s). \tag{2.18}$$

Finally, taking $\min_{y^*, z^*}$ in (2.17) and $\min_{x'}$ in (2.18) yields $U(s) = T'(s)$, as desired. □

2.3.2 Decomposition-based approximation algorithms

Even with this decomposition, the problem is still difficult 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 $\mathcal{S}(s_r)$ is not easily characterizable, e.g., as a system of inequalities. In other words, we could say that the upper- and lower-problem are *tightly coupled* by feasibility.

Nevertheless, the decomposition provides a good conceptual basis for approximation algorithms. One approach that has been explored in the literature is to explicitly approximate the set of feasible time slots $\mathcal{S}(s_r)$ and the objective function F. Suppose we have a single vehicle per route, so $n_r = 1$ for all $r \in \mathcal{R}$ and $(y_r, z_r) \equiv (y_i, z_i)$. For this case, the following approximation scheme has been proposed [28].

Evaluate $F(y_i, z_i, s_i)$ at fixed grid points to fit a strictly convex approximation $\hat{F}$. Next, the set of feasible occupancy slots can be estimated by the following polyhedral subset

$$\hat{\mathcal{S}}(s_i) = \{(y,z) : y \in [T_i^l, T_i^h],\ l_i(y) \leq z \leq u_i(y)\} \subseteq \mathcal{S}(s_i). \tag{2.19}$$

Here, the occupancy time slot is required to be between the earliest crossing time T_i^l and the latest crossing time T_i^h. Roughly speaking, these bounds can simply be computed by finding the fastest and the slowest route, respectively. The exit time bounds $u_i(y)$ and $l_i(y)$ are strictly increasing affine.

The approximations $\hat{F}$ and $\hat{\mathcal{S}}$ turn the upper-level problem into a mixed-integer quadratic problem, which has been shown to be easier to solve than the original direct transcription, but at the cost of suboptimality. Of course, the set $\mathcal{S}(s_i)$ is larger than its polyhedral approximation, so optimality is not guaranteed. The authors prove [30] that this scheme does not affect feasibility of the problem.

2.3.3 Isolating the timing problem

Although the approximation approach sketched above is very interesting, we will not aim to tackle the problem in its full generality, by making some mild assumptions that expose and separate the combinatorial nature of the problem.

Assumption 2.3 (Full speed)**.** We assume that all vehicles in the system start at full speed $v_i^0 = \bar{v}$ and enter the intersection at full speed. Therefore, we add $\dot{x}_i(\tau(x_i)) = \bar{v}$ as additional constraint to (T′) and, equivalently, we add $\dot{x}_i(y_i) = \bar{v}$ to the lower-level problem (L).

Definition 2.2 (Earliest crossing time)**.** Given some trajectory x_i with initial state $s_i = (x_i^0, \bar{v})$, define the *earliest crossing time* of vehicle i to be

$$a_i := (B - x_i^0)/\bar{v}. \tag{2.20}$$

Definition 2.3. Suppose some vehicle drives at full speed as long as it occupies the intersection. In that case, the vehicle occupies the intersection for a duration of precisely

$$\sigma := (E + L - B)/\bar{v}. \tag{2.21}$$

Furthermore, define the *processing time*

$$\rho := L/\bar{v}. \tag{2.22}$$

We will call the difference $\delta := \sigma - \rho$ the *switch-over time*, which is non-negative by definition.

Definition 2.4 (Delay criterion)**.** We introduce the following simple trajectory cost criterion. Let the *delay criterion* be defined $J_d(x_i) = \tau(x_i) - a_i$. To simplify notation in the next chapter, we will also consider the related criterion $J'_d(x_i) = \tau(x_i)$.

The delay criterion simplifies our problem significantly, because F becomes a trivial function of the start times of the occupancy time slots. For every route $r \in \mathcal{R}$, we have

$$\begin{aligned} F(y_r, z_r, s_r) &= \min_{x_r} \sum_{i \in \mathcal{N}_r} J_d(x_i) \qquad \text{s.t. } (\text{L.1}, \text{L.2}, \text{L.3}, \text{L.4}) \\ &= \left.\begin{aligned} &\min_{x_r} \sum_{i \in \mathcal{N}_r} \tau(x_i) - a_i && \text{s.t. } (\text{L.1}, \text{L.2}, \text{L.4}) \\ &\quad\text{s.t. } x_i(y_i) = B && \text{for all } i \in \mathcal{N}_r \end{aligned}\right\} = \sum_{i \in \mathcal{N}_r} y_i - a_i, \end{aligned}$$

such that the upper-level problem reduces to

$$\begin{aligned} U(s) = \min_{y,z} \quad & \sum_{i \in \mathcal{N}} y_i - a_i && && (2.23\text{a}) \\ \text{s.t.} \quad & (y_i, z_i) \cap (y_j, z_j) = \varnothing && \text{for all } \{i, j\} \in \mathcal{D}, && (2.23\text{b}) \\ & (y_r, z_r) \in \mathcal{S}(s_r) && \text{for all } r \in \mathcal{R}. && (2.23\text{c}) \end{aligned}$$

The difficulty that remains is how to pick occupancy schedules that guarantee feasibility of the lower-level problem. Given some $(y, z) \in \mathcal{S}(s)$, every lower-level solution x_i satisfies $\dot{x}_i(y_i) = \bar{v}$, so we can safely let vehicles continue at full speed across the intersection, because z_i is not involved in $J_d(\cdot)$. In that case, each vehicle leaves the intersection completely after σ time, so we fix

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

Observe that σ is the minimum time between the crossing times of vehicles from conflicting routes. Consequently, we can limit ourselves to

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

where $y_r + \sigma$ means $\{y_i + \sigma : i \in \mathcal{N}_r\}$. Similarly, ρ is the minimum time between crossing times of vehicles on the same route. The sufficient conditions of Assumption 2.2 allow a polyhedral characterization of $\mathcal{Y}(s_r)$; it can be shown[3] that

$$y_r \in \mathcal{Y}(s_r) \iff \begin{cases} a_i \leq y_i & \text{for all } i \in \mathcal{N}_r, \\ y_i + \rho \leq y_j & \text{for all } (i,j) \in \mathcal{C} \cap \mathcal{N}_r^2. \end{cases} \tag{2.25}$$

Hence, problem (2.23) reduces to the following *crossing time scheduling* problem:

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

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, which we will do in the next chapter.

Instances and schedules. The next chapter will be about solving (C), so let us first take a step back and spend a few words on explaining our notation for specifying (sets of) instances of this problem. We will write $a := \{a_i\}_{i \in \mathcal{N}}$ to denote the set of *earliest crossing times*. We will mostly keep the indices $\mathcal{N}$ implicit in this notation; when necessary, we write $\mathcal{N}(a)$. Note that R, $\{n_r\}_{r \in \mathcal{R}}$, $\mathcal{C}$ and $\mathcal{D}$ are all uniquely determined by $\mathcal{N}(a)$; we use similar notation when we need to stress that they belong to a. Therefore, we will write an instance of (C) as a triple

$$(a, \rho, \sigma).$$

It follows immediately from its definition that a has to satisfy the following property, which is stated here for easy reference.

Proposition 2.1. *The arrival times satisfy $a_i + \rho \leq a_j$, for each conjunctive pair $(i,j) \in \mathcal{C}$.*

Throughout the rest of this thesis, we will use a simple way of visualizing problem instances and candidate solutions that is somewhat similar to the classical Gantt chart: we use a bar chart with time on the vertical axis. Each row corresponds to one of the routes of the instance and each vehicle is represented by a block, whose width corresponds to the fixed processing time ρ. The block for vehicle i is positioned such that its left border is located at the earliest crossing time a_i. For example, Figure 2.5 depicts some instance with

$$\begin{aligned} a = \{&a_{11} = 0.31,\ a_{12} = 1.78, \\ &a_{21} = 0.74,\ a_{22} = 2.86,\ a_{23} = 4.60\}. \end{aligned}$$

To visualize some candidate schedule y, we collapse all the route rows into a single row, see Figure 2.6, such that every block starts at the schedule time y_i. Therefore, every block represents the crossing time slot that is allocated to the corresponding vehicle. We sometimes add arrows of length $\delta = \sigma - \rho$ to illustrate the necessary switch-over time between consecutive time slots for different routes.

[3] A rigorous proof of this fact is outside the scope of the current project.

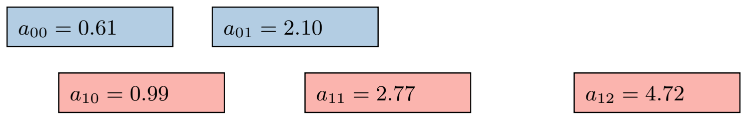


Figure 2.5: Instance example with earliest arrival times indicated for each vehicle.

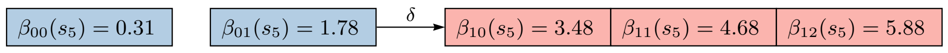


Figure 2.6: Some schedule as example solution to the problem instance shown in Figure 2.5.

2.4 Notes and references

The single intersection model is mostly based on the model described in the PhD thesis of Hult [27, 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 [38, Appendix A] and [41, Section 2.2].

The bilevel decomposition and approximation scheme of Section 2.3.1 are further detailed in [28, 30]. 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 [22, 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 the end of this section, we included some comments from the optimal control perspective. For a textbook introduction of optimal control theory, we recommend the book of Liberzon [42]. 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. [23]. 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.

Chapter 3

Crossing time scheduling

Problems like (C) have been studied extensively, so we first discuss traditional solution methods. Our problem is a combinatorial optimization problem, for which a wealth of general solution techniques is available [14], including algorithms specifically tailored to the case when feasible solutions can be thought of as time schedules [17, 52]. One of the most prominent distinction among 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: the number of feasible solutions typically explodes whenever larger instances are considered—it is virtually impossible to enumerate all feasible solutions, evaluate their objective value, and pick the best one, in any reasonable amount of time.

Integer programming. The first part of this chapter illustrates a standard method of dealing with combinatorial problems, in which the discrete decision are encoded as integers. More specifically, in Section 3.1, we show how (C) can be formulated as a *mixed-integer linear programming* [8], which can be solved in a principled way using the branch-and-bound strategy. Integer programming is a mature technology, as reflected in the availability of general solvers and software tooling. 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. We conclude the first part of this chapter by evaluating how much the running time is reduced by these cutting planes.

Constructive heuristics. In many practical applications—like in our case of motion planning for autonomous vehicles—the time that is available to obtain a solution is limited. Hence, it is not essential to obtain the best solution. This is the main motivation for the development of so-called heuristics, which discard the optimality guarantee in favor of speed. In other words, we want a good solution, fast!

While integer programming can be interpreted as a smart exhaustive search over the space of feasible solutions, we will consider a simple class of heuristic algorithms that is based on a simple step-by-step construction. Section 3.2 provides the necessary setup required for our methodology. Specifically, we use a simple graph reformulation of (C) to show that it can be reduced to a problem of finding a sequence of route indices, which we will call the *route order*. The resulting route order is the starting point for our discussion in Chapter 4, where we propose to use techniques from machine learning to automate the process of developing an algorithm for finding good route orders.

3.1 Integer programming

One of the fundamental algorithmic ideas in combinatorial optimization is the branch-and-bound strategy, in which the space of feasible solutions is systematically explored by keeping track of the search tree and iteratively subdividing the feasible region into smaller subproblems (branching), computing bounds on the best possible objective value within each subproblem (bounding), and discarding those parts of the feasible region that contain subproblems whose bound proves them incapable of containing an optimal solution (pruning).

The branch-and-bound 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 the 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.

3.1.1 Reformulation as MILP

We show how to reformulate the crossing time scheduling problem (C) into the form (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

$$\bar{\mathcal{D}} = \{(i, j) : \{i, j\} \in \mathcal{D},\ r(i) < r(j)\}. \tag{3.1}$$

With this definition, we obtain the MILP reformulation

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

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

Remark 3.1. *The careful reader may observe that we removed the earliest arrival times a_i from the delay criterion. Since a is assumed to be given, we can always compute the original delay cost by subtracting $\sum_i a_i$.*

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 [20] (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[1] package to call the solver from the comfort of Python. We report some numerical results based on this method in Section 3.1.4. First, we show that the above formulation can be made a little sharper.

[1] https://amplpy.ampl.com/

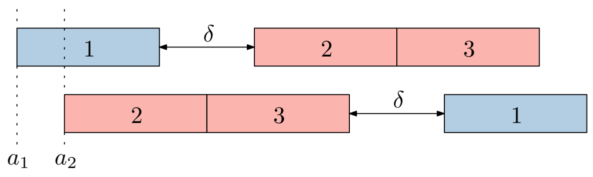


Figure 3.1: Illustration of the two possible sequences of vehicles in Example 3.1.

3.1.2 Optimal substructure: platoons

The branch-and-bound approach guarantees to find an optimal solution, but it does not provide any guarantees on the required running time. Roughly speaking, the running time depends mainly on two aspects: (i) size of the instance—in our case: number of vehicles—and (ii) the amount of structure it exhibits. More specifically, the running 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 introduce additional inequalities without changing the set of feasible solutions

$$\{y : Ay \leq b, A'y \leq b', y_j \in \mathbb{Z}, j \in \mathcal{I}\} = \{y : Ay \leq b, y_j \in \mathbb{Z}, j \in \mathcal{I}\}, \tag{3.2}$$

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*.

Next, we present insight into some optimal substructures of optimal solutions. This analysis will serve as the basis for defining three types of cutting planes. Let us first consider some simple problem instances to start shaping our intuition.

Example 3.1. Consider two routes having one and two vehicles, see Figure 3.1. 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 $a_2 \geq a_1 + \sigma$, the sequence $1,2,3$ is simply optimal. For $a_2 < a_1 + \sigma$, we compare the sequence $1,2,3$ with $2,3,1$, which are illustrated in Figure 3.1. The first has $\sum_i y_i = \sigma + (\sigma + \rho) = 3\rho + 2\delta$, while the second sequence has $\sum_i y_i = a_2 + (a_2 + \rho) + (a_2 + 2\rho + \delta) = 3a_2 + 3\rho + \delta$. Therefore, we conclude that the second sequence is optimal if and only if

$$a_2 \leq \delta/3, \tag{3.3}$$

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

In this 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 δ. 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 3.1. A sequence of consecutive vehicles $(r, l+1), (r, l+2), \dots, (r, l+n)$ from some route r is called a *platoon* of size n if and only if

$$a_{(r,k)} + \rho = a_{(r,k+1)} \qquad \text{for all } l < k < l + n. \tag{3.4}$$

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

$$y_{(r,k)} + \rho < y_{(r,k+1)} \qquad \text{for some } l < k < l + n. \tag{3.5}$$

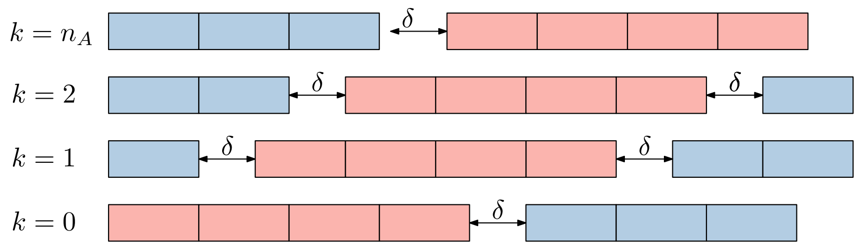


Figure 3.2: Different ways to split platoon A (in blue), regarding Example 3.2, assuming equal earliest crossing times $a_A = a_B$ with $n_A = 3$ vehicles in platoon A and some arbitrary number of vehicles n_B in platoon B (here 4).

Example 3.2. Suppose we have two routes $\mathcal{R} = \{A, B\}$, each having exactly one platoon, denoted as $P_A = ((A,1), \dots, (A, n_A))$, $P_B = ((B,1), \dots, (B, n_B))$. 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 $a_A \leq a_B < n_A\rho + \delta$. 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 3.2 for an illustration of the situation in case of $a_A = a_B$. For $0 < k \leq n_A$, we have

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

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

$$\sum_{i \in \mathcal{N}} y_i(n_A) = \delta n_B + \sum_{j=1}^{n_A+n_B} (j-1)\rho, \tag{3.6}$$

since $\max\{\delta, a_B - n_A\rho\} = \delta$ 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

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

Comparing this to (3.6), we conclude that placing B in front is optimal whenever

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

which directly generalizes the condition (3.3) 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 3.1 (Platoon Preservation [43])**.** *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 $y_{i^*} + \rho = y_{j^*}$.*

3.1.3 Cutting planes

Based on the above analysis of optimal substructure, we will define three types of cutting planes for (C′). First, we will use Theorem 3.1 to formulate two types of cutting planes—the idea is to incorporate this necessary condition in the integer program. For every conjunctive pair $(i, j) \in \mathcal{C}$, we introduce a binary variable $b_{ij} \in \{0, 1\}$, which must satisfy

$$\begin{aligned} b_{ij} = 0 &\iff y_i + \rho < a_j, \\ b_{ij} = 1 &\iff y_i + \rho = a_j. \end{aligned}$$

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

$$\begin{aligned} &y_i + \rho < a_j + b_{ij}M, \\ &y_i + \rho \geq a_j - (1 - b_{ij})M. \end{aligned}$$

Now observe that the statement of Theorem 3.1 applied to (i, j) is equivalent to the inequality

$$y_i + \rho \geq y_j - (1 - b_{ij})M. \tag{conj.cut}$$

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

Using the definition of b_{ij}, we can derive a second type of cutting planes on the disjunctive decision variables γ. Whenever $b_{ij} = 1$, Theorem 3.1 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

$$b_{ij} + (1 - \gamma_{ik}) + \gamma_{jk} \leq 2, \tag{disj.cut.1}$$

$$b_{ij} + \gamma_{ik} + (1 - \gamma_{jk}) \leq 2. \tag{disj.cut.2}$$

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

The last type of cutting plane 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 $\mathit{pred}(i)$ denote the set of all vehicles on route $r(i)$ that cross the intersection strictly before vehicle i and let $\mathit{succ}(i)$ denote those that cross strictly later, so we have

$$\begin{aligned} \mathit{pred}(i) &:= \{(r(i), k) : k > k(i)\}, \\ \mathit{succ}(i) &:= \{(r(i), k) : k < k(i)\}. \end{aligned}$$

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

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

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

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

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.*

3.1.4 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 four routes and report running times as a function of the number of vehicles per route, see Figure 3.3. Each average is computed over 10 problem instances. Problem instances are generated with fixed $\rho = 1.2$ and $\sigma = 1.7$

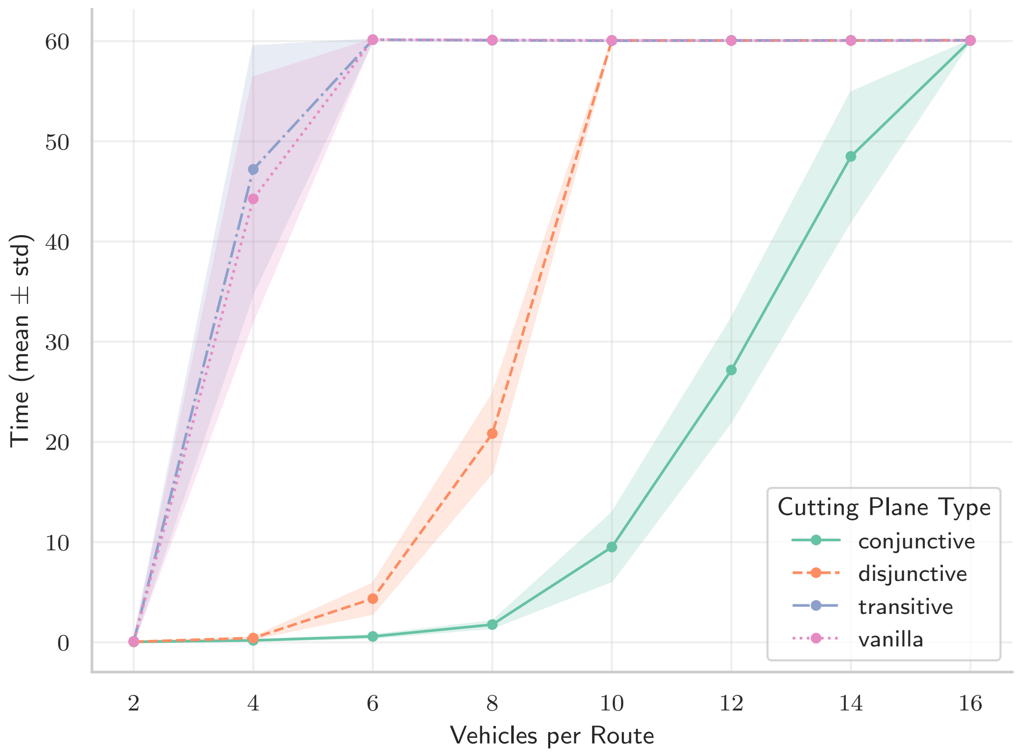


Figure 3.3: Average run time of branch-and-cut as a function of the number of arriving vehicles per route, for instances with four routes. Vanilla means MILP without cuts.

and random earliest crossing times a_i, generated with gaps that are uniformly distributed on $[0, 1]$. A detailed discussion of how a_i is generated is deferred to Section 4.3. To keep the total computational effort within reasonable limits, we impose a time limit of 60 seconds per instance, so some observations correspond to cases when the time limit is reached. 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.

The results clearly show that the conjunctive cutting planes provide the most runtime improvement, enabling us to solve instances that contain four times more vehicles than the vanilla approach without any cutting planes. Interestingly, it seems that adding transitive cuts even degrades performance slightly, which might be explained by the fact that there are a lot of these cuts. It could be interesting for further research to try some sort of lazy row generation approach here

3.2 Route order optimization

In Chapter 4, we will present a way to use machine learning methods to solve the crossing time scheduling problem (C). To enable the approach there, we will first show that there exists a more compact representation of optimal schedules: *they can actually be represented as a sequence of route indices.* This allows us to formulate the scheduling problem in terms of a particularly simple sequential decision problem. We will make this precise in Section 4.1, where we will define this in terms of a Markov decision process, that we can try to solve with existing reinforcement learning methods.

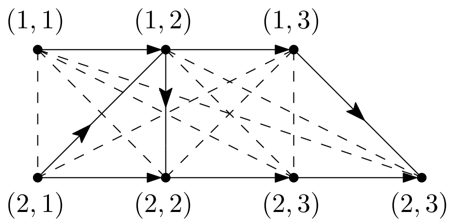


Figure 3.4: Illustration of disjunctive graph for some instance with $R = 2$ routes, and $N = 7$ vehicles in total, with $n_1 = 3$ on the first route and $n_2 = 4$ on the second route. Dummy nodes are not shown to keep the figure clean. The horizontal solid arrows represent the conjunctive arcs, the three other solid arrows represent some partial choice $\mathcal{O}$, so the remaining dashed lines represent pairs of disjunctive arcs that are not chosen.

For ease of reference, let us restate the MILP reformulation we derived above

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

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 chains of inequalities. To make this 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.

3.2.1 Disjunctive graph

Suppose we are given some instance (a, ρ, σ) of problem (C). The idea of the disjunctive graph representation is to encode each of the inequality constraints in (C$'$) as a weighted arc in a directed graph. For each vehicle $i \in \mathcal{N}$, the earliest crossing time constraint $a_i \leq y_i$ is encoded by introducing a dummy node $i' \in \mathcal{N}'$ and introducing a directed arc $i' \to i$ with weight $w(i', i) := 0$. Let $\mathcal{N}' := \{i' : i \in \mathcal{N}\}$ denote the set of all dummy nodes and let $\mathcal{C}'$ denote the set of all dummy arcs. For each pair $(i,j) \in \mathcal{C}$, the conjunctive constraint $y_i + \rho \leq y_j$ is encoded by a so-called *conjunctive arc* $i \to j$ with weight $w(i,j) := \rho$. Lastly, recall that the the remaining constraints encode that we want exactly one of the constraints $y_i + \sigma \leq y_j$ or $y_j + \sigma \leq y_i$ to hold for each conflict $\{i,j\} \in \mathcal{D}$. In the disjunctive graph, we model this as choosing between the *disjunctive* arcs $i \to j$ or $j \to i$, each having weight $w(j,i) = w(i,j) := \sigma$. This leads to the formal definition below.

Definition 3.2. Let $\omega = (a, \rho, \sigma)$ be some given instance of problem (C) and let $\mathcal{O} \subset \mathcal{N}^2$ be a *selection* of disjunctive arcs such that

$$(i,j) \in \mathcal{O} \implies (j,i) \notin \mathcal{O}, \qquad \text{for all } \{i,j\} \in \mathcal{D}. \tag{dg.1}$$

The *disjunctive graph of ω with respect to $\mathcal{O}$* is defined as the weighted directed graph

$$\mathcal{G}^\omega(\mathcal{O}) := (\mathcal{N} \cup \mathcal{N}', \mathcal{C} \cup \mathcal{C}' \cup \mathcal{O}, w). \tag{dg.2}$$

The graph $\mathcal{G}_0^\omega := \mathcal{G}^\omega(\varnothing)$ is called the *empty disjunctive graph* for ω. When $\mathcal{O}$ is some *complete selection*, which is when

$$(i,j) \notin \mathcal{O} \implies (j,i) \in \mathcal{O}, \quad \text{for all } \{i,j\} \in \mathcal{D}, \tag{dg.3}$$

then we call $\mathcal{G}^\omega(\mathcal{O})$ a *complete disjunctive graph* for ω. If $\mathcal{O}$ is neither empty nor complete, then $\mathcal{G}^\omega(\mathcal{O})$ is called a *partial disjunctive graph* for ω.

Figure 3.4 shows a partial disjunctive graph for some small example instance. Furthermore, when the particular instance is clear from the context or left unspecified, we will drop the ω superscript to keep notation simple.

3.2.2 Reducing the space of feasible 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 $\mathcal{G}(\mathcal{O})$, 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

$$y_j \geq \max_{i \in \mathcal{N}^-(j)} y_i + w(i,j) \quad \text{for all } j \in \mathcal{N}. \tag{3.8}$$

Definition 3.3. If schedule y satisfies (3.8) with equality, it is called an *active schedule*.

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

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

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 ρ and σ are positive, such that all the weights in the above inequality are positive. This shows that the absence of cycles in $\mathcal{G}(\mathcal{O})$ is necessary for feasibility of (AS). It is not surprising that it is actually also sufficient. Specifically, we show that when $\mathcal{G}(\mathcal{O})$ is acyclic, there exists an active schedule, so (AS) is feasible in that case. For brevity, we will say "$\mathcal{O}$ is acyclic" to mean "$\mathcal{G}(\mathcal{O})$ is acyclic". We first recall the following elementary property of directed acyclic graphs, whose proof can be found in Appendix F.

Proposition 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 $v_1, v_2, \dots, v_{|V|}$ 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 3.1. *Let $\mathcal{O}$ be an acyclic selection, then there exists a unique active schedule $y(\mathcal{O})$.*

Proof. Since $\mathcal{G}(\mathcal{O})$ is acyclic, Proposition 3.1 gives us some topological order $v_1, v_2, \dots, v_{2N}$. Observe that there are exactly N minimal nodes in $\mathcal{G}(\mathcal{O})$, which are precisely the dummy nodes $\mathcal{N}'$, for which we will use the notational convention that $y_{i'} := a_i$, for each $i \in \mathcal{N}$. Next, we visit the nodes $\mathcal{N}$ in their topological order $v_{N+1}, \dots, v_{2N}$ and for each visited node v, we set

$$y_v := \max_{u \in \mathcal{N}^-(v)} y_u + w(u, v). \tag{3.9}$$

Every time we perform this update, note that each in-neighbor $u \in \mathcal{N}^-(v)$ has been visited before, because the fact that arc $u \to v$ exists means that u appears before v in the topological order. Therefore, y_u has already been assigned a value so the right-hand side of (3.9) is well-defined. Hence, we obtain the unique schedule $y(\mathcal{O}) = \{y_i : i \in \mathcal{N}\}$ satisfying (3.8) with equality. $\square$

We emphasize that the lemma does not require $\mathcal{O}$ to be complete. Observe that adding more disjunctive arcs to $\mathcal{O}$ can only cause the crossing times in the resulting active schedule to increase. Hence, we have the following lower bounding property of active schedules.

Corollary 3.1. *Let $\mathcal{O} \subseteq \mathcal{O}^*$ be two acyclic selections and let $y = y(\mathcal{O})$ and $y^* = y(\mathcal{O}^*)$ denote the corresponding active schedules, then $y_i \leq y_i^*$ for all $i \in \mathcal{N}$.*

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. This way, we essentially reduce the infinite space of candidate schedules to a the finite space of candidate active schedules. The discussion so far can be concisely summarized as:

Lemma 3.2. *Problem (C) is equivalent to*

$$\min_{\mathcal{O}} \sum_{i \in \mathcal{N}} y_i(\mathcal{O}) \quad \textit{s.t. } \mathcal{O} \textit{ is acylic and complete.} \tag{C$''$}$$

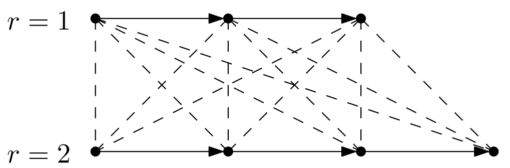
Proof. When $\mathcal{O}$ is acyclic and complete, then the unique active schedule $y(\mathcal{O})$ is clearly a feasible solution to (C$'$) when setting $\gamma_{ij} = \mathbf{1}\{(i,j) \in \mathcal{O}\}$. Conversely, let y be an optimal solution of (C$'$). It must be an active schedule, otherwise some component y_i is not minimal, contradicting optimality. Furthermore, γ_{ij} determines an complete selection $\mathcal{O}$. This selection must be acyclic, otherwise there is a contradicting chain of inequalities in (C$'$). Hence, $\mathcal{O}$ is a feasible solution for (C$''$) such that $y = y(\mathcal{O})$. $\square$

3.2.3 Ordering vehicles and routes

The disjunctive graph representation was helpful in deriving the results above, but it is less suitable for the sequential decision making problem that we will introduce in Chapter 4. A more natural representation of the crossing order is to just consider a permutation of vehicle indices. Since the relative order of vehicles on the same route is fixed, it is even simpler to only consider a sequence of routes. This leads to the following definition.

Definition 3.4. Let $\nu \in \mathcal{V}_t$ denote a *vehicle order* of length t, which is some sequence $\nu_{1:t} = (\nu_1, \nu_2, \dots, \nu_t)$ such that for each $t_1 \leq t_2$, we have

$$r(\nu_{t_1}) = r(\nu_{t_2}) \implies k(\nu_{t_1}) \leq k(\nu_{t_2}). \tag{3.10}$$



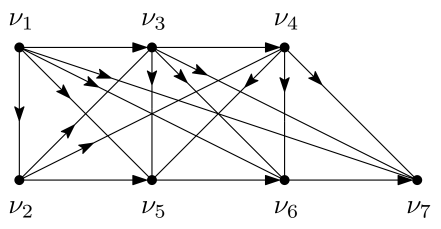


Figure 3.5: Illustration of disjunctive graphs, again without dummy nodes. The left graph illustrates the *empty* disjunctive graph with $\mathcal{O} = \varnothing$ and $\eta = \nu = ()$. The selection shown on the right is *acyclic* and *complete* and corresponds uniquely to the vehicle order $\nu = ((1, 1), (2, 1), (1, 2), (1, 3), (2, 2), (2, 3), (2, 4))$ and the route order $\eta = (1, 2, 1, 1, 2, 2, 2)$.

We refer to $\mathcal{V} := \mathcal{V}_N$ as the set of all *complete* vehicle orders. Similarly, let $\eta \in E_t$ denote a *route order* of length t, which is some sequence $\eta_{1:t} = (\eta_1, \eta_2, \dots, \eta_t)$ such that it contain at most n_r occurences of symbol r for every route $r \in \mathcal{R}$. We refer to $E := E_N$ as the set of all *complete* route orders.

The requirement (3.10) essentially says that the relative vehicle order of each route must be respected. Given some vehicle order $\nu \in \mathcal{V}_t$, let the route order $\eta = \eta(\nu)$ be uniquely defined by $\eta_\tau = r(\nu_\tau)$, for every $\tau \in \{1, \dots, t\}$. Conversely, given some route order $\eta \in E_t$, it is easy to see that there must also be a unique vehicle order $\nu = \nu(\eta)$ such that $\eta = \eta(\nu)$; this vehicle order can be constructed using a simple procedure in which the routes are visited according to the route order.[2] This shows that there is a bijection between vehicle orders and route orders, so we can use them interchangeably.

Suppose we are given some vehicle order $\nu \in E_t$, then we construct a selection $\mathcal{O}(\nu)$. We visit the non-dummy nodes according to order ν and for each node ν_k, we just pick all disjunctive arcs $\nu_k \to v$ such that $u \in \mathcal{N}$ is on another route, so $r(\nu_k) \neq r(u)$, and we did not visit u before. This way, we end up with a unique acyclic selection, which is complete if and only if $t = N$.

Remark 3.2. *Conversely, suppose that $\mathcal{O}$ is some acyclic selection, then there is generally not a single candidate route or vehicle order, see for example Figure 3.4. In other words, the subgraph of $\mathcal{G}(\mathcal{O})$ induced over $\mathcal{N}$—which is a DAG—might have multiple topological orders (Proposition 3.1). However, when $\mathcal{O}$ is complete, there is a unique topological order $\nu(\mathcal{O})$, which we prove by contradiction: suppose ν' is a topological order, $\nu' \neq \nu$, then there is at least one pair of indices $i, j \in \mathcal{N}$ that appears in opposite orders in ν and ν', say, i appears before j in ν and j appears before i in ν'; due to the conjunctive arcs, it follows that $r(i) \neq r(j)$, but then either $(i, j) \in \mathcal{O}$ or $(j, i) \in \mathcal{O}$ would contradict either ν or ν'.*

Figure 3.5 illustrates the above discussion by showing the three related representations of schedules. We introduce the following notation.

Definition 3.5. Let $\eta \in E_t$ be some route order, uniquely corresponding to some vehicle order $\nu \in \mathcal{V}_t$, uniquely corresponding to some acyclic selection $\mathcal{O}(\nu)$, obtained using the construction above, then we write $y(\eta) = y(\nu) = y(\mathcal{O}(\nu))$ to denote the corresponding unique active schedule.

Theorem 3.2. *Problem* (C) *is equivalent to the* route ordering problem

$$\min_{\eta \in E} \sum_{i \in \mathcal{N}} y_i(\eta) - a_i. \tag{D}$$

Proof. We established a bijection between $\mathcal{V}$ and E. Let Ω denote thet set of all acyclic complete selections $\mathcal{O}$. The construction above shows that there is also a bijection between Ω and $\mathcal{V}$. The theorem then follows from Lemma 3.2 and Remark 3.1. □

[2]We can think of the vehicles $(r, 1), (r, 2), \dots (r, n_r)$ forming a queue. Each time we visit a route, we pick the next vehicle in the queue; for example, the first time we visit route r, we pick vehicle $(r, 1)$.

3.3 Notes and references

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 [52, Appendix A]. Note that decomposition techniques similar in nature to the decomposition of Section 2.3.1 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.[3] Such techniques have been applied in air traffic scheduling [45] and job-shop problems arising in general traffic scheduling problems [37]. Interestingly, these works also propose an alternative ("noncompact") MILP formulation that does not depend on the big-M trick. From our current limited understanding, this seems to be related to, and might be a better alternative to our transitive cutting planes.

The disjunctive graph formulation is originally due to [57], but this technical note does not seem to be publicly accessible via the internet. This reformulation is widespread in the scheduling literature. For a little bit more background and to see it applied to some canonical scheduling problem, we refer the reader to the discussion of job shop scheduling in Appendix B. Lemma F.3 is very much related to the general notion of an *active schedule* in the scheduling literature, see [52, Definition 2.3.3].

[3]Fun fact: this famous method is named after Jacques Benders [1], who was a professor here at TU/e .

Chapter 4

Learning to schedule

We will now explain how the general ML for CO framework presented in the previous chapter can be applied to the crossing time problem (C). As we discussed before, the most important decision is the design of the state space of the MDP algorithm model, since this essentially determines the class of algorithms over which we are optimizing—it is our main way of incorporating prior knowledge and intuition into the model. We will now show how our findings so far lead to a natural choice of states and corresponding algorithmic structure.

4.1 Constructive scheduling

We concluded Chapter 3 by reformulating the crossing time scheduling problem (C) as the equivalent route ordering problem

$$\min_{\eta \in E} \sum_{i \in \mathcal{N}} y_i(\eta) - a_i, \tag{D}$$

in which we try to find some valid route order $\eta \in E$ that minimizes the total vehicle delay. Given such an optimal route order η^*, we showed that the corresponding optimal crossing time schedule $y(\eta^*)$ can be obtained by solving the linear program (AS) or by using the simple procedure described in the proof of Lemma 3.1. Instead of searching directly over all feasible $\eta \in E$, we propose to develop a *constructive procedure*. The general idea is as follows. Each state encodes a *partial schedule*, in which only the first few vehicles of each route are said to be *scheduled*. Initially, all vehicles are unscheduled. Each step corresponds to picking some route and then adding the next unscheduled vehicle on that route to the schedule. We continue until we arrive at a complete schedule.

4.1.1 MDP definition

We first define the single-instance scheduling MDP, represented as the tuple

$$\mathcal{M}^\omega := (\mathcal{S}^\omega, \mathcal{A}^\omega, p^\omega, r^\omega), \tag{4.1}$$

for a single problem instance $\omega = (a, \rho, \sigma)$ for the route ordering problem (D). We extend this to sets of instances afterwards. To keep notation light, we will mostly drop the superscript ω when it is clear from the context.

States and decision epochs. The initial state of $\mathcal{M}^\omega$ is $s_0 := (\omega, \varnothing)$. All vehicles are said to be *unscheduled* in s_0. The MDP has exactly N decision epochs, which is equal to the total number of vehicles in ω. Each intermediate state $s_t = (\omega, \eta)$ encodes some current partial route order $\eta \in E_t$. We will define the actions and transitions such that the final state $s_N = (\omega, \eta_{1:N})$ is guaranteed to represent a complete schedule. The state space is thus

$$\mathcal{S} = \{\omega\} \times \bigcup_{t=0}^{N} E_t. \tag{4.2}$$

Actions and transitions. At every decision epoch, we have to choose some route r that has unscheduled vehicles left, so the routes form the action space $\mathcal{A} = \mathcal{R}$. Vehicles are added to the partial schedule according to the relative order in which they appear on their route. To make this precise, let $k_r(s_t)$ denote the number of vehicles on route r that are *scheduled* in state s_t, so initially, we have $k(s_0) = 0$. Hence, the admissible actions at state s_t are

$$\mathcal{A}(s_t) := \{r : k_r(s_t) < n_r\}. \tag{4.3}$$

When we choose $r \in \mathcal{A}(s_t)$ in some state $s_t = (\omega, \eta_{1:t})$, we simply append r to the current route order to end up in $s_{t+1} = (\omega, \eta_{1:t+1})$ with $\eta_{t+1} = r$. Let the transition function $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ be defined as

$$p((\omega, \eta_{1:t+1}) \mid (\omega, \eta_{1:t}), r) = \mathbf{1}\{\eta_{t+1} = r\}. \tag{4.4}$$

This is a deterministic transition function, so we equivalently write $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ with

$$p(s_t, a) = s_{t+1}. \tag{4.5}$$

Final state. After N steps, all vehicles must have been scheduled. Since the transitions are deterministic, the final state $s_N = (\omega, \eta)$ is uniquely determined by the sequence of actions $\eta \in E$ that were taken during the episode. Hence, given some deterministic policy function $\pi : \mathcal{S} \to \mathcal{A}$, let $\eta(\omega, \pi)$ denote the unique final state that is reached by *rolling out* policy π, which means that each next action is chosen as $\eta_t = \pi(s_{t-1})$.

Given some complete route order $\eta \in E$, the quantity

$$R(\eta) := \sum_{i \in \mathcal{N}} a_i - y_i(\eta) \tag{4.6}$$

is a natural candidate to use as the definition of the final reward. Note that we reversed the sign with respect to the original objective in (D), because the convention is that we aim to maximize reward in MDPs. However, a single final reward is not convenient in practice. It is often better for convergence of the learning algorithm when rewards happen more evenly throughout the episode. Therefore, we show how such equivalent *dense rewards* can be defined in terms of changes of certain lower bounds, which we define next.

Lower bounds. Consider some state $s_t = (\omega, \eta_{1:t})$. Let $y_i^\omega(\eta)$ be the active schedule for instance ω with respect to route order η, as given by Lemma 3.1. Define the *crossing time lower bounds*

$$\beta_i(s_t) = y_i^\omega(\eta_{1:t}) \quad \text{for each } i \in \mathcal{N}. \tag{4.7}$$

These values can be interpreted as providing lower bounds on the crossing times for any *completion* of the current partial schedule, as shown by the following properties:

(i) Recall that the earliest crossing times a need to satisfy $a_i + \rho \leq a_j$, see Proposition 2.1. Since the empty disjunctive graph consists of only a single chain of conjunctive arcs per route, this implies that the initial lower bounds are given by

$$\beta_i(s_0) = a_i \quad \text{for all } i \in \mathcal{N}. \tag{4.8}$$

(ii) Consider some state $s_t = (\omega, \eta)$ and let $\nu = \nu(\eta)$ be the current vehicle order. Let $\mathcal{O}(s_t) = \mathcal{O}(\nu)$ denote the current acyclic selection induced by ν, see Definition 3.5. Now let $\eta^* \in E_{t^*}$ be some completion of η, so such that $t^* \geq t$ and $\eta^*_{1:t} = \eta$. Observe that $s^* = (\omega, \eta^*)$ is some possible future state following s_t. Let $\mathcal{O}(\omega^*)$ denote the corresponding acyclic selection, then we have $\mathcal{O}(s_t) \subset \mathcal{O}(s^*)$ by construction. Hence, it follows from Corollary 3.1 that we have

$$\beta_i(s_t) \leq \beta_i(s^*) \quad \text{for all } i \in \mathcal{N}. \tag{4.9}$$

(iii) Let s_t be some non-initial state, so with $t \geq 1$ and let i be the vehicle that was scheduled at step t, so $i = \nu_t = (\eta_t, k_r(s_{t-1}) + 1)$. Let $\omega^* \in \{s_{t+1}, \dots, s_N\}$ be some possible future state, then as argued in the previous point, we have $\mathcal{O}(s_t) \subset \mathcal{O}(\omega^*)$. By construction, $\mathcal{O}(\omega^*)$ does not contain additional arcs that end in node i, which means that its lower bound does not change further:

$$\beta_i(s_t) = \beta_i(s_{t+1}) = \cdots = \beta_i(s_N). \tag{4.10}$$

(iv) For a final state $s_N = (\omega, \eta)$, the lower bounds are the final active schedule

$$\beta_i(s_N) = y_i(\eta) \quad \text{for all } i \in \mathcal{N}.$$

Dense rewards. For each transition $s_{t-1} \xrightarrow{\eta_t} s_t$, we define the reward to be

$$r(s_{t-1}, \eta_t, s_t) = \sum_{i \in \mathcal{N}} \beta_i(s_{t-1}) - \beta_i(s_t). \tag{4.11}$$

Hence, for some episode $\tau = ((\omega, \varnothing), \eta_1, s_1, \dots, s_{N-1}, \eta_N, s_N)$, which is uniquely determined by $\eta \in E$ because p is deterministic, together with some reward sequence $(r_t)_{t=1}^N$, with $r_t = r(s_{t-1}, \eta_t, s_t)$, the total episodic reward is given by the telescoping sum

$$\begin{aligned} \sum_{t=1}^{N} r_t &= \sum_{i \in \mathcal{N}} \sum_{t=1}^{N} \beta_i(s_{t-1}) - \beta_i(s_t) \\ &= \sum_{i \in \mathcal{N}} \beta_i(s_0) - \beta_i(s_N) = \sum_{i \in \mathcal{N}} a_i - y_i(\eta) = R(\eta), \end{aligned}$$

where we used the properties of β_i outlined above. Hence, maximizing the episodic reward corresponds to minimizing the delay objective of (D).

Joining single-instance MDPs. Observe that the state space is a union of disjoint sets

$$\mathcal{S}^\omega = \{(\omega, \varnothing)\} \sqcup \mathcal{S}_1^\omega \sqcup \mathcal{S}_2^\omega \sqcup \cdots \sqcup \mathcal{S}_N^\omega, \tag{4.12}$$

where each $\mathcal{S}_t^\omega = \{\omega\} \times E_t$ denotes the set of all possible partial schedules that can be obtained from $(\omega, \varnothing)$ after scheduling exactly t vehicles. Further, suppose we have two distinct problem instances $\omega_1 \neq \omega_2$, then we clearly have

$$\mathcal{S}^{\omega_1} \cap \mathcal{S}^{\omega_2} = \varnothing. \tag{4.13}$$

Given some set of instances $\mathcal{I}$ such that each $\omega \in \mathcal{I}$ has the same set of routes $\mathcal{R}$, we can consider the following *joint state space*

$$\mathcal{S} = \bigsqcup_{\omega \in \mathcal{I}} \mathcal{S}^\omega. \tag{4.14}$$

Therefore, let $\mathcal{A} := \mathcal{R}$ and define $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ such that

$$\begin{aligned} p(u, a) &= p^\omega(u, a) \quad && \text{when } u \in \mathcal{S}^\omega, \\ r(u, a) &= r^\omega(u, a) \quad && \text{when } u \in \mathcal{S}^\omega, \end{aligned}$$

then the *joint constructive scheduling* MDP for $\mathcal{I}$ is

$$\mathcal{M}_{\text{constr}} = (\mathcal{S}, \mathcal{A}, p, r). \tag{M}$$

Episode visualization. Recall the visualization for instances and corresponding complete schedules we introduced in the previous chapter. In order to strengthen our intuition about the above MDP definition, we extend this type of visualization to partial schedules such that we can visualize all steps of the MDP, which is explained in Figure 4.1.

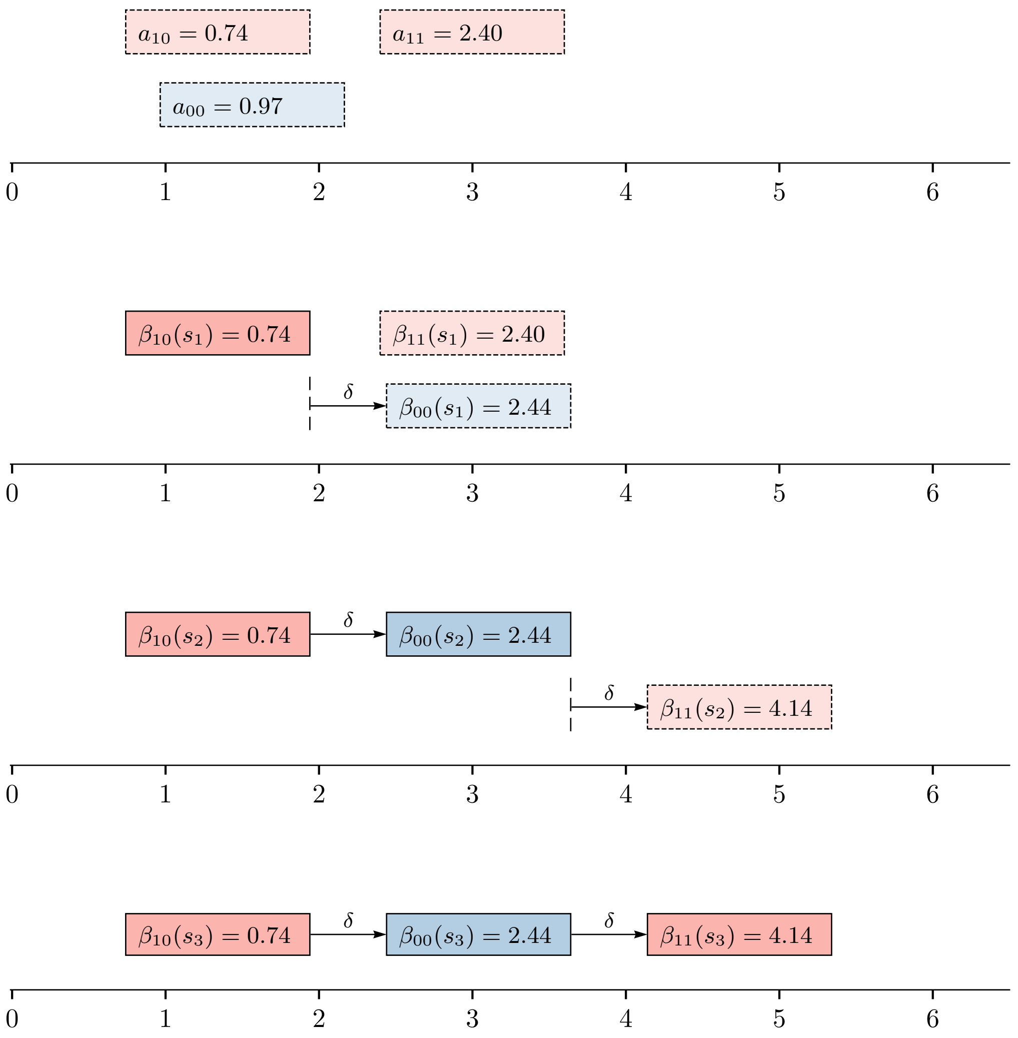


Figure 4.1: *Visualizing an episode of the MDP.* Every one of the four frames in the figure illustrates some state s_t of the MDP, by the box of vehicle i at its current lower bound $\beta_i(s_t)$. Hence, the first and last frame visualize the initial state of the instances and some possible complete schedule, respectively, in the same way as before. The frames in between show partial schedule states: the solid boxes in the top row of each frame show the current partial schedule; the remaining boxes in the top row show the unscheduled vehicles on the route that was last scheduled; the other rows show the unscheduled vehicles of the other routes. Note that we draw the "switch" arrow only for the routes other than the current and we draw them in the final complete schedule.

4.1.2 Learning objective

For the MDP defined above, we now define the learning objective in terms of finding some policy π, to which the rest of this chapter is devoted. Afterwards, we briefly discuss the set Π of policies under consideration. Concrete distributions P will be defined at the end of the chapter, when we do some numerical experiments.

Definition 4.1. Given some distribution P over problem instances of some class $\mathcal{I}$, the *learning to schedule* problem is defined as

$$\max_{\pi \in \Pi} \mathbb{E}_{\omega \sim P}[R(\eta(\omega, \pi))], \tag{MDP}$$

where $\eta(\omega, \pi)$ denotes the unique schedule obtained for ω by applying π in $\mathcal{M}_{\text{constr}}$ with initial state $(\omega, \varnothing)$, and where $R(\cdot)$ denotes the total episodic reward as defined above.

A common starting point is to let Π be all stochastic policies[1] $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. However, since the state space of $\mathcal{M}_{\text{constr}}$ decomposes into finite subspaces as sketched above, it can be shown that the following class of policies is general enough, in the sense that it contains optimal policies.

Definition 4.2. Let Π_{p} be the set of *pure policies*, being all functions $\pi : \mathcal{S} \to \mathcal{A}$.

Theorem 4.1. *There exists an optimal policy π^* for $\mathcal{M}_{\text{constr}}$ such that $\pi^* \in \Pi_{\text{p}}$.*

Proof (sketch). For each initial state $(\omega, \varnothing)$, only a finite part $\mathcal{S}^\omega$ of the state space is reachable. It is well-known that finite-state MDPs have an optimal deterministic policy π^ω, which is not necessarily stationary, so it consists of a sequence of decision rules $\pi^\omega = (d_1^\omega, d_2^\omega, \dots, d_N^\omega)$. However, since $\mathcal{S}^\omega = \{(\omega, \varnothing)\} \sqcup \mathcal{S}_1^\omega \sqcup \cdots \sqcup \mathcal{S}_N^\omega$, each decision rule d_t^ω is only ever applied to states in $\mathcal{S}_t^\omega$, so we can safely assume $d_1^\omega = d_2^\omega = \cdots = d_N^\omega$ without changing the final solution $\eta(\omega, \pi^\omega)$. Since the state space is partitioned as $\mathcal{S} = \sqcup_\omega \{\mathcal{S}^\omega\}$, we can join π^ω for each ω to obtain a pure policy π. $\square$

4.1.3 State space reduction

It can be shown that the optimal completion of a partial schedule does not depend on the vehicles that have already been scheduled, see for example Figure 4.2. To make this more precise, we introduce a mapping from the state space to particular subset of *abstract states*. More concretely, given some partial schedule, we perform the following two operations.

1. The vehicles that have already been scheduled are discarded. We take the unscheduled vehicles and do a simple renumbering of their indices.
2. It is not difficult to see that the optimal completion of a partial schedule is invariant to uniform time-shifts of all lower bounds. Hence, we fix a unique reference time for the partial schedule.
3. We subtract the reference time from the lower bound of each unscheduled vehicle. The resulting shifted lower bounds become the arrival times in a new empty instance, consisting of only unscheduled vehicles.

State abstraction. Let $s = (\omega, \eta)$ be some state of $\mathcal{M}_{\text{constr}}$. Let $\mathcal{N}(s)$ denote the index set of the problem instance and let $n_r(s)$ and $k_r(s)$ denote the total number vehicles and the number of scheduled vehicles on route r, respectively. Furthermore, let $\nu(s)$ denote the unique vehicle order corresponding to the current route order η.

[1]Even more general classes of policies can be considered, when we allow the next action to depend on the entire history of states and actions encountered so far. In our setting, it can be shown that this does not result in more powerful policies, in the sense that there always exists an optimal policy that is Markov, i.e, is only a function of the current state.

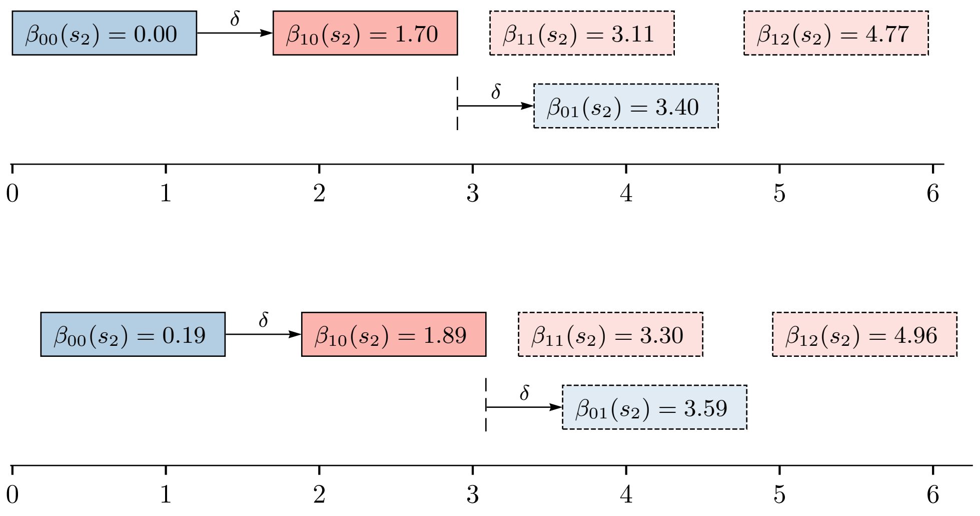


Figure 4.2: Two states of the constructive scheduling MDP that are equivalent with respect to optimal completion: it can be shown that any optimal completion of the above partial schedule is also optimal for the lower one.

We define the *abstract state mapping* $\phi : \mathcal{S} \to \mathcal{S}$ by directly constructing $\phi(s)$. The abstract state is an empty instance $\phi(s) = ((\bar{a}, \rho, \sigma), \varnothing)$ with index set $\bar{\mathcal{N}}(s)$. We simply keep the indices of the unscheduled vehicles and renumber them to obtain

$$\bar{\mathcal{N}}(s) := \{(r, k) : r \in \mathcal{R},\ k_r(s) + k \leq n_r(s),\ k \in \mathbb{N}^+\}. \tag{4.15}$$

Let $\{\beta_i(s) : i \notin \nu(s)\}$ be the set of lower bounds of all unscheduled vehicles. Among these lower bounds, let $\beta^{(1)}(s)$ and $\beta^{(2)}(s)$ denote the smallest and second smallest, respectively. Define the arrivals times by time-shifting the lower bounds by

$$c(s) := \min\{\beta^{(1)}(s), \beta^{(2)}(s) - \delta\}, \tag{4.16a}$$
$$\bar{a}_i(s) := \beta_i(s) - c(s) \quad \text{for all } i \in \bar{\mathcal{N}}(s). \tag{4.16b}$$

Theorem 4.2. *Let $\pi^* \in \Pi_{\text{p}}$ be some optimal policy for* (MDP)*, then there exists some function $f : \mathcal{S} \to \mathcal{S}$ such that $\pi^* = f \circ \phi$.*

Proof (sketch). Consider a pair $s_1, s_2 \in \mathcal{S}$ such that $\phi(s_1) = \phi(s_2)$. This means that they have the same sets of admissible actions, so pick some arbitrary $a \in \mathcal{A}(s_1) = \mathcal{A}(s_2)$. For any such s_1, s_2 and a, it can be shown that $r(s_1, a, p(s_1, a)) = r(s_2, a, p(s_2, a))$ and

$$\phi(p(s_1, a)) = \phi(p(s_2, a)),$$

which shows that ϕ is a model-irrelevant state abstraction, see Definition 3 in [40], then Theorem 3 of the same paper guarantees the existence of f. The same result can also be established using the more general framework of MDP homomorphisms [54]. □

Next, we provide two suggestions for parameterizing f.

4.2 Policy parameterizations

4.2.1 Platoon preserving policies

Let s_t denote the current partial schedule and assume that it is not an initial state, so $t \geq 1$. Let the vehicle that was scheduled in the previous step be denoted by $i^* = \nu_t = (r^*, k^*)$ and suppose $j^* = (r^*, k^* + 1)$ exists. By construction of the lower bounds, $\beta_{i^*}(s_t) = y_{i^*}$ is the final crossing time of vehicle i^*, regardless of the remaining actions taken in the MDP.

Recall from Section 3.1.2 the platoon preservation Theorem 3.1, which essentially says that whenever it is possible to schedule a vehicle immediately after its predecessor on the same route, then this must be done in an optimal schedule. Therefore, the platoon preservation rule requires

$$\beta_t(i^*) + \rho \geq a(j^*), \qquad \text{then next action is } r^*, \tag{exhaustive}$$

where $r^* = \eta_t$ is the last scheduled route. Whenever a policy π satisfies this rule, we say that it is *exhaustive*, so the platoon preservation theorem implies that each optimal policy π^* must be exhaustive.

Threshold heuristic. We will propose a simple class of exhaustive policies. The idea is to consider the next unscheduled vehicle of the route that was last chosen. We aim to continue on the same route as long as possible, to avoid introducing unnecessary switch-over time δ. Recall that a platoon was defined as a sequence of consecutive vehicles $(r, l+1), (r, l+2), \dots, (r, l+n)$ from some route r such that

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

An alternative interpretation of the platoon preservation theorem says that platoons should be treated as a whole unit: given some other vehicle on a different route, the platoon is either scheduled entirely before it or entirely after it. In other words, the platoon is never *split* in an optimal schedule.

Based on this idea, we might think that the same holds whenever a vehicle can be scheduled *sufficiently* soon after its predecessor. In other words, we might relax the definition of a platoon to

$$a(r,k) + \rho + \tau \geq a(r, k+1) \quad \text{for all } l < k < l + n,$$

for some small *threshold* parameter $\tau \geq 0$. Therefore, we construct a policy that satisfies

$$\text{if } \beta_t(i^*) + \rho + \tau \geq a(j^*), \quad \text{then next action is } r^*, \tag{threshold}$$

where $r^* = \eta_t$ is again the last scheduled route.

Definition 4.3. The *threshold policy* is defined as $\pi_\tau(s) := f_\tau(\phi(s))$, with

$$f_\tau(((\bar{a}, \rho, \sigma), \varnothing)) := \begin{cases} r(\arg\min_i \bar{a}_i) & \text{if } \min_i \bar{a}_i \leq \tau, \\ r(\arg\min_i \{\bar{a}_i : \bar{a}_i \geq \delta\}) & \text{otherwise.} \end{cases} \tag{4.17}$$

Proposition 4.1. *Policy π_τ satisfies* (threshold) *and hence* (exhaustive).

Grid search. Given some set of instances D_{train}, we approximate the best parameter τ^* through a simple grid search by selecting K evenly spaced $\tau_k \in [\tau_{\min}, \tau_{\max}]$ and evaluating

$$\tau^* := \arg\max_{\tau_k} \sum_{\omega \in D_{\text{train}}} R(\eta(\omega, \pi_{\tau_k})).$$

Alternatively, we could use some derivative-free optimization method like Brent's method.

4.2.2 Differentiable policies

Instead of manually trying to develop a good parameterization of f, we will weaken our prior by considering a simple neural parameterization. Instead of only considering the first unscheduled vehicle on each route, we now propose a parameterization that takes into account the sequence of relative lower bounds of unscheduled vehicles as follows. Let $\bar{n}_r$ denote the number of unscheduled vehicles in s, such that $\bar{\mathcal{N}}(s)$ contains precisely $\bar{n}_r$ vehicles on route r, which we denote as

$$\bar{a}_r := \{\bar{a}_{rk} : 1 \leq k \leq \bar{n}_r\}. \tag{4.18}$$

Each of these *route sequences* $\bar{a}_r$ is encoded by a recurrent neural network f_θ^{RNN} to produce a *route embedding* $f_\theta^{\text{RNN}}(\bar{a}_r)$. The concatenation of these route embeddings is fed through a fully connected layer f_θ^{full} and softmax to produce a distribution over routes.

This means that we will explicitly consider stochastic policies, which is numerically attractive. We could call this policy the *training policy* π_θ. However, note that eventually want a deterministic *target policy* π_d. Hence, we need to specify how to make the training policy deterministic again. We will use the following simple greedy action selection. Let $\pi = f \circ \phi : \mathcal{S} \to \Delta(A)$ be some stochastic policy, then let $\pi_\text{d} : \mathcal{S} \to \mathcal{A}$ be defined as

$$\pi_\text{d}(s) = \max_{a \in \mathcal{A}(s)} \pi(a|s). \tag{4.19}$$

Imitation learning. The simplest method to find θ is to use an *imitation learning* approach. First, we collect *expert demonstration* by taking some instance from the distribution of interest and replaying an optimal route sequence on the MDP, while keeping track of the visited states. This results in a set of state-action pairs for which we can consider a simple prediction task, which we will now make precise.

Let $\mathcal{I}_\text{train}$ be some set of training instances. For each $\omega \in \mathcal{I}_\text{train}$, let η^ω denote some optimal route sequence, which we can be found, for example, by solving (C′), and let $N(\omega)$ denote the total number of vehicles in ω. For each $\omega \in \mathcal{I}_\text{train}$, we let $\mathcal{M}_\text{constr}$ start in $s_0 = (\omega, \varnothing)$ and we replay η^ω to obtain the set of state-action pairs

$$\mathcal{X}_\text{train} = \{(\phi((\omega, \eta_{1:t-1})), \eta_t) : t \in \{1, \dots, N(\omega)\}, \omega \in \mathcal{I}_\text{train}\}. \tag{4.20}$$

We can now consider the supervised learning task of predicting the next action, given the current state. For example, in the case of two route $\mathcal{R} = 2$, we can use the binary cross entropy loss

$$L_\theta(\mathcal{X}_\text{train}) = -\frac{1}{|\mathcal{X}_\text{train}|} \sum_{(s,a) \in \mathcal{X}_\text{train}} \mathbf{1}\{a = 1\} \log(\pi_\theta(1|s)) + \mathbf{1}\{a = 2\} \log(\pi_\theta(2|s)), \tag{4.21}$$

where $\pi_\theta(a|s) \in [0, 1]$ denotes probability assigned to action a by the policy.

Policy-gradient optimization. The above method relies on the ability to compute optimal schedules in the first place. For very large instances, this is not possible in reasonable time. Therefore, it is attractive to consider a method that is based on sampling from the MDP. These kinds of policy-gradient methods rely on the fundamental Policy Gradient Theorem. Define

$$J(\theta) := \mathbb{E}_{\omega \sim P}[R(\eta(\omega, \pi_\theta))] \tag{4.22}$$

which is the expected reward of $\mathcal{M}_\text{constr}$, when sampling the initial state according to P. Note that the expectation is also taken over the random actions of our training policy. The policy gradient theorem essentially tells us that we can use samples of an episode $\tau = (s_0, \eta_1, r_1, s_1, \dots, s_N, r_N)$ to estimate the gradient as

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{\omega \sim P, \tau \sim \pi_\theta} \left[\sum_{t=0}^{N-1} \nabla_\theta \log \pi_\theta(\eta_{t+1}|s_t) \sum_{t'=t+1}^{N} r_{t'} \right]. \tag{4.23}$$

A well-known manifestations of this sampling approach is the classical REINFORCE with baseline algorithm. However, our current purpose is not to understand the fundamentals of policy-gradient algorithms, but rather to investigate the feasibility of this approach in general. Hence, we will use the modern and very popular Proximal Policy Optimization (PPO) algorithm [59].

4.3 Experimental results

The evaluation of a computational method's performance is based on two aspects. Of course, the quality of the produced solutions is important. Second, we need to take into account the

Table 4.1: *Overview of results.* This table shows all our results.

time that the algorithm requires to compute the solutions. We need to be careful here, because some of our methods involve both training time as well as inference time. Furthermore, we will evaluate generalization of learned policies along the axis of instance size and the axis of instance structure, which will be made more concrete next.

Instance size. Assume that parameters ρ and σ are fixed globally. For some set of instances $\mathcal{I}$, we fix the number of routes R and the number n_r of vehicles per route. Therefore, we write $N(\omega) = N(\mathcal{I})$ to denote the size each instance $\omega \in \mathcal{I}$.

Instance distribution. Next, we define some distribution $P(\mathcal{I})$ over instances in terms of *distributions over arrival sequences.* For each route $r \in \mathcal{R}$, consider the stochastic process

$$A_r := (A_{r1}, A_{r2}, \dots, A_{rn_r}),$$

satisfying the recursion

$$A_{rk} = A_{r,k-1} + X_{rk} + \rho,$$

where $X_{rk} \overset{\text{iid}}{\sim} F_r$ are *interarrival times* with a common mean μ_r. We will call $\lambda_r := (\mu_r + \rho)^{-1}$ the *arrival intensity*[2] of route r. We define $(a, \rho, \sigma) \sim P(\mathcal{I})$ if and only if $a \sim A$ with $a_r := \{a_{rk} : 1 \leq k \leq n_r\} \overset{\text{iid}}{\sim} A_r$ for each $r \in \mathcal{R}$. This definition naturally satisfies $a_i + \rho \leq a_j$ for each $(i, j) \in \mathcal{C}$, see Proposition 2.1.

Degree of platooning. In order to model the natural occurrence of platoons, we could let the interarrival time distribution F be a mixture 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 instance, we could use a mixture of two exponentials, such that

$$\begin{aligned} X \sim F(x) &= p(1 - e^{-x/\mu_s}) + (1 - p)(1 - e^{-x/\mu_l}), \qquad \text{(bimodal)} \\ \mathbb{E}X &= p\mu_s + (1 - p)\mu_l, \end{aligned}$$

assuming $\mu_s < \mu_l$. Observe that the parameter p determines the average length of platoons. Given some fixed target arrival intensity λ, we can use the transformation

$$\mu_l = \frac{(\lambda^{-1} - \rho) - p\mu_s}{1 - p}$$

to keep the arrival intensity equal to λ when varying the other parameters.

Experiment design. We study the effect of problem distribution $P(\mathcal{I})$ by varying R, n and F. Let us first fix three route arrival distributions $F_{\text{low}}, F_{\text{med}}$ and F_{high}, modeling arrivals with low, medium and high chance of vehicles occuring in platoons; we achieve this by letting the interarrival times be distributed according to (bimodal) with parameters as given in the following table:

[2]Assume $n_r = \infty$ and let N_t denote the *counting process* of the *renewal process* $A_r = (A_{rk})_{k=1}^{\infty}$ with interarrival times $X_{rk} + \rho$. In other words, N_t counts the cumulative number of arrivals up to time t. By the *renewal theorem*, we obtain the *limiting density* of arrivals

$$\mathbb{E}(N_{t+h}) - \mathbb{E}(N_t) \to h(\mu_r + \rho)^{-1} \quad \text{as } t \to \infty,$$

for time distance $h > 0$. Hence, $\lambda_r := (\mu_r + \rho)^{-1}$ naturally captures the limiting density of arrivals, when we increase the time window under consideration.

F	p	μ_s	μ_l
F_{low}	0.5	0.1	10.0
F_{med}	0.3	0.1	7.17
F_{high}	0.1	0.1	5.6

To enable a sensible comparison across instances of various sizes, we report the quality of a solution in terms of the average delay per vehicle $R(\eta, \omega)/N(\omega)$. Given some problem instance $\omega \sim P$, let η^* denote some optimal route order, which we compute by solving the MILP problem (C′). We use a fixed time limit of 60 seconds per instance for the branch-and-bound procedure, in order to keep the total analysis time within reason. Therefore, we might not always be able to obtain the actual η^* some of the larger instances; we use the best solution found by the solver as a proxy. Given some η, we define its *optimality gap* as

$$\text{gap}(\omega, \eta) := (R(\omega, \eta)/R(\omega, \eta^*)) - 1.$$

For each method π, we report the average optimality gap

$$\text{gap}(\pi) = \sum_{\omega \in D_{\text{test}}} \frac{\text{gap}(\omega, \eta^\pi(\omega))}{|D_{\text{test}}|}, \quad \text{with } D_{\text{test}} \sim P(\mathcal{I}).$$

4.4 Notes and references

Constructive scheduling procedures similar to the one proposed in this chapter are common in the scheduling literature and are also known as *dispatching rules* in the context of job shop scheduling. For some broader context, we refer to Appendix D, where we discuss some existing works in the ML for CO literature that focus on finding dispatching rules for job shop problems, which are very similar to our crossing time scheduling problem.

Chapter 5

Networks of intersections

We now turn to extending the methodology of the previous chapters to networks. With some mild assumptions on the paths that vehicles take through the network, much of what we have done so far extends to networks of intersections.

Network model. First, we introduce some graph notation to model simple networks of intersections with fixed routes. To keep it simple, we assume that routes are independent, in the sense that vehicles of different routes only meet at intersection; there is no merging and splitting of routes. We will see that the trajectory optimization problem (T) considered so far extends immediately to this model.

Decomposition. It is possible to solve the resulting network trajectory problem using direct transcription. However, we will now demonstrate this. Instead, we show how the bilevel decomposition extends to networks. Similar to what we did in Section 2.3.3, we will again assume that the objective only involves delay and that vehicles are required to cross intersections at full speed. In this case, however, delay can be measured in two ways: at every intersection, or at the last intesection of the vehicle's route.

We argue that—in principle—these assumption can be used to reduce the problem to a scheduling problem. However, compared to the single intersection setting, there is one additional complicating factor. Before, we did not have to take into account the maximum number of vehicles that an approaching lane can handle. However, in the current setting, we can no longer ignore the finite nature of the lanes between intersection, so they have a *finite capacity* for vehicles. The precise relationship between the length of the lane and the maximum acceleration and speed of vehicles turned out to be nontrivial. In other words, it is not easy to characterize the feasibility of trajectories as a simple set of linear inequalities, like we did for the single intersection, see (2.25). Our preliminary investigation towards a similar characterization are outlined in Chapter 6.

Constructive scheduling. Instead of a precise characterization of feasibility, we will work with some sufficient conditions. This results in a scheduling problem that is somewhat similar to the classical job shop scheduling problem. Again, we aim to solve this problem using a constructive heuristic. Quite a few existing works have adopted the same strategy and applied deep reinforcement learning successfully. Hence, we mention some pointers for further research to leverage the recent advances in this field to solve our particular job shop variant.

We identify an issue with the constructive approach. Scheduling decisions involve picking both an intersection and a route. However, we will see that the order in which intersections are chosen does not matter for the final schedule. This redundancy makes the convergence of procedures like policy-gradient very slow. Removing this redundancy is an interesting direction for further research.

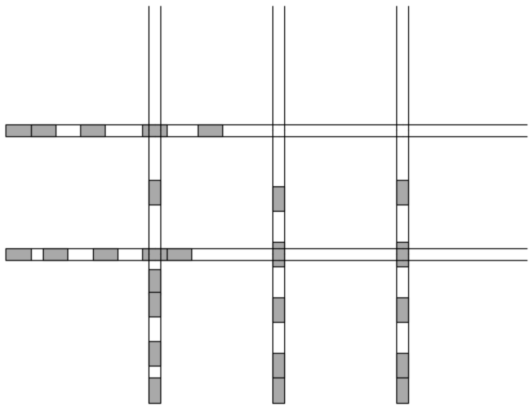


Figure 5.1: Illustration of some grid-like network of intersections with vehicles drawn as grey rectangles. There are five vehicle routes: two from east to west and three from south to north. Turning at intersections is not allowed.

5.1 Trajectory optimization in the network model

We now extend the single intersection model to a simple network of intersections with single lanes where turning is not allowed, illustrated in Figure 5.1.

Connectivity graph. We define a directed graph $(\bar{V}, A)$ with nodes $\bar{V}$ and arcs A, representing the possible paths that vehicles can follow. Nodes with only outgoing arcs are *entrypoints* and nodes with only incoming arcs are *exitpoints*. Let V be the set of *intersections*, which are all the nodes with in-degree at least two. Let $d(v, w)$ denote the distance between nodes v and w. For each route index $r \in \mathcal{R}$, we let

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

be the path that vehicles on this route follow through the network. We require that the first node $v_r(0)$ is an entrypoint and that the last node $v_r(m_r + 1)$ is an exitpoint and we write

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

to denote the path restricted to intersections. We say that some $(v, w) \in A$ is on path V_r whenever v and w are two consecutive nodes on the path and we write A_r to denote the set of all these arcs. We require that routes can only overlap at nodes by making the following assumption.

Assumption 5.1. For every distinct routes $p, q \in \mathcal{R}$ such that $p \neq q$, we assume $A_p \neq A_q$.

Conflicting positions. We start by considering networks in which all roads are axis-aligned, such that intersections always involve perpendicular lanes and where routes are such that no turning is required. For each $k \in \{1, \dots, m_r\}$, we define the conflict zone $\mathcal{E}_{rk} = (B_{rk}, E_{rk} + L)$ and define the union

$$\mathcal{E}_r = \bigcup_{k=1}^{m_r} \mathcal{E}_{rk}$$

corresponding to the positions of vehicles for which it occupies an intersection on its path V_r, see also Figure 5.2. By setting $\mathcal{E}_r \equiv \mathcal{E}$, the single intersection problem naturally extends to the network case. We will also write $\mathcal{E}_r(v) := \mathcal{E}_{rk}$ with k such that $v_r(k) = v$ to denote the positions of vehicles from route r that occupy intersection v.

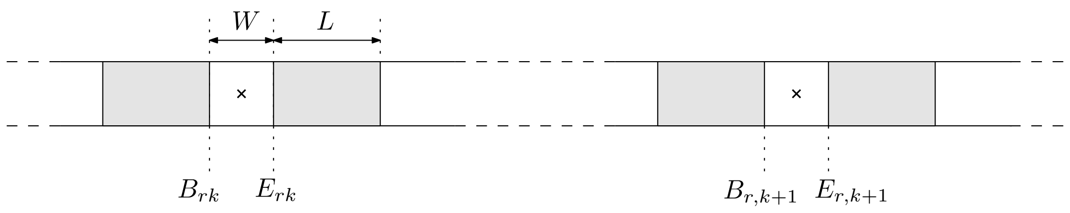


Figure 5.2: Illustration of a route with intersections, marked with a little cross. The length of each intersection is W. The four shaded rectangles illustrate four possible vehicle positions. The conflict areas $\mathcal{E}_{rk} = (B_{rk}, E_{rk} + L)$ indicated the positions of the front of the vehicle that correspond to occupying some intersection.

Vehicle indices. Again, we will write $\mathcal{N}$ to denote all the vehicles in the system and $\mathcal{N}_r$ to denote vehicles on route $r \in \mathcal{R}$. Assumption 5.1 ensures that the relative ordering of vehicles on each route is completely determined by their initial positions. Hence, we write $\mathcal{C}$ for all consecutive pairs of vehicles and $\mathcal{C}_r$ for a specific route. Collisions can now occur at multiple intersections, so we define

$$\mathcal{D}^v := \{\{i, j\} \subset \mathcal{N} : r(i) \neq r(j), v \in V_{r(i)} \cap V_{r(j)}\}, \tag{5.1}$$

which are pairs of vehicles that have conflicting paths at some intersection $v \in V$.

Trajectory optimization problem. The single intersection trajectory optimization problem (T) immediately extends to the network case. Let $s = \{s_i = (x_i^0, v_i^0) : i \in \mathcal{N}\}$ denote the initial state of all vehicles in the system. We consider the same vehicle dynamics as in the single intersection model, written as $x_i \in D(s_i)$. Let $J(x_i)$ be some trajectory performance criterion. Assume that the network (V, A) is given and the system parameters $(L, \mathcal{E}_r, \bar{v}, \omega, \bar{\omega}, t_f)$ are fixed, with t_f some final time and let $x := \{x_i : i \in \mathcal{N}\}$ denote the joint position of all vehicles. Consider the following optimization problem over x.

$$J(s) := \min_x \sum_{i \in \mathcal{N}} J(x_i) \tag{NT}$$

$$\text{s.t. } x_i \in D(s_i) \qquad \text{for all } i \in \mathcal{N}, \tag{NT.1}$$

$$x_i(t) - x_j(t) \geq L \qquad \text{for all } (i, j) \in \mathcal{C}, \tag{NT.2}$$

$$(x_i(t), x_j(t)) \notin \mathcal{E}_{r(i)} \times \mathcal{E}_{r(j)} \qquad \text{for all } \{i, j\} \in \mathcal{D}^v \text{ and } v \in V. \tag{NT.3}$$

Like before, the resulting problem can be numerically solved by a direct transcription method. However, this does not scale to networks of practically relevant sizes, so we will immediately turn our attention to decomposing and reducing the problem into a more manageable formulation.

5.1.1 Bilevel decomposition

The decomposition for the single intersection extends to the network model as follows. Let for each pair (i, v) of some vehicle $i \in \mathcal{N}$ and an intersection $v \in V_{r(i)}$ along its route, let

$$y(i, v) := \inf\{t : x_i(t) \in \mathcal{E}_r(v)\} \quad \text{and} \quad z(i, v) := \sup\{t : x_i(t) \in \mathcal{E}_r(v)\}$$

be the crossing time and exit time, respectively. We also define $\sigma(i, v) = z(i, v) - y(i, v)$, which is the amount of time vehicle i occupies intersection v. Let (NT$'$) denote the problem (NT) with the additional requirement that $y(i, v) < \infty$ and $z(i, v) < \infty$ for all $i \in \mathcal{N}$ and $v \in V_r$, then we claim that (NT$'$) is equivalent to solving the upper-level occupancy time slot scheduling problem

$$
\begin{aligned}
U(s) := \min_{y,z} \quad & \sum_{r \in \mathcal{R}} F(y_r, z_r, s_r) && && \text{(NU)} \\
\text{s.t. } \quad & y(i,v) + \sigma(i,v) \leq y(j,v) \text{ or} && && \\
& y(j,v) + \sigma(j,v) \leq y(i,v), && \text{for all } \{i,j\} \in \mathcal{D}^v \text{ and } v \in V, && \text{(NU.1)} \\
& (y_r, z_r) \in \mathcal{S}_r(s_r), && \text{for all } r \in \mathcal{R}, && \text{(NU.2)}
\end{aligned}
$$

where $F(y_r, z_r, s_r)$ and $\mathcal{S}_r$ are the value function and set of feasible parameters, respectively, of the parametric lower-level trajectory optimization problems

$$
\begin{aligned}
F(y_r, z_r, s_r) = \min_{x_r} \quad & \sum_{i \in \mathcal{N}_r} J(x_i) && && \text{(NL)} \\
\text{s.t. } \quad & x_i(t) \in D_i(s_i), && \text{for } i \in \mathcal{N}_r, && \text{(NL.1)} \\
& x_i(t) - x_j(t) \geq L, && \text{for } (i,j) \in \mathcal{C} \cap \mathcal{N}_r^2, && \text{(NL.2)} \\
& x_i(y(i,v)) = B_r(v), && \text{for } v \in V_r \text{ and } i \in \mathcal{N}_r, && \text{(NL.3)} \\
& x_i(z(i,v)) = E_r(v) + L, && \text{for } v \in V_r \text{ and } i \in \mathcal{N}_r, && \text{(NL.4)}
\end{aligned}
$$

where we use subscript r to group variables according to their associated route. A rigorous proof of this fact would be a trivial extension of the proof presented for Theorem 2.1 and does not provide further insight.

5.1.2 Isolating the occupancy scheduling problem

We make the following assumptions, similar to Section 2.3.3.

Assumption 5.2 (Full speed)**.** Each vehicle starts at full speed $v_i^0 = \bar{v}$ and must enter each intersection $v \in \mathcal{V}_{r(i)}$ along its path at full speed.

Definition 5.1 (Earliest crossing time)**.** Consider some vehicle i on route $r = r(i)$. For some trajectory x_i and initial state $s_i = (x_i^0, \bar{v})$, define the *earliest crossing time* of vehicle i at intersection $v \in V_r$ to be $a(i,v) := (B_r(v) - x_i^0)/\bar{v}$.

Assumption 5.3 (Uniform intersection)**.** We assume that all intersection areas are of the same length, so $E_{rk} - B_{rk} = E_{r'k'} - B_{r'k'}$, for all r, r' and k, k'. Suppose some vehicle drives at full speed as long as it occupies an intersection. In that case, the vehicle occupies the intersection for a duration of precisely $\sigma := (E_{rk} + L - B_{rk})/\bar{V}$, with arbitrary r and k. Again, define the *processing time* $\rho := L/\bar{v}$ and let $\delta := \sigma - \rho$ denote the *switch-over time.*

Definition 5.2 (Delay criteria)**.** Consider the total delay a vehicle experiences along its entire trip through the network, which we define to be $J_d(x_i) = y(i, v_r(m_r)) - a(i, v_r(m_r))$. Alternatively, we could take into account the sum of delay experienced at each intersection, given by $J'_d(x_i) = \sum_{v \in V_{r(i)}} y(i,v) - a(i,v)$.

Using the same reasoning as in Section 2.3.3, the full speed crossing assumption causes the feasibility of the lower-level problems to depend only on y. This means that we have $z(i,v) = y(i,v) + \sigma$. Let $y_r := \{y(i,v) : i \in \mathcal{N}, v \in V_r\}$, then we can focus on

$$
\mathcal{Y}(s_r) := \{y_r : (y_r, y_r + \sigma) \in \mathcal{S}(s_r)\}. \tag{5.2}
$$

Furthermore, it can be shown that each lower-level trajectory optimization problem for a given route $r \in \mathcal{R}$ decomposes into a sequence of problems, each corresponding to two consecutive intersection along V_r. This means that $y_r \in \mathcal{Y}_r$ is essentially equivalent to $y_{(v,w)} \in \mathcal{Y}_{(v,w)}$ for each $(v,w) \in A_r$, where $y_{(v,w)}$ denotes the vector of all variables $y(i,v)$ and $y(i,w)$ for all $i \in \mathcal{N}_r$ and $\mathcal{Y}_{(v,w)}$ denotes the set of values of $y_{(v,w)}$ for which a feasible trajectory part can be found. Like we mentioned in Section 2.3.3, it is non-trivial to formulate a polyhedral characterization of $\mathcal{Y}_{(v,w)}$. Without rigorous proof, we now provide some sufficient conditions:

$$y_r \in \mathcal{Y}(s_r) \iff \begin{cases} a(i,v) \leq y(i,v) & \text{for } i \in \mathcal{N}_r \text{ and } v \in V_r, & (5.3c) \\ y(i,v) + \rho \leq y(j,v) & \text{for } (i,j) \in \mathcal{C}, v \in V_{r(i)=r(j)}, & (5.3d) \\ y(i,v) + d(v,w)/\bar{v} \leq y(i,w) & \text{for } i \in \mathcal{N}, (v,w) \in A_{r(i)}, & (5.3e) \\ y(i + c(v,w), w) \leq y(i,v) & \text{for } i \in \mathcal{N}, (v,w) \in A_r, & (5.3f) \end{cases}$$

where $c(v,w)$ denotes the *capacity* of the lane between intersections v and w, being the maximum number of vehicles for which there is room on this lane. We refer to (5.3e) as the *travel constraint*, since it encodes the minimum time necessary for some vehicle i to travel from node v to the next w. We refer to (5.3f) as the *buffer constraint*, because it essentially says that there must be room in the lane (v,w) before vehicle i can enter. In general, we conjecture that this constraint can be relaxed slightly. Our preliminary investigation of this issue is presented in the next chapter.

5.2 Network scheduling

The discussion above guides us in formulating the extension of the crossing time scheduling problem (C) to networks of intersections. Consider the following scheduling problem.

$$\min_y \sum_{i \in \mathcal{N}} \sum_{v \in V_{r(i)}} y(i,v) \tag{NC}$$

$$\begin{aligned} \text{s.t. } & a(i,v) \leq y(i,v) && \text{for all } i \in \mathcal{N}, v \in V_{r(i)}, && \text{(NC.1)} \\ & y(i,v) + \rho \leq y(j,v) && \text{for all } (i,j) \in \mathcal{C}, v \in V_{r(i)=r(j)}, && \text{(NC.2)} \\ & y(i,v) + \sigma(i,v) \leq y(j,v) \text{ or} && && \\ & y(j,v) + \sigma(j,v) \leq y(i,v), && \text{for all } \{i,j\} \in \mathcal{D}^v \text{ and } v \in V, && \text{(NC.3)} \\ & y(i,v) + d(v,w)/\bar{v} \leq y(i,w) && \text{for } i \in \mathcal{N}, (v,w) \in A_{r(i)}, && \text{(NC.4)} \\ & y(i + c(v,w), w) \leq y(i,v) && \text{for } i \in \mathcal{N}, (v,w) \in A_{r(i)}. && \text{(NC.5)} \end{aligned}$$

After encoding the constraints (NC.3) with binary variables and the big-M trick, we can again reformulate this problem as a mixed-integer linear program.

5.2.1 Constructive scheduling

Without going into detail, we will now discuss some of our preliminary findings with respect to extending the constructive scheduling approach of Chapter 4. Instead of a single vehicle and route order, we must now deal with a separate vehicle order ν^v and route order η^v for each intersection $v \in V$. The constructive scheduling MDP must be extended accordingly. This means that each state is of the form

$$s_t := (\omega, \{\eta^v : v \in V\}) \tag{5.4}$$

where an instance is given by $\omega = (a, \rho, \sigma)$ with $a := \{a(i,v)\}$ denoting all earliest arrival times. Consequently, each action now corresponds to selecting a route-intersection pair (r,v), causing r to be appended to the route order η^v.

Furthermore, the analysis of Section 3.2 can be extended, by considering the disjunctive graph over nodes (i,v) and encoding the constraints of (NC). Note that we now have a new type of conjunctive arc $(i,v) \to (i,w)$ with weight $d(v,w)/\bar{v}$ due to the constraints (NC.4), and conjunctive arc $(i + c(v,w), w) \to (i,v)$ with weight 0 due to the constraints (NC.5). Again, it can be shown that optimal schedules need to satisfy (AS), where $\mathcal{N}^-$ now involves these additional arcs. Furthermore, the definition of crossing time lower bounds naturally extends to $\beta_{iv}(s_t)$, for each vehicle-intersection pair (i,v).

5.2.2 Learning to schedule

We will now briefly discuss some of our preliminary findings on extending the methodology of Chapter 4 to networks.

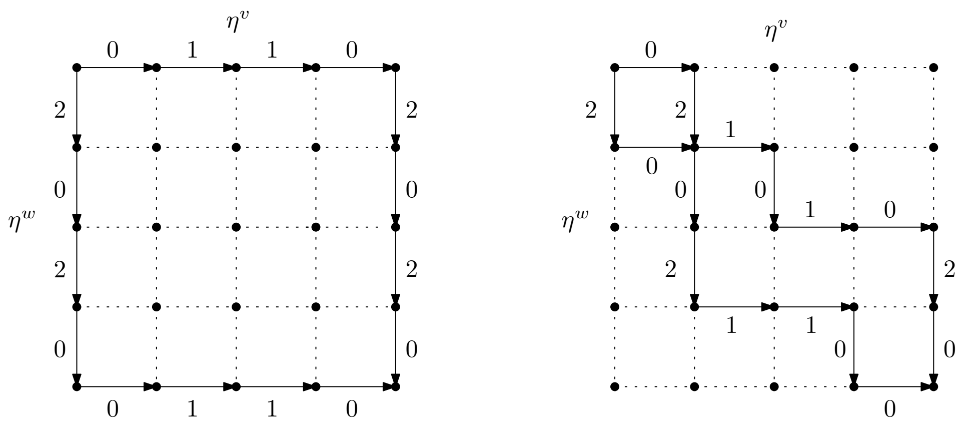


Figure 5.3: When each route order is locally fixed at every intersection, the global crossing sequence is not uniquely determined, because these local sequences may be merged in any order. Suppose we have a tandem of two intersections and a horizontal arrows correspond to taking the next local action at intersection v, a vertical arrows correspond to taking next local action at intersection w. Each valid crossing sequence corresponds to some path from the top-left to the bottom-right corner. Although any such crossing sequence produces the same schedule, it might be that our model fits better on some sequences than others. For example, we might expect that sequences on the boundary of the grid, shown in the left grid, are harder to learn from data than sequences that stay closer to the diagonal, like in the right grid. The intuition is that we need to "look into the future" more to learn the former, while in the latter trajectories, progress in the two intersections is more balanced.

Intersection visit order. In the general class of sequence models, of which our MDP model is a special case (we are essentially learning a sequence of actions), it has been noted before that the order in which inputs or outputs are presented to the model during training has a considerable impact on the final model fit [71]. For our model, we also expect to find this effect, so we investigated the impact of the order in which intersections are visited. First of all, the simple *random* strategy would be to sample some intersection with pending crossings at each step. In the *boundary* strategy, we keep visiting the same intersection until it is done (when it has no pending crossings anymore), then move to some next intersection. When the network of intersections is free of cycles, we could for example follow some topological order. We use the term "boundary" because this strategy produces trajectories along the boundary of the grid in Figure 5.3. In the *alternating* strategy, we keep alternating between intersection to keep the number of scheduled vehicles balanced among them. This produces trajectories that can be understood as being close to the "diagonal" of the grid in Figure 5.3.

Threshold heuristics. It is straightforward to extend the threshold rule to networks of intersections, when assuming a fixed intersection order. Each time some next intersection is visited, we apply the single intersection threshold rule to pick the next route. This is straightforward to do, because we can just consider the disjunctive subgraph induced by the nodes belonging to that intersection to arrive at the single intersection case. Furthermore, the definition of the threshold rule itself does not depend on the network of intersections. This is a desirable property, because it allows us to tune the threshold on small networks and then apply it on larger ones.

Neural constructive heuristic. The neural network parameterization of (4.2.2) can be extended to networks. The model can be best understood as solving a multi-label classification problem, because it needs to provide a distribution over routes at every intersection. Under the imitation learning regime, the training data set $\mathcal{X}$ consists of state-action pairs $(s_{t-1}, (r_t, v_t))$. To obtain these pairs, we again sample a collection of problem instances, which are solved to optimality using a MILP solver, and for each optimal schedule, we backtrack the corresponding optimal route order η^v for each intersection.

Table 5.1: Average optimal objective computed using MILP and using our neural parameterized policy with imitation learning, using one of the three indicated fixed intersection visit orders. Each class of problem instances is identified by the number n of vehicle arrivals per route and the grid network size as cols x rows.

n	size	MILP	time	random (gap)	boundary (gap)	alternate (gap)
5	2x1	57.27	0.06	58.96 (2.95%)	58.72 (2.52%)	58.23 (1.66%)
5	3x1	57.67	0.12	59.77 (3.65%)	59.82 (3.74%)	58.72 (1.83%)
5	3x2	57.35	1.38	60.88 (6.16%)	60.36 (5.25%)	58.82 (2.56%)

From these optimal route orders, we can construct $\mathcal{X}$ in different ways, depending on the specific intersection visit order that is used. The model might become more robust when training on multiple samples of intersections orders per instance. Alternatively, we can consider one of the fixed intersection orders described at the start of this section (random, boundary, alternating).

5.3 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 6

Capacitated lanes

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 formulate and analyze a model for such a *capacitated lane*.

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 conjecture that the answer positive, in the sense that the feasibility question is precisely answered by a system of linear inequalities in terms of the schedule times.

6.1 Model formulation

We start by establishing the notion of a feasible set of trajectories in the capacitated lane.

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

$$\dot{x}(t) \in [0,1] \quad \text{and} \quad \ddot{x}(t) \in [-\omega, \bar{\omega}], \quad \text{for all } t \in [a,b], \tag{6.1}$$

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

$$x(a) = A \;\text{ and }\; x(b) = B \tag{6.2}$$

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

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

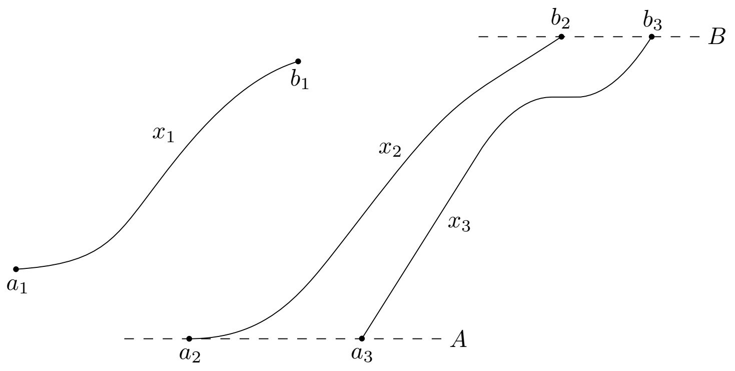


Figure 6.1: Example trajectories $x_1 \in \mathcal{D}[a_1, b_1]$, $x_2 \in D_1[a_2, b_2]$ and $x_3 \in D_2[a_3, b_3]$ for each the three classes of trajectories that are used throughout this chapter (horizontal axis is time, vertical axis is position).

$D_2[a, b] \subset D_1[a, b]$ further induce the second-order boundary conditions

$$\dot{x}(a) = \dot{x}(b) = 1. \tag{6.3}$$

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 6.1 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 6.1, then $x_1 \preceq x_3$ (void) and $x_3 \preceq x_2$, but clearly $x_1 \npreceq x_2$.

Definition 6.1. 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 $x_1, \dots, x_N$ such that

$$x_i \in D_2[a_i, b_i] \qquad \text{for each } i \in \{1, \dots, N\}, \tag{6.4a}$$
$$x_i \preceq x_{i-1} - L \qquad \text{for each } i \in \{2, \dots, N\}. \tag{6.4b}$$

We will refer to (6.4b) 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

$$\min \sum_{i=1}^{N} J(x_i). \tag{6.5}$$

We emphasize again that (6.4a) 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.

6.2 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 (6.4b) 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 6.2. 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 $x \preceq u$.

6.2.1 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 6.2. 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

$$x^1(t) = A + t - a, \tag{6.6}$$

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.

Lemma 6.1. *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

$$x(t) = x(a) + \int_a^t \dot{x}(\tau)\,\mathrm{d}\tau \geq A + \int_a^t \max\{0, 1 - \omega(\tau - a)\}\,\mathrm{d}\tau =: \check{x}(t), \tag{6.7}$$

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

$$x(t) = x(b) - \int_t^b \dot{x}(\tau)\,\mathrm{d}\tau \leq B - \int_t^b \max\{0, 1 - \bar{\omega}(b - \tau)\}\,\mathrm{d}\tau =: \hat{x}(t), \tag{6.8}$$

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 6.2. *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. *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.*

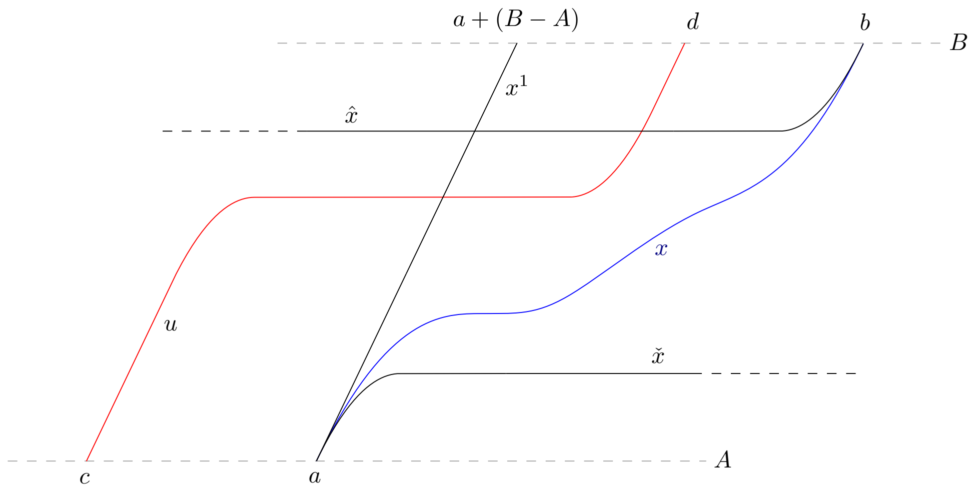


Figure 6.2: Illustration of the four bounding trajectories $u, x^1, \hat{x}, \check{x}$ that bound feasible trajectories from above and below. We also drew an example of a feasible trajectory x in blue. The horizontal axis represents time and the vertical axis corresponds to the position on the lane, so the vertical dashed grey lines correspond to the start and end of the lane.

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 (6.7) and (6.8) 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 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 6.1. 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 6.1 as requiring enough space in the lane such that there is at least one *waiting position*. We will return to this observation in Section 6.3. (check that we do this)

6.2.2 Sufficient conditions

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

Theorem 6.1 (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 6.1, 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 6.1 and Lemma 6.2 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

$$\gamma(t) := \min\{u(t), \hat{x}(t), x^1(t)\}. \tag{6.9}$$

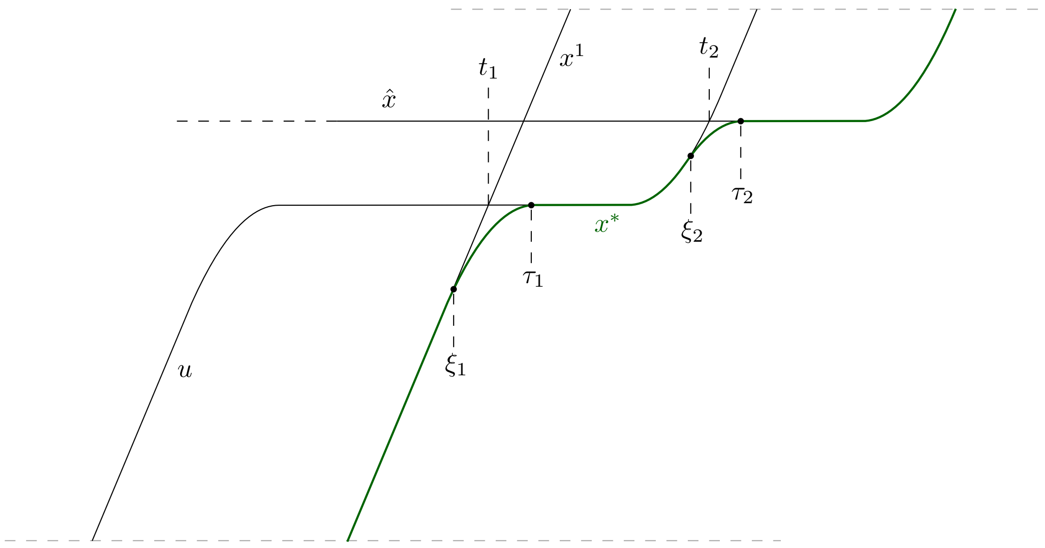


Figure 6.3: The minimum boundary γ, induced by three upper boundaries u, $\hat{x}$ and x^1, is smoothened around time t_1 and t_2, where the derivative is discontinuous, to obtain the smooth optimal trajectory x^*, drawn in green. The times ξ_i and τ_i correspond to the start and end of the connecting deceleration as defined in Section 6.2.4.

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 6.3. 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 $i \in \{0,\dots,n\}$, and for which the one-sided limits of $\dot{\mu}$ satisfy

$$\dot{\mu}(t_i^-) := \lim_{t\uparrow t_i} \dot{\mu}(t) > \lim_{t\downarrow t_i} \dot{\mu}(t) =: \dot{\mu}(t_i^+), \tag{6.10}$$

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

Under the conditions of Theorem 6.1, it is not difficult to see from Figure 6.2 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 6.3. We will argue that this procedure can be repeated as many times as necessary to smoothen out every discontinuity.

In Section 6.2.3, 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 6.2.4. 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 6.2.5.

6.2.3 Deceleration boundary

Recall the derivation of $\check{x}$ in equation (6.7) 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

$$\dot{x}(t) \geq \max\{0,\, \min\{1,\, \dot{x}(\xi) - \omega(t-\xi)\}\} =: \{\dot{x}(\xi) - \omega(t-\xi)\}_{[0,1]}, \tag{6.11}$$

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

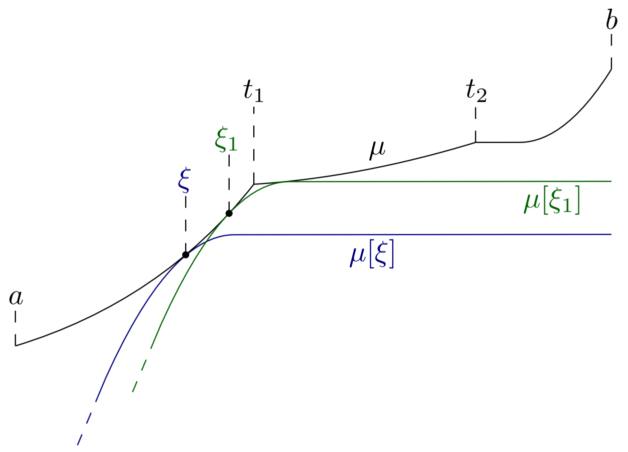


Figure 6.4: Illustration of some piecewise trajectory $\mu \in \mathcal{P}[a,b]$ with a discontinuous derivative at times t_1 and t_2. Furthermore, the figure shows some arbitrary deceleration boundary $\mu[\xi]$ at time ξ in blue and the unique connecting deceleration $\mu[\xi_1]$ to the cover discontinuity 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 (6.9), but please note that the class of piecewise trajectories $\mathcal{P}[a,b]$ is just slightly more general than necessary for our current purposes.

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

$$x(t) = x(\xi) + \int_\xi^t \dot{x}(\tau)\,\mathrm{d}\tau \geq x(\xi) + \int_\xi^t \{\dot{x}(\xi) - \omega(\tau - \xi)\}_{[0,1]}\,\mathrm{d}\tau =: x[\xi](t), \tag{6.12}$$

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 (6.12) as

$$x^-[p,v,\xi](t) := p + \int_\xi^t \{v - \omega(\tau - \xi)\}_{[0,1]}\,\mathrm{d}\tau, \tag{6.13}$$

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

$$x^-[p,v,\xi](t) = p + \begin{cases} (1-v)^2/(2\omega) + (t - \xi) & \text{for } t \leq \delta_1, \\ v(t-\xi) - \omega(t-\xi)^2/2 & \text{for } t \in [\delta_1, \delta_0], \\ v^2/(2\omega) & \text{for } t \geq \delta_0. \end{cases} \tag{6.14}$$

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 $x^-[p,v,\xi] \in \mathcal{D}(-\infty,\infty)$.

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 6.3. 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

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

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 6.1. *Please note that we cannot just replace x with μ in inequality (6.12) 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 $t \in I$.*

6.2.4 Smoothing procedure

Let $\mu \in \mathcal{P}[a, b]$ be some piecewise trajectory and let $a = t_0 < \dots < t_{n+1} = b$ denote the subdivision as in Definition 6.3. 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 6.4 for an example. The construction relies on the following technical assumption.

Assumption 6.2. Throughout the following discussion, we assume $\mu \succeq \mu[a]$ and $\mu \succeq \mu[b]$.

Touching. Recall Remark 6.1, 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

$$d(\xi) := \min_{t \in [t_1, b]} \mu(t) - \mu[\xi](t). \tag{6.15}$$

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 F.1). Observe that $d(a) \geq 0$, because $\mu \succeq \mu[a]$ by Assumption 6.2. 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

$$X := \{\xi \in [a, t_1) : d(\xi) = 0\} \tag{6.16}$$

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 F.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 (6.14). 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

$$\frac{\partial}{\partial \xi} \mu[\xi](t) = \dot{\mu}(\xi) + \begin{cases} \ddot{\mu}(\xi)(\dot{\mu}(\xi) - 1)/\omega - 1 & \text{for } t \leq \delta_1(\xi), \\ \ddot{\mu}(\xi)(t - \xi) - \dot{\mu}(\xi) + \omega(t - \xi) & \text{for } t \in [\delta_1(\xi), \delta_0(\xi)], \\ \ddot{\mu}(\xi)\dot{\mu}(\xi)/\omega & \text{for } t \geq \delta_0(\xi). \end{cases} \tag{6.17}$$

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:

$$\frac{\partial}{\partial \xi} \mu[\xi](t) = (\ddot{\mu}(\xi) + \omega)(t - \xi) \geq 0 \qquad \text{for } t \in [\delta_1(\xi), \delta_0(\xi)], \tag{6.18a}$$

$$\frac{\partial}{\partial \xi} \mu[\xi](t) \geq \dot{\mu}(\xi) + (-\omega)\dot{\mu}(\xi)/\omega = 0 \qquad \text{for } t \geq \delta_0(\xi). \tag{6.18b}$$

This concludes the argument for X being a closed interval.

Assuming ξ to be fixed, observe that there is equality in (6.18a) for some $t \in [\delta_1(\xi), \delta_0(\xi)]$ if and only if there is equality in (6.18b) 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 6.5 for an example of this case. This observation is key to the remaining uniqueness argument.

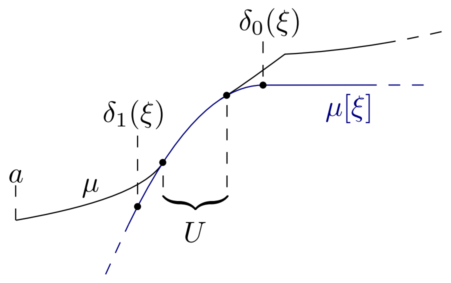


Figure 6.5: Example of a piecewise trajectory μ with a part of full deceleration over some interval U such that any choice of $\xi \in U$ produces the same deceleration boundary $\mu[\xi]$, which naturally coincides with μ on U.

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

$$\frac{\partial}{\partial \xi}\mu[\xi](t') > 0 \text{ for all } \xi \in U. \tag{6.19}$$

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 (6.19) 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

$$\dot{\mu}(t_1^+) > \dot{\mu}[\xi_0](t_1^+) = \dot{\mu}[\xi_0](t_1), \tag{6.20}$$

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

$$\dot{\mu}[b](t) \leq \dot{\mu}[\xi_0](t) \text{ for } t \leq b, \tag{6.21}$$

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 6.2.

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

$$\mu[\xi_0](t) \leq \mu(t) \quad \text{for} \quad t \in [\xi_0, \tau_1], \tag{6.22a}$$

$$\dot{\mu}[\xi_0](\xi_0) = \dot{\mu}(\xi_0) \quad \text{and} \quad \dot{\mu}[\xi_0](\tau_1) = \dot{\mu}(\tau_1). \tag{6.22b}$$

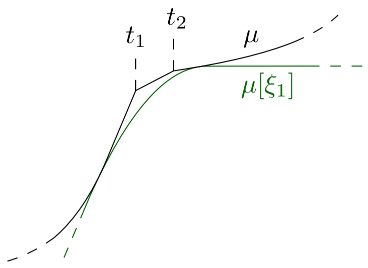


Figure 6.6: Part of a piecewise trajectory μ on which a single connecting deceleration covers the two discontinuities at t_1 and t_2 at once.

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

$$\mu_1(t) = \begin{cases} \mu(t) & \text{for } t \in [a, \xi_1] \cup [\tau_1, b], \\ \mu[\xi_1](t) & \text{for } t \in [\xi_1, \tau_1]. \end{cases} \tag{6.23}$$

From the way we constructed $\mu[\xi_1]$, it follows from (6.22) 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 6.6. 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 6.2, so that we can keep repeating the exact same process, obtaining connecting decelerations $(\xi_2, \tau_2), (\xi_3, \tau_3), \dots$ and the corresponding piecewise trajectories $\mu_2, \mu_3, \dots$ 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 6.1. Let us now return to the minimum boundary γ defined in (6.9). From Figure 6.2 and the conditions of Theorem 6.1, 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. 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 6.1.

6.2.5 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 6.3. *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. *The above upper boundary property has the following interesting consequence if we extend the single vehicle problem to the optimal control of maximizing the haste criterion, which was defined as*

$$J(x_i) = \int_{a_i}^{b_i} -x_i(t)\,\mathrm{d}t. \tag{6.24}$$

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 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*

$$\int_a^b x(t)\,\mathrm{d}t \leq \int_a^b \gamma^*(t)\,\mathrm{d}t \tag{6.25}$$

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

$$\max_{x \in D_2[a,b]} J(x) \;\; \text{such that} \;\; x \preceq u. \tag{6.26}$$

6.3 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 $a = (a_1, a_2, \dots, a_N)$ and $b = (b_1, b_2, \dots, b_N)$, which are assumed to be ordered as $a_1 \leq \dots \leq a_N$ and $b_1 \leq \dots \leq b_N$. For brevity, we will write $x \in D_2^N[a,b]$ to denote the vector $x = (x_1, \dots, x_N)$ 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

$$x_i \in D_2[a_i, b_i] \quad \text{for each } i \in \{1, \dots, N\}, \tag{6.27a}$$
$$x_i \preceq x_{i-1} - L \quad \text{for each } i \in \{2, \dots, N\}. \tag{6.27b}$$

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 6.3. Assume that vehicle lengths are limited by $L < B - A$.

Now consider the safe following constraints (6.27b). 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 6.1. It becomes clear from Figure 6.7 that inequality (6.27b) 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

$$y^{-1}(p) = \inf\{t \in [\alpha, \beta] : y(t) = p\}. \tag{6.28}$$

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

$$\bar{u}(t) = \begin{cases} u(t) - L & \text{for } t \in [u^{-1}(A+L), d], \\ B - L + t - d & \text{for } t \in [d, d+L]. \end{cases} \tag{6.29}$$

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

Lemma 6.4 (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.*

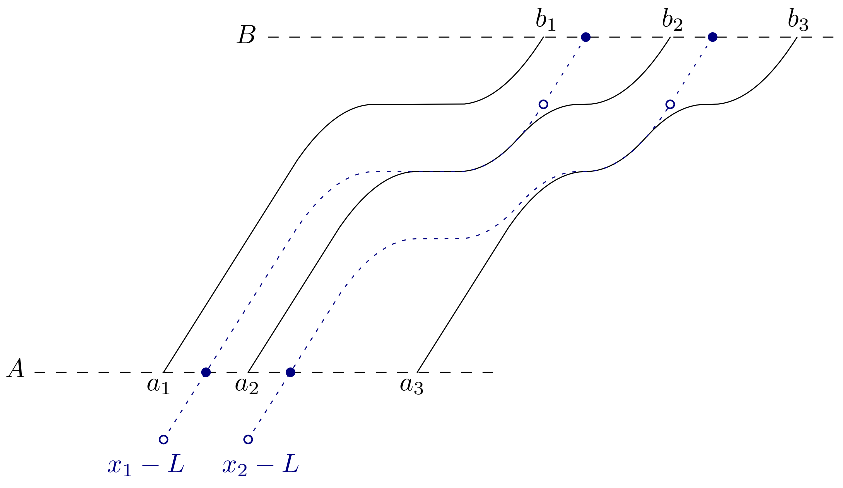


Figure 6.7: Optimal trajectories x_i for three vehicles. The dotted blue trajectories between the little open circles illustrates the safe following constraints (6.27b). The dotted blue trajectories between the solid dots are the following boundaries $\bar{x}_2 \in \bar{D}[\bar{a}_2, \bar{b}_2]$ and $\bar{x}_3 \in \bar{D}[\bar{a}_3, \bar{b}_3]$.

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 6.5 (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 6.4 to $u - L$ (using Assumption 6.3 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 6.6. *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 6.5. Equivalence of (C2) and (C3) follows from a straightforward application of Theorem 6.1 by setting $x = x_i$ and $u = \bar{x}_{i-1}$ for each $i \in \{2, \dots, N\}$. □

Conjecture: simpler conditions. Our next goal 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 want to show that the statements of Lemma 6.6 above are further equivalent to:

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

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

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

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

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 need some better understanding of the smoothing procedure, which means that try to derive explicit formulas for finding the touching times ξ and τ to obtain optimal trajectories under the haste objective. This will be the subject of further research.

6.4 Notes and references

The analysis of feasibility conditions in this chapter is very much related to the proof of the safety guarantee in [48], 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 7

Conclusion and discussion

The coordination of autonomous vehicles at intersections is a challenging optimization problem that has been approached from many perspectives across different research communities—each bringing their own modeling paradigms and solution methodologies. In this thesis, we have taken a deliberately simplified yet rigorous mathematical optimization perspective, focusing on understanding the fundamental computational challenges that arise even in idealized settings. By abstracting away from many practical complexities—such as unobservability, decentralized control, and mixed traffic—we have been able to isolate and analyze a core computational difficulty: *the joint optimization of discrete crossing order decisions and continuous-time vehicle trajectories.*

Starting with a single intersection model, we identified a key structural property of this problem. A central theme has been the decomposition of the joint optimization into more manageable components: an upper-level scheduling problem that determines when vehicles cross intersections, and lower-level trajectory generation problems that compute the actual vehicle paths. Under specific assumptions—particularly that vehicles cross intersections at full speed and that delay minimization is the primary objective—we showed that this decomposition becomes proper, reducing an infinite-dimensional trajectory optimization problem to a *combinatorial scheduling problem.*

This reduction opened the door to applying both classical optimization techniques and modern machine learning approaches. We demonstrated how mixed-integer linear programming can solve small to medium-sized instances optimally, while learning-based constructive heuristics offer a promising path forward. The conceptual framework developed for the single intersection case—particularly the formulation as a MDP for learning-based scheduling—provides a foundation that could scale to networks of intersections, though this extension introduces additional challenges that merit further investigation. Below, we reflect on the contributions of this work, discuss its limitations, and outline directions for future research that could help bridge the gap between our idealized models and real-world autonomous traffic management systems.

7.1 Contributions

Motion planning model with direct transcription baseline. We formulated a simple model of a single intersection and provided a simple extension to networks of intersections. We formulated safe motion planning as a problem of trajectory optimization with collision-avoidance constraints. Within this framework, we studied two approaches to safe motion planning. First of all, we showed that the collision-avoidance constraints can be directly encoded within the context of direct transcription to an integer program, yielding an immediate solution for safe motion planning in the single intersection, as well as networks of intersections.

Although this method is easy to implement and it guarantees to find optimal solutions (up to time discretization) that are provably safe, it scales very poorly to larger instances. Hence, without further changes, it is not a good candidate for enabling large-scale coordination in networks. This issue is the main motivation for our concrete proposals.

Decomposition and improved integer programming. As an alternative to direct transcription, we discussed the technique of separating the decisions regarding occupancy time slot reservations. Although the decomposition itself does not immediately provide better algorithms, we stress that it is a useful conceptual tool, as previously demonstrated by data-driven approximation methods [28, 30]. Instead of keeping the optimality guarantees, we choose to make some simplifying assumptions in favor of a proper decomposition, which then leads to a provably safe coordination scheme that has far better potential for scaling up.

Based on the proper decomposition, we can focus on solving the resulting crossing time scheduling problem. Integer programming provides a readily available and mature solution. It can handle single intersection problems fairly well, but computing optimal solutions becomes expensive (>1 minute of computation time) for more than 20 vehicles. Nevertheless, modern solvers provide a pretty good heuristic when computation time is limited. To further improve the applicability of integer programming, we show how a particular structure in optimal schedules leads to the natural formulation of three types of cutting planes. We show that one type of cuts clearly improves the running time and increases the size of instances that we can solve roughly by a factor 4.

Hand-crafted and machine-learned constructive heuristics. As an alternative to integer programming, we explored the feasibility of using a constructive scheduling approach for solving the crossing time problem. We defined a Markov decision process that simulates a constructive step-by-step schedule building process for the single intersection problem. There exists a redundancy in this natural formulation, which led us to propose a state abstraction function, resulting in a significant reduction of the state space.

Policies based on a very elementary rule already provide a very good heuristics that is computationally very cheap, demonstrating the effectiveness of the state reduction. Furthermore, we illustrate the potential for machine-learned heuristics by proposing a simple neural policy architecture, where the sequence of upcoming arrivals of each lane is embedded by a recurrent neural network. We demonstrated two ways of fitting the parameters of this class of policies. When instance sizes still allow us to compute optimal schedules, it seems a good idea to use the imitation learning setting. For tackling larger instance sizes, the case for reinforcement learning becomes very strong.

To extend the constructive scheduling method to the network problem, we extended the action space to also include the decision at which intersection to take the next step. This introduces additional redundancy. In the imitation learning regime, we observed that "order matters" and confirm our hypothesis that keeping the amount of progress—in terms of scheduling steps taken—balanced across intersection, leads to better convergence and final model performance.

7.2 Limitations and further work

Beyond proper decomposition: true joint optimization. We believe that full speed crossing and the delay objective are sensible assumptions from a practitioner's point of view. However, from a methodological standpoint, our setting is not very satisfying. In the general setting, the trajectory optimization problem does not decompose: the upper- and lower-level problems must be considered simultaneously. We identify the following two issues:

1. 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 how to leverage sampling-based methods to approximate these constraints on the time schedule. In other words, can we approximate the space of feasible time schedules, with guarantees?

2. In general, the optimization objective is not just a linear function of the upper-level decisions variables. We would like to see how an data-driven approximation of the objective, similar to the proposal in [28, 30], could be extended to network scheduling.

Discovering structure in optimal network schedules. The major limitation of our current network model is the fact that vehicle routes are fully independent, meaning that traffic can only follow predefined straight paths. We made this assumption mainly out of computational convenience and for the sake of a relatively simpler implementation and presentation. Our intuition is that optimal network schedules behave *somewhat like the platoon preservation theorem* for single intersections. We say “somewhat”, because it is quite straightforward to find counterexamples to the platoon perservation theorem in small networks with only a couple of intersections.

When the independent route assumption is relaxed, the combinatorial space of crossing order grows significantly, so following the above intuitive argument, this means that a lot of feasible solutions are introduced that are clearly suboptimal. It would be great if this intuitive statement could be captured by a formal statement, but it is probably too fuzzy. Rather, we would like to push the machine learning perspective, because we believe this example is an excellent candidate for this paradigm.

Connection with RL for job shop scheduling. Very related to our investigation is the huge amount of work that has been done on neural reinforcement learning methods for job shop scheduling, which is very related to our network scheduling problem. Hence, this is a highly suggested direction of further research and collaboration.

Bibliography

[1] Karen Aardal, Cor Hurkens, and Jan Karel Lenstra. "Jacques Benders and His Decomposition Algorithm". In: *Operations Research Letters* 63 (Nov. 2025), p. 107361. (Visited on 09/06/2025).

[2] Brandon Amos and J. Zico Kolter. *OptNet: Differentiable Optimization as a Layer in Neural Networks*. Dec. 2021. arXiv: 1703.00443 [cs]. (Visited on 04/09/2025).

[3] 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. AAAI Press, Jan. 2010, pp. 2–9. (Visited on 09/06/2025).

[4] Richard Bellman. *Dynamic Programming*. Princeton, NJ: Princeton Univ. Pr, 1984.

[5] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. *Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon*. Mar. 2020. arXiv: 1811.06128 [cs, stat]. (Visited on 09/24/2023).

[6] Dimitri Bertsekas. *Dynamic Programming and Optimal Control: Volume I*. Vol. 4. Athena scientific, 2012.

[7] Suresh Bolusani et al. *The SCIP Optimization Suite 9.0*. Technical Report. Optimization Online, Feb. 2024.

[8] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer Programming*. Vol. 271. Graduate Texts in Mathematics. Cham: Springer International Publishing, 2014. (Visited on 08/27/2025).

[9] Hanjun Dai et al. *Learning Combinatorial Optimization Algorithms over Graphs*. Feb. 2018. arXiv: 1704.01665 [cs]. (Visited on 09/17/2025).

[10] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. "Benchmarks for Shop Scheduling Problems". In: *European Journal of Operational Research* 109.1 (1998), pp. 137–141.

[11] Priya L. Donti, David Rolnick, and J. Zico Kolter. *DC3: A Learning Method for Optimization with Hard Constraints*. Apr. 2021. arXiv: 2104.12225. (Visited on 11/13/2024).

[12] K. Dresner and P. Stone. "A Multiagent Approach to Autonomous Intersection Management". In: *Journal of Artificial Intelligence Research* 31 (Mar. 2008), pp. 591–656. (Visited on 11/28/2024).

[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*. July 2004, pp. 530–537. (Visited on 09/06/2025).

[14] Ding-Zhu Du et al. *Introduction to Combinatorial Optimization*. Vol. 196. Springer Optimization and Its Applications. Cham: Springer International Publishing, 2022. (Visited on 08/27/2025).

[15] Saba Fatima, Chieh Hsiu Lee, and Andrew L Dannenberg. "Equity Issues Associated with the Widespread Implementation of Autonomous Vehicles". In: *Oxford Open Infrastructure and Health* 2 (Apr. 2024), ouae002. (Visited on 10/25/2025).

[16] Maxime Gasse et al. *Exact Combinatorial Optimization with Graph Convolutional Neural Networks*. Oct. 2019. arXiv: 1906.01629 [cs]. (Visited on 09/16/2025).

[17] R. L. Graham et al. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey". In: *Annals of Discrete Mathematics*. Ed. by P. L. Hammer, E. L. Johnson, and B. H. Korte. Vol. 5. Discrete Optimization II. Elsevier, Jan. 1979, pp. 287–326. (Visited on 10/23/2023).

[18] Alex Graves, Greg Wayne, and Ivo Danihelka. *Neural Turing Machines*. Dec. 2014. arXiv: 1410.5401 [cs]. (Visited on 01/21/2025).

[19] Qiangqiang Guo, Li Li, and Xuegang (Jeff) Ban. "Urban Traffic Signal Control with Connected and Automated Vehicles: A Survey". In: *Transportation Research Part C: Emerging Technologies* 101 (Apr. 2019), pp. 313–334. (Visited on 08/26/2025).

[20] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2024.

[21] Andreas Hadjigeorgiou and Stelios Timotheou. *Energy Consumption Optimization for Autonomous Vehicles via Positive Control Input Minimization*. June 2025. arXiv: 2506.04685 [math]. (Visited on 10/25/2025).

[22] Lanshan Han et al. "A Unified Numerical Scheme for Linear-Quadratic Optimal Control Problems with Joint Control and State Constraints". In: *Optimization Methods and Software* 27.4-5 (Oct. 2012), pp. 761–799. (Visited on 08/31/2025).

[23] Richard F. Hartl, Suresh P. Sethi, and Raymond G. Vickson. "A Survey of the Maximum Principles for Optimal Control Problems with State Constraints". In: *SIAM Review* 37.2 (1995), pp. 181–218. JSTOR: 2132823. (Visited on 02/02/2025).

[24] 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*. Sept. 2011, pp. 4581–4586. (Visited on 09/06/2025).

[25] Hilary. *Making Automated Vehicles Work for Better Transport Services: Regulating for Impact*. https://www.itf-oecd.org/automated-vehicles-better-transport-services. Text. July 2022. (Visited on 10/25/2025).

[26] Shan Huang, Adel W. Sadek, and Yunjie Zhao. "Assessing the Mobility and Environmental Benefits of Reservation-Based Intelligent Intersections Using an Integrated Simulator". In: *IEEE Transactions on Intelligent Transportation Systems* 13.3 (Sept. 2012), pp. 1201–1214. (Visited on 09/06/2025).

[27] Robert Hult. *Optimization Based Coordination Strategies for Connected and Autonomous Vehicles*. Göteborg: Chalmers University of Technology, 2019.

[28] Robert Hult et al. "An Approximate Solution to the Optimal Coordination Problem for Autonomous Vehicles at Intersections". In: *2015 American Control Conference (ACC)*. Chicago, IL, USA: IEEE, July 2015, pp. 763–768. (Visited on 04/23/2024).

[29] Robert Hult et al. *Optimal Coordination of Three Cars Approaching an Intersection*.

[30] Robert Hult et al. *Technical Report: Approximate Solution to the Optimal Coordination Problem for Autonomous Vehicles at Intersections*. Tech. rep.

[31] Purva Joshi, Marko Boon, and Sem Borst. "Trajectories and Platoon-forming Algorithm for Intersections with Heterogeneous Autonomous Traffic". In: *ACM Journal on Autonomous Transportation Systems* 2.3 (Sept. 2025), pp. 1–32. (Visited on 08/26/2025).

[32] Matthew Kelly. "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation". In: *SIAM Review* 59.4 (Jan. 2017), pp. 849–904. (Visited on 08/27/2025).

[33] Mohammad Khayatian et al. "A Survey on Intersection Management of Connected Autonomous Vehicles". In: *ACM Transactions on Cyber-Physical Systems* 4.4 (Oct. 2020), pp. 1–27. (Visited on 11/25/2024).

[34] Mohammad Khayatian et al. "Crossroads+: A Time-aware Approach for Intersection Management of Connected Autonomous Vehicles". In: *ACM Transactions on Cyber-Physical Systems* 4.2 (Apr. 2020), pp. 1–28. (Visited on 11/25/2024).

[35] Stefan Kojchev. *On Optimization Based Coordination of Automated Vehicles in Confined Sites*. Göteborg: Chalmers University of Technology, 2024.

[36] Wouter Kool, Herke van Hoof, and Max Welling. *Attention, Learn to Solve Routing Problems!* Feb. 2019. arXiv: 1803.08475 [stat]. (Visited on 04/01/2025).

[37] Leonardo Lamorgese and Carlo Mannino. "A Noncompact Formulation for Job-Shop Scheduling Problems in Traffic Management". In: *Operations Research* 67.6 (Nov. 2019), pp. 1586–1609. (Visited on 10/12/2023).

[38] Michael W. Levin and David Rey. "Conflict-Point Formulation of Intersection Control for Autonomous Vehicles". In: *Transportation Research Part C: Emerging Technologies* 85 (Dec. 2017), pp. 528–547. (Visited on 09/06/2025).

[39] Jinjue Li et al. "A Survey on Urban Traffic Control under Mixed Traffic Environment with Connected Automated Vehicles". In: *Transportation Research Part C: Emerging Technologies* 154 (Sept. 2023), p. 104258. (Visited on 09/06/2025).

[40] Lihong Li, Thomas J Walsh, and Michael L Littman. "Towards a Unified Theory of State Abstraction for MDPs". In: ().

[41] Zhenning Li et al. "Temporal-Spatial Dimension Extension-Based Intersection Control Formulation for Connected and Autonomous Vehicle Systems". In: *Transportation Research Part C: Emerging Technologies* 104 (July 2019), pp. 234–248. (Visited on 01/26/2024).

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

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

[44] Andrea Lodi and Giulia Zarpellon. "On Learning and Branching: A Survey". In: *TOP* 25.2 (July 2017), pp. 207–236. (Visited on 11/09/2024).

[45] Carlo Mannino and Giorgio Sartor. "The Path&Cycle Formulation for the Hotspot Problem in Air Traffic Management". In: (2018), 11 pages. (Visited on 10/12/2023).

[46] Stefano Mariani, Giacomo Cabri, and Franco Zambonelli. "Coordination of Autonomous Vehicles: Taxonomy and Survey". In: *ACM Computing Surveys* 54.1 (Jan. 2022), pp. 1–33. (Visited on 09/24/2023).

[47] Nina Mazyavkina et al. *Reinforcement Learning for Combinatorial Optimization: A Survey.* Dec. 2020. arXiv: 2003.03600. (Visited on 11/13/2024).

[48] David Miculescu and Sertac Karaman. *Polling-Systems-Based Autonomous Vehicle Coordination in Traffic Intersections with No Traffic Signals.* July 2016. arXiv: 1607.07896 [cs, math]. (Visited on 12/05/2023).

[49] Youngjae Min, Anoopkumar Sonar, and Navid Azizan. *Hard-Constrained Neural Networks with Universal Approximation Guarantees.* Oct. 2024. arXiv: 2410.10807. (Visited on 11/13/2024).

[50] Mohammad Naiseh et al. "Trust, Risk Perception, and Intention to Use Autonomous Vehicles: An Interdisciplinary Bibliometric Review". In: *AI & SOCIETY* 40.2 (Feb. 2025), pp. 1091–1111. (Visited on 10/25/2025).

[51] B. Padmaja et al. "Exploration of Issues, Challenges and Latest Developments in Autonomous Cars". In: *Journal of Big Data* 10.1 (May 2023), p. 61. (Visited on 10/25/2025).

[52] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Cham: Springer International Publishing, 2016. (Visited on 10/18/2023).

[53] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley Series in Probability and Statistics v.414. Hoboken: John Wiley & Sons, Inc, 2009.

[54] Balaraman Ravindran and Andrew G Barto. *Symmetries and Model Minimization in Markov Decision Processes.* 2001.

[55] David Rey, Michael W. Levin, and Vinayak V. Dixit. *Online Incentive-Compatible Mechanisms for Traffic Intersection Auctions.* Oct. 2020. arXiv: 2006.01382 [cs]. (Visited on 09/06/2025).

[56] Federico Rossi et al. "Routing Autonomous Vehicles in Congested Transportation Networks: Structural Properties and Coordination Algorithms". In: *Autonomous Robots* 42.7 (Oct. 2018), pp. 1427–1442. (Visited on 09/24/2025).

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

[58] Muhammed O. Sayin et al. "Information-Driven Autonomous Intersection Control via Incentive Compatible Mechanisms". In: *IEEE Transactions on Intelligent Transportation Systems* 20.3 (Mar. 2019), pp. 912–924. (Visited on 09/03/2025).

[59] John Schulman et al. *Proximal Policy Optimization Algorithms.* Aug. 2017. arXiv: 1707.06347 [cs]. (Visited on 12/06/2024).

[60] Igor G. Smit et al. *Graph Neural Networks for Job Shop Scheduling Problems: A Survey.* 2024. (Visited on 11/15/2024).

[61] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Second edition. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2018.

[62] Remi Tachet et al. “Revisiting Street Intersections Using Slot-Based Systems”. In: *PLOS ONE* 11.3 (Mar. 2016), e0149607. (Visited on 09/24/2025).

[63] Éric Taillard. “Benchmarks for Basic Scheduling Problems”. In: *European Journal of Operational Research* 64 (1993), pp. 278–285.

[64] Pavankumar Tallapragada and Jorge Cortes. *Distributed Control of Vehicle Strings under Finite-Time and Safety Specifications*. July 2017. arXiv: 1701.03580. (Visited on 11/25/2024).

[65] Pavankumar Tallapragada and Jorge Cortés. *Hierarchical-Distributed Optimized Coordination of Intersection Traffic*. Jan. 2017. arXiv: 1601.00246 [cs, math]. (Visited on 04/23/2024).

[66] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. *Reinforcement Learning for Integer Programming: Learning to Cut*. July 2020. arXiv: 1906.04859 [cs, math, stat]. (Visited on 09/25/2023).

[67] Pierre Tassel, Martin Gebser, and Konstantin Schekotihin. *A Reinforcement Learning Environment For Job-Shop Scheduling*. Apr. 2021. arXiv: 2104.03760 [cs]. (Visited on 09/27/2023).

[68] 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. arXiv: 2306.05747. (Visited on 11/15/2024).

[69] R. W. Timmerman and M. A. A. Boon. “Platoon Forming Algorithms for Intelligent Street Intersections”. In: *Transportmetrica A: Transport Science* 17.3 (Feb. 2021), pp. 278–307. (Visited on 01/10/2023).

[70] *Trajectory Optimization*. https://underactuated.mit.edu/trajopt.html. (Visited on 08/27/2025).

[71] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. *Order Matters: Sequence to Sequence for Sets*. Feb. 2016. arXiv: 1511.06391. (Visited on 11/13/2024).

[72] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. *Pointer Networks*. Jan. 2017. arXiv: 1506.03134 [cs, stat]. (Visited on 02/21/2024).

[73] Christian Vitale, Panayiotis Kolios, and Georgios Ellinas. “Autonomous Intersection Crossing With Vehicle Location Uncertainty”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.10 (Oct. 2022), pp. 17546–17561. (Visited on 10/25/2025).

[74] Dawei Wang et al. “Learning to Control and Coordinate Mixed Traffic Through Robot Vehicles at Complex and Unsignalized Intersections”. In: *The International Journal of Robotics Research* 44.5 (Apr. 2025), pp. 805–825. arXiv: 2301.05294 [cs]. (Visited on 08/28/2025).

[75] Michael I. -C. Wang, Charles H. -P. Wen, and H. Jonathan Chao. “Assessing the Impact of Communication Delays for Autonomous Intersection Management Systems”. In: *Vehicular Communications* 49 (Oct. 2024), p. 100829. (Visited on 10/25/2025).

[76] Xun Yang et al. “Autonomous Driving under V2X Environment: State-of-the-Art Survey and Challenges”. In: *Intelligent Transportation Infrastructure* 1 (Sept. 2022), liac020. (Visited on 10/25/2025).

[77] Ran Yu et al. *Uncertainty-Aware Safety-Critical Decision and Control for Autonomous Vehicles at Unsignalized Intersections*. July 2025. arXiv: 2505.19939 [cs]. (Visited on 10/25/2025).

[78] Cong Zhang et al. *Deep Reinforcement Learning Guided Improvement Heuristic for Job Shop Scheduling*. Feb. 2024. arXiv: 2211.10936. (Visited on 11/15/2024).

[79] Cong Zhang et al. *Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning*. Oct. 2020. arXiv: 2010.12367 [cs, stat]. (Visited on 09/27/2023).

[80] Weiming Zhao, Ronghui Liu, and Dong Ngoduy. “A Bilevel Programming Model for Autonomous Intersection Control and Trajectory Planning”. In: *Transportmetrica A: Transport Science* (Jan. 2021). (Visited on 11/25/2024).

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 2.2. 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 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

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

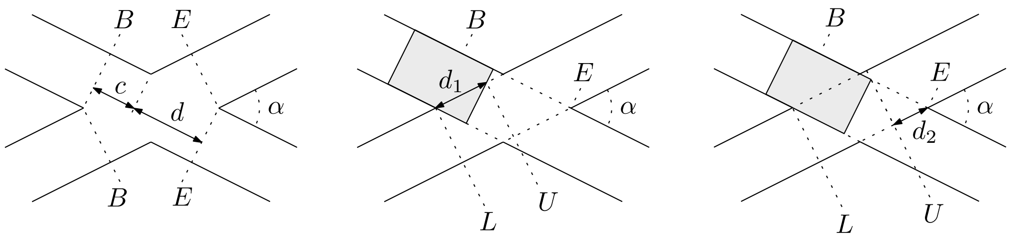


Figure A.1: Sketches to derive the feasible configurations of two vehicles in the intersecting routes model. Using some elementary trigonometry, the distances in the first figure can be shown to be $c = W/\tan(\alpha)$ and $d = W/\sin(\alpha)$. Furthermore, observe that we have $(x_i - B)/d_1 = \cos(\alpha)$ 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.

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 [52, 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 [52, 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 $(1, 1) \rightarrow (2, 1) \rightarrow (3, 1)$ and $(3, 2) \rightarrow (2, 2) \rightarrow (1, 2)$. 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 $(i, j) \rightarrow (k, j)$ 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.

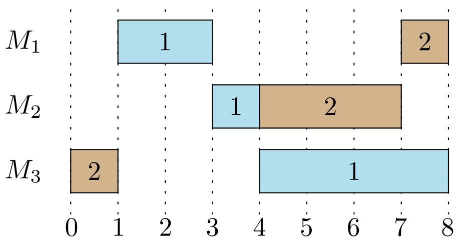


Figure B.1: Example of an optimal schedule for Example B.1, shown as a Gantt chart. Each row M_i corresponds to machine i and each block numbered j on this row represents the operation (i, j). The dashed lines indicate unit time steps. Note that machine 2 is kept idle, while operation $(2, 2)$ could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

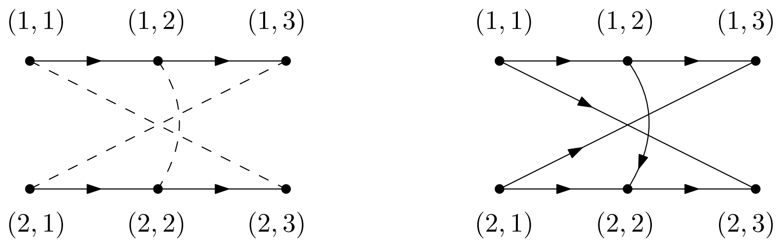


Figure B.2: Illustration of disjunctive graphs for Example B.1. Horizontal arrows represent conjunctive arcs. We used dashed lines to for the pairs of disjunctive arcs as dashed lines. The left graph corresponds to an empty selection $\mathcal{O} = \varnothing$ while the right graph shows the selection $\mathcal{O}$ 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 [52]. 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

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

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

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 C.1.

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 C.1: 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)	

Appendix D

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 [5].

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 [47]. 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 [44] 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 [66]. 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 [72]. 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 [11] employs two differentiable processes, completion and correction, to solve any violations of equality or inequality constraints, respectively. The more recent HardNet framework [49] uses a closed-form projection to map to feasible solutions under affine constraints and relies on a differentiable convex optimization solver (e.g., OptNet [2]) 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 [60] 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 [67], 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 [63] 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 [52], 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 [79] 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 [78] 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 [68], 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 E

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 [61] 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 $S_0, S_1, S_2, \dots$ 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

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

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

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

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 E.1. Furthermore, we will write $S_t^{(i)}$ to denote the state at step t of episode i.

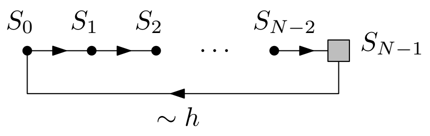


Figure E.1: Illustration of the induced Markov chain when dealing with finite episodes. The next state after the final state, indicated as the grey rectangle, is sampled according to initial state distribution h.

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

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

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

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

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 $\tau^{(1)}, \dots, \tau^{(M)}$ following $\pi_\theta$,
2. compute gradient estimate $\widehat{\nabla v_{\pi_\theta}}(\tau^{(1)}, \dots, \tau^{(M)})$,
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 [61], 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 [61], the gradient of the value function for a fixed initial state s_0 with respect to the parameters is given by

$$\nabla v_\pi(s_0) = \sum_s \sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \sum_a q_\pi(s,a) \nabla \pi(a|s). \tag{E.1}$$

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 [61]:

$$\begin{aligned}
\nabla v_\pi :=& \nabla \mathbb{E}_{s_0 \sim h}[v_\pi(s_0)] && \text{(E.2a)}\\
=& \sum_{s_0} h(s_0) \sum_s \sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \sum_a q_\pi(s,a) \nabla \pi(a|s) && \text{(E.2b)}\\
=& \sum_s \eta(s) \sum_a q_\pi(s,a) \nabla \pi(a|s) && \text{(E.2c)}\\
=& \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s) && \text{(E.2d)}\\
\propto& \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s), && \text{(E.2e)}
\end{aligned}$$

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

$$\nabla v_\pi \propto \mathbb{E}_{h,\pi}\left[\sum_a q_\pi(S_t,a)\nabla\pi(a|S_t)\right] \tag{E.3a}$$

$$= \mathbb{E}_{h,\pi}\left[\sum_a \pi(a|S_t) q_\pi(S_t,a)\frac{\nabla\pi(a|S_t)}{\pi(a|S_t)}\right] \tag{E.3b}$$

$$= \mathbb{E}_{h,\pi}\left[q_\pi(S_t,A_t)\frac{\nabla\pi(A_t|S_t)}{\pi(A_t|S_t)}\right] \tag{E.3c}$$

$$= \mathbb{E}_{h,\pi}\left[G_t \nabla \log\pi(A_t|S_t)\right]. \tag{E.3d}$$

Baseline. Let $b(s)$ be some function of the state s only, then we have for any $s \in \mathcal{S}$

$$\sum_a b(s)\nabla\pi(a|s) = b(s)\nabla\sum_a \pi(a|s) = b(s)\nabla 1 = 0. \tag{E.4}$$

This yields the so-called REINFORCE estimate with *baseline*

$$\nabla v_\pi \propto \sum_s \mu(s)\sum_a (q_\pi(s,a)+b(s))\nabla\pi(a|s) \tag{E.5a}$$

$$= \mathbb{E}_{h,\pi}\left[\big(q_\pi(S_t,A_t)+b(S_t)\big)\nabla\log\pi(A_t|S_t)\right] \tag{E.5b}$$

$$= \mathbb{E}_{h,\pi}\left[\big(G_t+b(S_t)\big)\nabla\log\pi(A_t|S_t)\right]. \tag{E.5c}$$

Although estimates (E.3d) and (E.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

$$b(s) = \mathbb{E}_{h,\pi}\left[\sum_{k=1}^{t} R_k \middle| S_t = s\right], \tag{E.6}$$

then observe that

$$q_\pi(s,a)+b(s) = \mathbb{E}_{h,\pi}\left[\sum_{k=t+1}^{\infty} R_k \middle| S_t = s, A_t = a\right] + \mathbb{E}_{h,\pi}\left[\sum_{k=1}^{t} R_k \middle| S_t = s\right] \tag{E.7a}$$

$$= \mathbb{E}_{h,\pi}\left[\sum_{k=1}^{\infty} R_k \middle| S_t = s, A_t = a\right] \tag{E.7b}$$

$$= \mathbb{E}_{h,\pi}[G_0|S_t = s, A_t = a], \tag{E.7c}$$

which is just the expected total episodic reward. Now define function f to be

$$f(s,a) := (q_\pi(s,a)+b(s))\nabla\log\pi(a|s) = \mathbb{E}_{h,\pi}\left[G_0|S_t=s,A_t=a\right]\nabla\log\pi(a|s) \tag{E.8a}$$

$$= \mathbb{E}_{h,\pi}\left[G_0\nabla\log\pi(a|s)|S_t=s,A_t=a\right], \tag{E.8b}$$

then applying the law of total expectation yields

$$\nabla v_\pi \propto \mathbb{E}_{h,\pi}[f(S_t,A_t)] = \mathbb{E}_{h,\pi}\left[G_0\nabla\log\pi(A_t|S_t)\right]. \tag{E.9}$$

Appendix F

Miscellaneous

Proposition 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 F.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 F.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. □

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 6.2.3, 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

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

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 (6.13), we define

$$x^+[p, v, \xi](t) := p + \int_{\xi}^{t} \{v + \bar{\omega}(\tau - \xi)\}_{[0,1]} \, \mathrm{d}\tau, \tag{F.2}$$

such that $x^+[\xi](t) = x^+[x(\xi), \dot{x}(\xi), \xi](t)$ and similar to (6.14), we calculate

$$x^+[p, v, \xi](t) = p + \begin{cases} \dots & \text{for } t \leq \bar{\delta}_0, \\ \dots & \text{for } t \in [\bar{\delta}_0, \bar{\delta}_1], \\ \dots & \text{for } t \geq \bar{\delta}_1, \end{cases} \tag{F.3}$$

with $\bar{\delta}_0 :=$ and $\bar{\delta}_1 :=$.

Recall the definition of $\hat{x}$ in equation (6.8). By carefully handling the $\max\{\cdot\}$, we can expand this expression as

$$\hat{x}(t) = \begin{cases} B - b + t + \bar{\omega}(b-t)^2/2 & \text{for } t \geq b - 1/\bar{\omega}, \\ B - 1/(2\bar{\omega}) & \text{for } t \leq b - 1/\bar{\omega}. \end{cases} \tag{F.4}$$

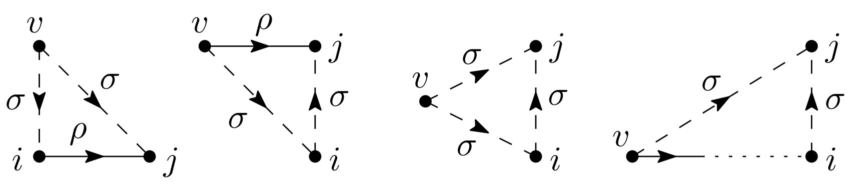


Figure F.1: Sketch of the four cases distinguished in the proof of Lemma F.3. Arc weights are indicated by σ and ρ. Conjunctive arcs $\mathcal{C}$ are drawn with solid lines and disjunctive arcs $\mathcal{O}$ are drawn with dashed lines. The dotted line depicts a chain of conjunctive arcs.

Lemma F.3. *Let $\mathcal{O}$ be some complete and acyclic selection, with unique vehicle order ν, see Remark 3.2. Suppose that $\sigma > \rho > 0$, then active schedule $y(\nu)$ is uniquely defined by starting with $y_{\nu_1} \leftarrow a_{\nu_1}$ and further following the recursion*

$$y_j \leftarrow \max\{a_j, y_i + w(i,j)\}, \tag{F.5}$$

for every pair $i = \nu_t$ and $j = \nu_{t+1}$ with $t = 1, \dots, N-1$.

Proof. We prove that y obtained in this way is a feasible solution to (AS) by showing that it satisfies (3.8). First of all, since ν_1 is a minimal node of $G(\mathcal{O})$, we have $\mathcal{N}^-(\nu_1) = \varnothing$, so feasibility condition (3.8) is trivially satisfied with equality for $y_{\nu_1} = a_{\nu_1}$. We proceed inductively, by showing that, for any further $i = \nu_t$ and $j = \nu_{t+1}$, we have

$$y_i + w(i,j) = \max_{v \in \mathcal{N}^-(j)} y_v + w(v,j), \tag{F.6}$$

so that the definition of y_j through (F.5) also satisfies (3.8) with equality. Hence, uniqueness of y simply follows from the fact that y_j satisfies (3.8) with equality for every $j \in \mathcal{N}$.

For the inductive step, we simply verify that $y_i + w(i,j) \geq y_v + w(v,j)$ for any $v \in \mathcal{N}^-(j) \setminus \{i\}$, in each of the cases illustrated in Figure F.1. Suppose i and j are connected by $(i,j) \in \mathcal{C}$, 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

$$y_i + w(i,j) = y_i + \rho \geq y_v + \sigma + \rho > y_v + \sigma = y_v + w(v,j).$$

Otherwise, i and j are connected by some 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

$$y_i + w(i,j) = y_i + \sigma \geq y_v + \sigma + \rho > y_v + w(v,j). \qquad \square$$