[everything in blue is a comment or “pending change”]

Machine Learning-Based Scheduling for Coordination of Automated Vehicles

Jeroen van Riel

2025-08-29

Abstract

Coordination of automated vehicles in a network of intersections is modeled as a trajectory optimization problem. Under certain model assumptions, this problem can be decomposed into (i) an upper-level scheduling problem of determining the crossing times at intersections and (ii) a set of lower-level trajectory optimization problems. We show that the feasibility of the lower-level problems is characterized as a set of linear inequalities in the crossing times. Since schedules can be interpreted as a sequence of discrete decisison, we experiment with sequence modeling to solve the crossing time scheduling. We compare a simple parameterization with a neural network model. As previously observed by others, our case study illustrates that the evaluation order of the sequence model matters for the final performance.

Contents

Chapter 1

Introduction and background

1.1 Introduction

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

Coordination of automated vehicles with communication has been studied at various levels of organization [1]. A good example of a local coordination methods is platooning of vehicles, where the aim is to lower energy consumption by reducing aerodynamic resistance. It has been shown that platooning can also result in a more efficient use of intersections. On a larger scale, methods like dynamic route optimization have been proposed to reduce travel delay for all vehicles in the network. The coordination problem has very many aspects that could be modeled and analyzed. For example, one may think of heterogenous vehicles—in terms of dynamics or priority—different models of centralized/decentralized communication between vehicles or with the infrastructure, under different guarantees on reliability; complex road topology, curved lanes, merging lanes.

However, as we will see, even the most basic models already present fundamental challenges in ensuring safety and efficiency. Therefore, we will only consider the two most essential elements in this thesis, being vehicle dynamics and the constraints that are required to model the allowed paths vehicles may take in order to avoid collisions with the infrastructure and other vehicles. To keep things simple, we assume that all vehicles are automated and share the same dynamics. Each vehicle follows a fixed route through the network and is centrally controlled through acceleration inputs under the assumption of perfect communication. With these assumptions, the traffic coordination task can be modeled a trajectory optimization problem.

Discrete decisions in traffic coordination. We need to take a discrete decision regarding the crossing order of vehicles at intersections. This remains true for any generalization of the model to 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.

Without additional assumptions, the upper-level problem is still as difficult as before, because the feasibility of a crossing time schedule may depend on the feasibility of the lower-level trajectory optimization problems in a non-trivial way. We will provide assumptions

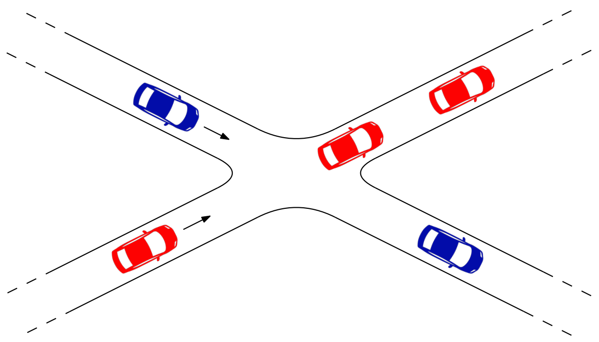


Figure 1.1: Stylized model of a single intersection with two conflicting flows that do not merge, so we assume vehicles must cross the intersection in a straight line, without turning.

under which this coupling becomes particularly simple. Specifically, we show that feasibility of the lower-level problems can be states as a system of linear inequalities in terms of the crossing times and we will see that they also have simple interpretations. This allows us to formulate the upper-level problem as a mixed-integer linear problem that looks very similar to the classical job shop scheduling problem.

Learning to schedule. Because the lower-level problems can in general be solved efficiently, the original trajectory optimization problem is essentially reduced to a scheduling problem. This enables us to explore some applictions of recent machine learning techniques for such problems. Specifically, the solution of a 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.

1.1.1 Contributions and outline

This thesis is centered around the following two main contributions:

(i). Decomposition. 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 to schedule. Our second contribution is an illustration of how machine learning techniques can be applied to solve scheduling problems in practice. Many practical instances contain structure that classic solutions techniques try to exploit, e.g., by defining smart heuristics based on human intuition and experience. We aim to automated this manual endeavor by formulating parameteric sequence models to capture the conditional probability of optimal solutions, given a problem instance. As has been noted before, we confirm that the order of evaluation during inference matters a lot for the final solution quality.

Outline. [probably going to change] The rest of this chapter discusses some preliminaries: we briefly discuss the job shop scheduling problem in, because our crossing time scheduling problem may be seen as an extension; we provide a brief overview of how machine learning methods can be applied to solve combinatorial optimization problems, with a focus on job shop scheduling. In Chapter 2, we consider a simple model of a single intersection, like the example above. After discussing the decomposition method, we present some classical solutions methods to solve the crossing time scheduling problem. We explain how the problem can be treated as a learning problem in Chapter 3. In order to generalize to a network of intersections, we need to precisely study the feasibility of trajectories in lanes of finite length, which is done in Chapter 4. The resulting scheduling problem is then subjected to a learning algorithm in Chapter 5. We provide some general discussion and pointers for further research in Chapter 6.

Ref.	Paradigm	Dynamics	Safety	Objective	Decision set	Communication	Notes / Limitations
[2]	Reservation-based	Planar vehicle kinematics	Conflict exclusion (grid cells)	Delay minimization	Entry times/reservation slots	Multiagent with central intersection manager	Simplified vehicle model; global heuristic

Table 1.1: Comparison of approaches to signal-free intersection management.

1.2 Literature review

We briefly survey some releated work that addressed coordination of autonomous vehicles in a similar setting. A good example of an early centralized approach is the "Autonomous Intersection Management" (AIM) paper [2], which is based on a reservation scheme. The conflict zone is modeled as a grid of cells. Vehicles that want to cross the intersection send a request to the central controller to occupy the cells containing its trajectory for a certain amount of time. The central controller then decides to grant or deny these requests based on previous granted requests, in order to facilitate collision-free trajectories. If a request is denied, the vehicle slows down and attempts to obtain a new reservation after some timeout.

Direct transcription. Optimal control problems can be approached in an end-to-end fashion by *direct transcription* to an equivalent mixed-integer optimization problem, which can be solved using off-the-shelf solvers (e.g., SCIP [3] or Gurobi [4]). Such methods can be used to compute optimal trajectories up to any precision, by choosing a fine enough time discretization. However, it is exactly this time discretization that causes prohibitive growth of the number of variables with respect to the size of the network and the number of vehicles, so this method is only useful for relatively small problem instances. Therefore, approximation schemes have been studied in previous works [5, 6, 7], which we will review next.

Decomposition methods. The approximation method in [5] 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 [6] 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. The optimization scheme in [7] deals explicitly with the complexity of the crossing order decisions by defining groups of consecutive vehicles on the same lane. The first step is to group vehicles into these so-called "bubbles". All vehicles in a bubble are required to cross the intersection together, while maintaining feasibility with respect to safe trajectories. Next, crossing times are assigned to bubbles while avoiding collisions. Based on this schedule, a local vehicular control method [8] is used that guarantees safety to reach the assigned crossing times.

1.2.1 Structure

Project scope.

- fully automated systems

  - We will use the term (connected) Automated Vehicles, shorthand AVs, because we ”take communication for granted”
  - already applicable in ”confined sites” like
    * ports, mines, quarries
    * warehouses
    * manufacturing sites (e.g. Prodrive)
    * indoor vertical farming sites (e.g. Helmond)

- focus on the optimization-based coordination of vehicle motion (joint=”decisions+control”)
  - lower-level steering control of vehicles is ignored: it is assumed that vehicles can follow planned trajectories perfectly

- why such stylized models? /(chatGPT suggestion: this also touches on challenges)/
  - better understand inherent difficulties
  - obtain results that are better interpreted:
    ”Simple models are relevant, because they provide upper bounds on what we can expect to gain in more realistic situations with decentralized control, communcation latency and noisy information.”

Goals. Understand how to *integrate* model-free (RL) with model-based (opt. control, scheduling) algorithms.
(optimization setting allows us to focus on goals that can be defined mathematically)

- conflicting optimization goals:
  - min. delay (max. throughput), like in studied in queueing theory
  - min. energy consumption / reduce emission
  - fairness of usage (think reservation / bidding systems)
  - guarantee safety (need to define what this means)

Challenges: [REVIEW] *(ChatGPT says: ”It’s long and risks overwhelming the reader before they even get to the fun methods.”)*
Emphasize the combinatorial problems that occur naturally.

- broad modeling landscape
  - mention some aspects (preview):
    * network topology
    * vehicle dynamics: degree of realism vs. computational tractability
    * random events, e.g., arrivals, failure models
    * topic on its own: communication systems (protocols, delay, robustness to failure)
  - no standard benchmarks yet like Taillard (they said this in the poorly written survey, but is this really so?)

- inherent algorithmic challenges (even in stylized deterministic models)
  - mix of discrete decisions and continuous-time control of vehicles (almost always non-convexity involved)

  - even when routes are fixed, there is still a lot of freedom in determining the speed profile over the course of routes. (infinite-dimensional optimization = optimal control)

- random network effects
  - inherent nonstationarity of mobility demand, with fluctuations throughout the day, makes requires adaptive control.
  - decisions at one intersection directly influence the arrival process at nearby intersections
  - in dense urban networks, these interactions are highly complex: a lot of effort has gone into developing methods for predicting traffic

- practical constraints /(for transfer to applications)/
  - real-time computation requirements (fast) (similar setting in which MPC is often employed)
  - real-time communication requirements (robust)

Main methodologies (overview). *(...and introduce our contributions at this point?)*

- bilevel decomposition -> scheduling -> MILP
  - observation: the combinatorial part of offline trajectory coordination shows similarities to job-shop scheduling
  - motivation: bilevel decomposition can provide a conceptual tool for further development of algorithmic ideas
  - goal: (systematic/interpretable) dimensionality reduction of the problem
    - for example, optimal control problem: discretization, splines
    - integer programming: benders' decomposition
    - why "systematic/interpretable": because e.g. model-free RL is not interpretable, so does provide less insight for further algorithm design
  - question: when is such a dimensionality reduction sound?

    i.e., when can the combinatorial part be completely separated from the continuous-time control problem
  - subquestion: given this decomposition, can the feasibility of lower-level be easily characterized or approximated?

- "ML for algo learning" -> sequential solution encoding/synthesis -> sequence modeling -> RL for param. fitting
  - observation: new perspective, recent advances
  - "automated heuristic search" inspiration: exploiting specific structure in the problem
    - key concept: algorithm as MDP
  - motivation: recent successes in neural combinatorial optimization (e.g. TSP)
  - question: how to apply this methodology in traffic coordination setting
    - subquestion: How do parametric sequence models perform on this task?
    - subquestion: How to parameterize the model?

Scope of literature review. ”We do a general literature review on *optimization-based methods for traffic coordination in fully automated mobility systems*. Brief introduction of *methodologies* in the chapter where we first need it. We can also put an overview of literature there, or in the appendix.”

Adjacent topics. *(so don’t spend too much time here) (present in order: least relevant/history to slightly adjacent)*

- dynamic path/route planning
- focus on decentralized nature -> communication becomes inherent issue
  - reservation systems
  - auction/bidding systems
  - fairness: average vehicle delay vs. all vehicles experience similar delay (e.g., in proportion to the length of their trip)
- traffic signal optimization (/”brief for historical context”/)
  - static (Fleuren)
  - adaptive (baroque RL studies)
- platooning in single lane, CACC
- platooning and CACC more generally: trajectory optimization, given fixed signal phases

special remark: why not model-free RL?

- model-free RL (”brute-force” control synthesis)
  - recognize that this is a powerful framework
- in essence, we advocate to use model-free RL, but at a more local level (to, hopefully, exploit structure)

topics / distinguishing features:

- modeling aspects
  - modeling of conflict zones ”mutex zones” (see Kojchev)
  - single int. vs network
  - single lane vs multilane
  - vehicle dynamics model
  - use of simulator like SUMO and VISSIM
  - central vs decentralized control
  - mixed traffic (legacy vehicles, human drivers)
- /specific focus on model/challenges/algorithmic approach/
  - focus on continuous control:
    - trajectory optimization, given fixed signal phases
      - different objectives:
      - minimizing fuel consumption

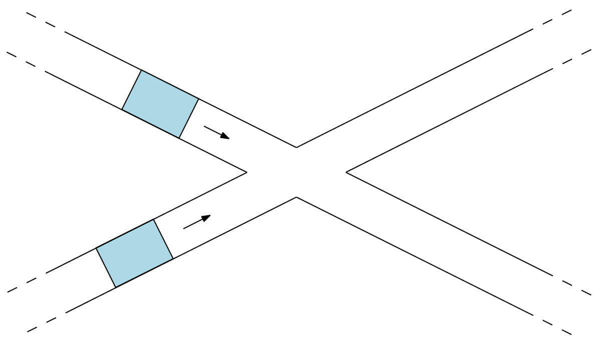
        - maximizing passenger comfort
    - platooning
        - is widely used as the basis for heuristic approaches
        - arises as a result of the assymetry of the combinatorial problem, namely the difference in processing time and switch-over time
        - cooperative adaptive cruise control (CACC), string stability
    - important in all these: real-time control, which causes the following common issues
        - how to actually implement this in electronic/software controller?
        - discretization
        - stability
        - decentralization causes problems: communication / synchronization (string stab.)
- focus on stochasticity: random arrivals and failure
    - -> offline vs online optimization
    - ”rolling horizon” schemes
        - Model Predictive Control (MPC)
    - robustness to failures, safety guarantees even without explicit ”fatal crash”-type failures happening (e.g., Miculescu Karaman: Theorem IV.5)
- focus on combinatorics:
    - of course: service order at intersections
    - ”MUTEX” zones, in general, see Kojchev
    - even: in which order to organize vehicles in a platoon approaching intersection (efficiently ”voorsorteren”)
    - larger scale: path planning in network
    - common methods:
        - dealing with binary choices: Mixed-Integer (Linear) Programming
        - heuristics and local search
        - classical search algorithms like A*

main topics/features for literature comparison table:

- model/challenge focus: ”combinatorics”, ”cont-time control”, ”decentralized control”
- main algorithmic paradigm: e.g. ”reservation”, ”decomposition”, ”MILP”, ”model-free RL”

Part I

Single Isolated Intersection



Chapter 2

Isolated intersection scheduling

Efficient coordination of vehicle motion at intersections is one of the central challenges in traffic management, since intersections are natural bottlenecks where safety requirements and efficiency objectives are directly in conflict. With the advent of automated vehicles and reliable wireless communication technologies, there is increasing potential to replace traditional traffic signal-based approaches with coordinated trajectory planning methods. As a first step toward analyzing such methods, this chapter focuses on a simplified single-intersection setting in which vehicles follow fixed routes and are centrally controlled to optimize traffic flow while guaranteeing safety.

To make the analysis tractable, we restrict attention to an intersection formed by two perpendicular lanes, with vehicles traveling straight through, without allowing turning maneuvers or overtaking. All vehicles are assumed to be homogeneous, sharing identical dimensions and dynamics, so that their motion can be modeled uniformly. A central controller determines the acceleration, and thus the speed, of each vehicle, under the assumption of perfect communication. Moreover, we do not consider randomness in arrivals or dynamics, so we assume that each vehicle's initial state is known precisely such that the system evolves deterministically as a function of the acceleration control inputs.

Within this setting, two performance objectives are of primary interest. Minimizing energy consumption is highly desirable in practice, as it directly contributes to the sustainability benefits of automated traffic systems. However, incorporating energy makes analysis significantly more complex. By contrast, minimizing travel delay is essential for capturing the efficiency of traffic flow and, importantly, we will see that it enables a very useful problem decomposition. For these reasons, the focus of this chapter is primarily on minimizing delay, while recognizing energy consumption as a complementary objective.

Formulated mathematically, the coordination problem at the intersection can be expressed as a problem of *trajectory optimization*, or optimal control. In this formulation, the accelerations of all vehicles are chosen to satisfy their dynamic constraints and collision-avoidance requirements while minimizing the chosen performance objective. Such problems can in principle be solved directly using standard numerical optimization techniques. However, by exploiting the structure of the intersection model, we show that the problem can be reduced to a much simpler form: a *scheduling problem*, in which only the sequencing of vehicles and their entry times into the intersection need to be determined.

The results presented in this chapter illustrate both perspectives. After precisely formulating our model, we first apply a direct transcription approach to the trajectory optimization problem, demonstrating how it can be solved numerically in its original form. The main contribution, however, is stating assumptions for which the trajectory problem allows a bilevel decomposition in which the scheduling problem provides high-level sequencing and timing

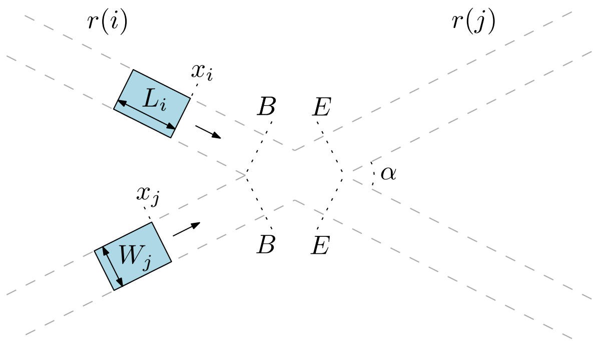


Figure 2.1: Example of two vehicle routes intersecting at some angle α. The indicated positions $x_i = B$ and $x_i = E$ for each vehicle i are chosen such that whenever the vehicle position x_i satisfies either $x_i \leq B$ or $x_i - L_i \geq E$, then the intersection area is not occupied by this vehicle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

decisions, while a continuous trajectory planning routine resolves the underlying vehicle dynamics. To solve the scheduling component, we apply the branch-and-cut framework and leverage the problem structure to generate effective cutting planes, leading to significant performance improvements. Finally, we briefly suggest a local search heuristic.

Remark 2.1. *We will consider a continuous-time trajectory optimization problem, which is typically studied in the framework of optimal control theory. However, as the author does not have a strong background in this field, the aim is not to provide a rigorous treatment from this perspective. Nevertheless, at some points remarks are included regarding the difficulties of such analysis, most notably, regarding the occurence of so-called state constraints. 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 using some sort of penalty-based encoding.*

2.1 Intersection model

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. Therefore, we will write $\mathcal{B}_i(x_i)$ to denote the corresponding translated rigid body in the plane, where x_i corresponds to the location of the front bumper; the rear bumper position is then $x_i - L_i$. We allow multiple routes to cross in a single point. Of course, this causes some joint vehicle positions to be invalid, because they would correspond to collisions. Before we allow arbitrary numbers of vehicles to have the same route, we briefly investigate the valid configurations of two vehicles, each on its own route.

Valid configurations. Consider two routes intersecting at some angle α, as illustrated in Figure 2.1, each with a single vehicle on it. Let these vehicles be denoted as i and j. We can try to characterize the set $\mathcal{X}_{ij} \subset \mathbb{R}^2$ of feasible configurations (x_i, x_j) for which these two vehicles are not in a collision, in the sense that their corresponding translated rigid bodies do not intersect. In general, this set can thus simply be defined as

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

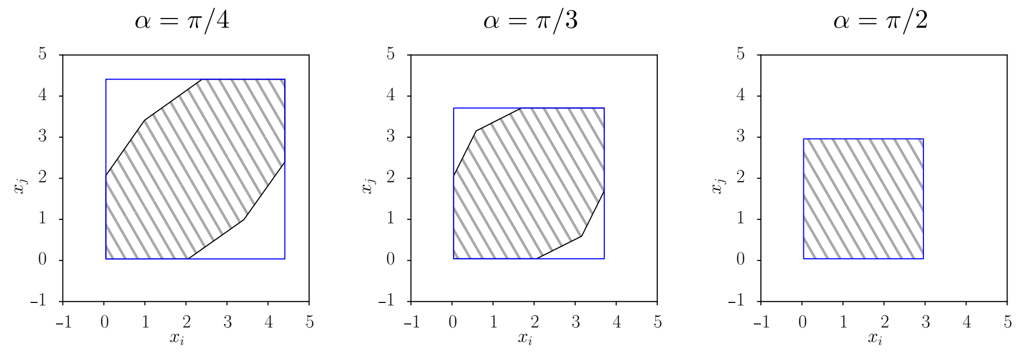


Figure 2.2: For three different intersection angles and fixed vehicle dimenions $W_i = W_j = 1$ and $L_i = L_j = 2$, we plotted the region $\mathcal{X}_{ij}^C$ in configuration space corresponding to collisions as the area marked in grey. The blue square regions correspond to the approximation of the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

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

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

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

Of course, the set of feasible configurations is generally a little larger and depends on the angle α of intersection, as illustrated by the three examples in Figure 2.2. In case of the third example, where the intersections make a right angle $\alpha = \pi/2$, it can be shown that there is actually equality in (2.2).

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

Let us now proceed to arbitrary numbers of vehicles. We will use the following notation for identifying routes and vehicles. Let the routes be identified by indices $\mathcal{R}$. Let n_r denote the number of vehicles following route r, then the set of vehicle indices is defined as

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

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

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

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

at all times, for all pairs of indices $i, j \in \mathcal{N}$ such that $r(i) = r(j)$ and $k(i) + 1 = k(j)$. Let $\mathcal{C}$ denote the set of all such ordered pairs of indices. Note that these constraints restrict vehicles from overtaking each other, so their initial relative order is always maintained. In other words, $(i, j) \in \mathcal{C}$ means that i crosses the intersection before j.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned} \min_x \quad & \int_{t_0}^{t_f} \mathcal{L}(t, x(t), u(t))\,\mathrm{d}t + \mathcal{K}(t_f, x_f), && \text{(2.10a)} \\ \text{such that} \quad & \text{(2.9) holds,} && \text{(2.10b)} \\ & h(x) = 0, && \text{(2.10c)} \\ & g(x) \leq 0, && \text{(2.10d)} \end{aligned}$$

with some functions $h : \mathbb{R}^n \to \mathbb{R}^p$ and $g : \mathbb{R}^n \to \mathbb{R}^q$ encoding the so-called state constraints. Functions f, g, h are usually assumed to posses certain regularity or smoothness properties. There are general methods to analyze optimal solutions of (2.10)*, most notably the necessary conditions given by the maximum principle of Pontryagin [9]. However, the occurence of state constraints like* (2.10c) *and* (2.10d) *make such analysis notoriously difficult and remains an area of ongoing research [10]. For our current problem, note that the collision-avoidance constraints* (T.2) *and* (T.3)*, as well as the speed constraint* (2.6a) *implied in* (T.1)*, are all of the type* (2.10d).

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

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

Next, observe that the bounds on speed and acceleration imply that it takes at least $1/\omega$ time to fully decelerate from full speed, during which the vehicle has traveled a distance of $1/(2\omega)$. For a full acceleration from a stop, we obtain the same expressions, but with ω replaced by $\bar{\omega}$. Therefore, by assuming that each vehicle starts at full speed $v_i^0 = 1$ and by imposing a minimum distance to the start of the intersection

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

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

2.2 From joint optimization to bilevel decomposition

To tackle the trajectory optimization problem (T), we first employ a numerical method known as *direct transcription.* The key idea is to reformulate the continuous-time optimal

[1]Note that we will precisely characterize the feasibility of trajectories for a related problem in Chapter 4. The main difference is that the analysis there will not assume that initial vehicle states are given, but rather the times at which vehicles enter the system.

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

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

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

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

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

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

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

Encoding of the safe crossing constraints needs some additional attention, because they represent a binary "disjunctive" decision. Following the approach in [5], these disjunctive constraints can be formulated using the common big-M method with binary decision variables. For each vehicle $i \in \mathcal{N}$, we introduce two binary decision variables $\delta_i(t), \gamma_i(t) \in \{0, 1\}$ and for each conflict $\{i, j\} \in \mathcal{D}$ and time step $t \in \mathbb{T}$, we introduce the following constraints:

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

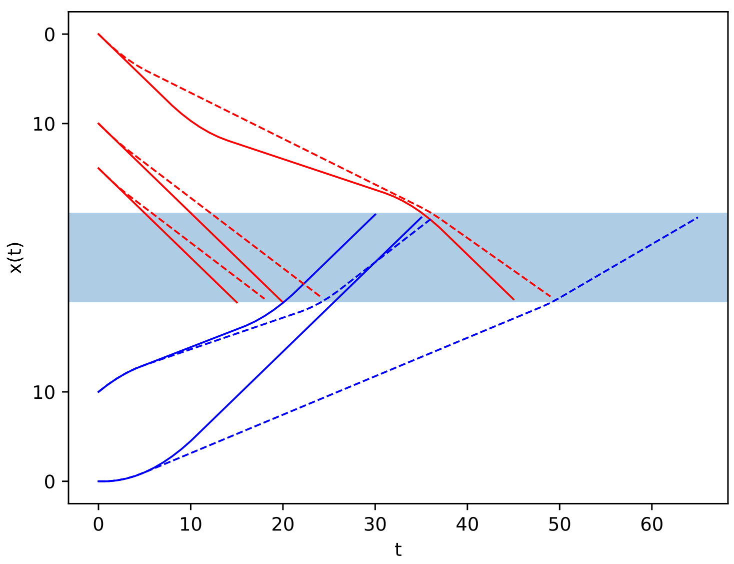
where M is some sufficiently large number. The idea behind this encoding is as follows. First, observe that setting $\delta_i(t)$ can be thought of as *deactivating* (2.16a), since M is chosen sufficiently large such that the inequality is trivially true. Analogously, setting $\gamma_i(t) = 1$ deactivates (2.16b). Hence, the constraint (2.16c) can be interpreted as limiting the number of deactivations to three, which is equivalent to requiring at least one of the following four inequalities to hold:

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

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

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

Remark 2.4. *We used the forward Euler integration scheme for sake of a simple presentation. However, note that in practice we most likely want to use a more numerically stable method like a higher-order Runge-Kutta scheme or by means of some spline interpolation technique (collocation). We refer to [11] for a light introduction of trajectory optimization and [12] for a tutorial on the direct collocation method, which also contains a concise overview of different numerical methods in (ibid., Section 9).*



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

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

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

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

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

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

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

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

are completely equivalent to the constraints

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

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

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

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

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

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

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

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

Example 2.1. Suppose we have two routes, with a single vehicle on each. Each vehicle has initial speed $v_i^0 = \bar{v}$ and some initial distance x_i^0 satisfying the assumption (2.11), such that it can perform a full deceleration ($\ddot{x}_i = \omega$) and come to a stop somewhere before the start of the intersection at some point $x_i(1/\omega) < B$. Consider the cost functional $J_{\alpha,\beta}$ with $\alpha = 1$ and $\beta = 0$, then it is easily seen that the optimal solution is to have both vehicles decelerate immediately as just described. However, this is not a feasible solution for (U).

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

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

[2]The authors of [5] refer to this assumption as “strongly output monotone”.

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

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

for some convex and quadratic function $\Lambda(x, v, u)$, it has been argued that (T) and (U) are equivalent [13, Theorem 1]. The argument there relies on the fact that the lower-level has a unique solution. However, as we will illustrate shortly hereafter, there are interesting problem settings for which this does not hold. Instead of assumption (2.21), we will explicitly restrict the set feasible trajectories to cross the intersection by introducing the problem variant

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

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

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

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

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

Remark 2.5. *In the above theorem, we assumed feasibility of* (T) *and the existance of an optimal solution. In general, these properties are not trivial to establish and rely on the compactness of the reachable sets, which are, roughly speaking, all the possible configurations a system can achieve after a fixed time; see [9, Section 4.5].*

Remark 2.6. *The constraints* (L.3)*,* (L.4)*,* (T*.4) *and* (T*.5) *are state constraints of a different type than those discussed earlier in Remark 2.3. Namely, $g(x(t)) \leq 0$ is imposed for all times $t \in [t_0, t_f]$. However, the constraint $x_i(y_i) = B$ only holds at some pre-specified intermediate time $y_i \in [t_0, t_f]$ and may be interpreted as some kind of "checkpoint". This case is not typically considered in the literature, but it is possible to reduce the problem into a canonical form that only has such equality constraints at the endpoints of the time interval, i.e., $x(t_0) = x_0$ and $x(t_f) = x_f$, see [14]. Whenever $t_f < \infty$, the constraints $\tau(x_i) < \infty$ and $\xi(x_i) < \infty$ together can be replaced by the equivalent endpoint constraint $x_i(t_f) \geq E + L$.*

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

For a single vehicle per route, so $(y_r, z_r) \equiv (y_i, z_i)$, the approach taken in [5] is to approximate both these objects as follows: they fit a quadratic function for $F(y_i, z_i, s_i)$ and they approximate the set of feasible occupancy slots by considering the polyhedral subset

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

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

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

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

which means that the only remaining difficulty is how to deal with feasibility of the lower-level problem. To make this simpler, we additionally assumptions that all vehicles start at full speed $v_i^0 = \bar{v}$ and that vehicles must enter the intersection at full speed, i.e., we add the constraint $\dot{x}_i(\tau(x_i)) = \bar{v}$ to (T*) and $\dot{x}_i(y_i) = \bar{v}$ to (L). [do we need this?] Let the resulting problems variants be denoted (T**) and (L**), respectively

Under these assumptions, the set of feasible parameters $\mathcal{T}(s_r)$ is now easier to characterize. Given some (y, z), every x_i in a a solution x_r for (L**) satisfies $x_i(y_i) = \bar{v}$. Therefore, we can let each vehicle continue at full speed across the intersection, without inducing extra cost. This takes exactly $\sigma := (B - E)/\bar{v}$ time, so we can always set

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

With this choice, observe that $\rho := L/\bar{v}$ is such that $x_i(y_i + \rho) = x(y_i) + L = B + L$, which is the time after which the next vehicle from the same route can enter the intersection, so we will refer to ρ as the *processing time*. Since z_i is not involved in $J(\cdot)$, this means we only have to care about finding

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

where $y_r + \sigma$ is to be understood as $\{y_i + \sigma : i \in \mathcal{N}_r\}$. It can be shown that $y_r \in \mathcal{T}_y(s_r)$ holds if and only if

$$\begin{aligned} a_i &\leq y_i && \text{for all } i \in \mathcal{N}_r, \\ y_i + \rho &\leq y_j && \text{for all } (i,j) \in \mathcal{C} \cap \mathcal{N}_r, \end{aligned}$$

where $a_i := (B - x_i^0)/\bar{v}$ is the earliest time at which vehicle i can enter the intersection. A rigorous proof of this fact is outside the scope of the current chapter, but we note that the analysis is very similar to that of Chapter 4.

To conclude this section, the stated assumptions mean that problem (U**) reduces to the following *crossing time scheduling* problem

$$\min_{y} \quad \sum_{i \in \mathcal{N}} y_i \tag{C}$$

$$\text{s.t.} \quad a_i \leq y_i \qquad \text{for all } i \in \mathcal{N}, \tag{C.1}$$

$$y_i + \rho \leq y_j \qquad \text{for all } (i,j) \in \mathcal{C}, \tag{C.2}$$

$$[y_i, y_i + \sigma] \cap [y_j, y_j + \sigma] = \varnothing \qquad \text{for all } \{i,j\} \in \mathcal{D}. \tag{C.3}$$

This is a typical scheduling problem, which can for example be solved within the mixed-integer linear programming framework after encoding the *disjunctive constraints* (C.3) using the big-M technique. This methodology will be applied in the next section.

2.3 Crossing time scheduling

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

This problem is inherently a combinatorial optimization problem, for which a wealth of general solution techniques is available [15], including algorithms specifically tailored to problems that can be stated in terms of finding optimal time schedules [16, 17]. One of the most prominent distinction among such algorithms lies in whether it guarantees to find an optimal solution or not. Of course, it is desirable to find optimal solutions, but this is often not tractable in practice, because the number of feasible solutions typically explodes whenever larger instances are considered. In other words: enumerating all solutions and evaluating their objective value to find the best one is typically not achievable in any reasonable amount of time. This is the main motivation for the development of so-called heurstics, which discard the optimality guarantee in favor of speed.

We will illustrate both types of methods in the rest of this section. First, we show how to leverage the general branch-and-cut framework by formulating (C) as a mixed-integer linear program and defining three types of cutting planes. We evaluate the computational performance of this method in Section 2.3.2. As an illustration of a classical approach to heuristic design, Section 2.3.3 provides a brief description of a local search procedure. The next chapter will discuss methods to automate the search for heuristics using a machine learning methodology.

2.3.1 Branch-and-cut

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

strenghts: optimality guarantee, flexibility: fairly broad modeling capacity, maturity: take advantage of modern solvers; *weaknesses:* no guarantees on computation time, reformulation can be involved;

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

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

given some matrix $A \in \mathbb{R}^{m \times n}$ and vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. This surprisingly simple setup provides a powerful modeling toolkit to approach many types of scheduling 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. Moreover, reformulating a problem as (MILP) opens up the possibility to leverage the collective effort that has gone into developing fast general solvers: many modern solvers, e.g., Gurobi [4] (commercial) or SCIP [3] (academic), employ specialized interal heuristics and techniques like cutting planes to derive better bounds and thus achieve more pruning of the search tree to speed up the solving process.

Reformulation. To formulate the crossing time scheduling problem (C) into (MILP), we rewrite the disjunctive constraints using the big-M method by introducing a binary decision variable γ_{ij} for every conflict $\{i,j\} \in \mathcal{D}$, such that for every $(i,j) \in \bar{\mathcal{D}}$, 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 necessary to impose some arbitrary ordering of the conflicting vehicles by defining the index set

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

We now have the following reformulation of (C):

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

where $M > 0$ is some sufficiently large number. In this form, the problem can be solved by any off-the-shelf MILP solver, for which a wide variety of 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[3] package to call the solver from the comfort of Python.

Disjunctive graph. Before we evaluate whether (C') can be solved in reasonable amounts of time we present some insights into the structure of optimal solutions, which also serves as the basis for defining three types of cutting planes that can be added to (C'), which might improve the solving time.

[some lack of continuity and motivation here] Recall the intial ordering of vehicles due to the safe headway constraints and the definition of earliest arrival time a_i. Let $\mathit{pred}(i)$ denote the set of vehicles that arrive no later than i on route $r(i)$ and let $\mathit{succ}(i)$ denote those that arrive no earlier, so we have

$$\mathit{pred}(i) := \{(r(i), k) : k > k(i)\}, \tag{2.26a}$$
$$\mathit{succ}(i) := \{(r(i), k) : k < k(i)\}. \tag{2.26b}$$

Consider some conflict pair $(i,j) \in \bar{\mathcal{D}}$ and suppose $\gamma_{ij} = 0$, so i crosses before j, then any feasible solution must satisfy

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

[include graphical illustration, motivating our later introduction of the disjunctive graph] Using the big-M method, we can equivalently encode this condition as

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

for every conflict $(i,j) \in \bar{\mathcal{D}}$, to which we refer as the *transitive cutting planes.*

Apart from the just described redundancy, which is inherent to the way conflicts are encoded, we will now study some less obvious structure in the problem. Let us first consider some basic examples to sharpen our intuition.

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

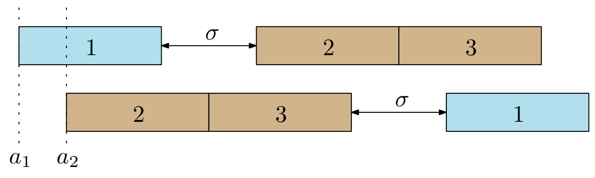


Figure 2.4: Illustration of the two possible sequences of vehicles in Example 2.3.

Example 2.2. Consider the trivial problem with two routes and a single vehicle per route. When both vehicles have the same earliest arrival time $a_i = a$, the order in which they cross the intersection does not matter for the final objective, which is σ. When either of the vehicles has a strictly earlier release date, then it is obvious that this vehicle must go first in the optimal schedule.

Example 2.3. Consider two routes having one and two vehicles, respectively. Instead of $(1, 1), (2, 1), (2, 2)$, we will use the labels $1, 2, 3$ to keep notation clear. We are interested in how the earliest crossing times influence the order of the vehicles in an optimal schedule. We set $a_1 = 0$, without loss of generality, and assume that $a_3 = a_2 + \rho$. Suppose $a_1 = a_2$, then we see that the order $2, 3, 1$ is optimal, which resembles some sort of "longest chain first" rule. Now suppose that $a_1 < a_2$. For $a_2 \geq a_1 + \rho + \sigma$, the sequence $1, 2, 3$ is simply optimal. For $a_2 < a_1 + \rho + \sigma$, we compare the sequence $1, 2, 3$ with $2, 3, 1$, which are illustrated in Figure 2.4. The first has $\sum_i y_i = (\rho + \sigma) + (\rho + \sigma + \rho) = 3\rho + 2\sigma$, while the second sequence has $\sum_i y_i = a_2 + (a_2 + \rho) + (a_2 + \rho + \rho + \sigma) = 3a_2 + 3\rho + \sigma$. Therefore, we conclude that the second sequence is optimal if and only if

$$a_2 \leq \sigma/3, \tag{2.27}$$

which roughly means that the "longest chain first" rule becomes optimal whenever the release dates are "close enough".

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

Definition 2.1. 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

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

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

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

Example 2.4. 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 $a_A \leq a_B < n_A\rho + \sigma$. Consider the ways the two platoons can merge by splitting A. Let k denote the number of vehicles of platoon A that go before platoon B and let $\sum y_i(k)$ denote the corresponding sum of crossing times. See Figure 2.5 for an illustration of the situation in case of $a_A = a_B$. For $0 < k \leq n_A$, we have

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

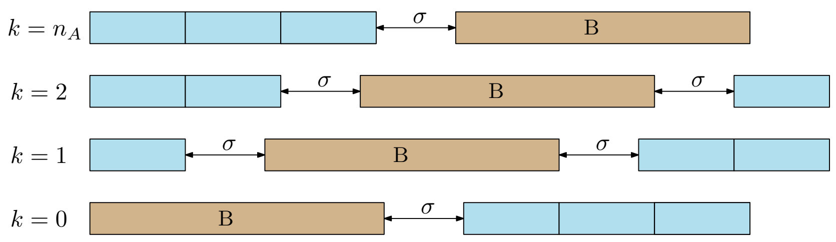


Figure 2.5: Different ways to split platoon A, regarding Example 2.4, assuming equal earliest arrival times $a_A = a_B$ with $n_A = 3$ vehicles in platoon A and some arbitrary number of vehicles n_B in platoon B.

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

$$\sum_{i\in\mathcal{N}} y_i(n_A) = \sigma n_B + \sum_{j=1}^{n_A+n_B} (j-1)\rho, \tag{2.28}$$

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

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

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

Comparing this to (2.28), we conclude that placing B in front is optimal whenever

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

which directly generalizes the condition (2.27) 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 the whole platoon in front. In other words, whenever a vehicle can be scheduled immediately after its predecessor, this should happen in any optimal schedule, as stated by the following result, for which we provide a proof in Appendix **??**.

Theorem 2.2 (Platoon Preservation [18])**.** *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 $y_{i^*} + \rho = y_{j^*}$.*

We will now use this result to define two types of additional cutting planes. In order to model this necessary condition, we introduce for every conjunctive pair $(i, j) \in \mathcal{C}$ a binary variable $\delta_{ij} \in \{0, 1\}$ that satisfies

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

which can be enforced by adding to the constraints

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

Now observe that Theorem 2.2 applied to (i, j) is modeled by the cutting plane

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

We refer to these cutting planes as *necessary conjunctive cutting planes.* Using the definition of δ_{ij}, we can add more cutting planes on the disjunctive decision variables, because whenever $\delta_{ij} = 1$, [remove disjunctive-graph-speak here] the directions of the disjunctive arcs $i \to k$ and $j \to k$ must be the same for every other vertex $k \in \mathcal{N}$. Therefore, consider the following constraints

$$\begin{aligned} \delta_{ij} + (1 - \gamma_{ik}) + \gamma_{jk} \leq 2, \\ \delta_{ij} + \gamma_{ik} + (1 - \gamma_{jk}) \leq 2, \end{aligned}$$

for every $(i, j) \in \mathcal{C}$ and for every $k \in \mathcal{N}$ with $r(k) \neq r(i) = r(j)$. We will refer to these types of cuts as the *necessary disjunctive cutting planes.*

2.3.2 Runtime benchmark

[either keep it simple here, we only want to illustrate that MILP is no silver bullet; or already setup our problem instance classes, which we will use in the results of the next chapter] [I think the first option is best, because then this dicussion is then closer to the actual tables in which these classes 'low', 'med', 'high' are actually used]

In order to make a comparison of the relative performance of the different cutting planes, we first define the distribution over problem instances from which we will take samples to use as a benchmark.

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

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

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

Remark 2.7. *Note that the arrival process may be interpreted as an renewal process with interarrivals times $X_n + \rho$. Let N_t denote the corresponding counting process, i.e., N_t counts the cumulative number of arrivals up to time t, then by the renewal theorem, we obtain the limiting density of arrivals*

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

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

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

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

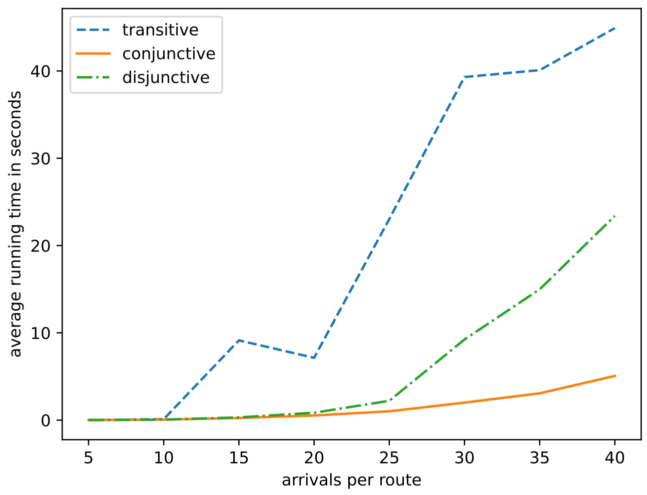


Figure 2.6: The average running time of the branch-and-cut procedure is plotted as a function of the number of arriving vehicles per route, for each of the three indicated cutting planes. Each average is computed over 20 problem instances. All instances use $\rho = 4$ and $\sigma = 1$. The arrivals of each of the two routes are generated using the bimodal exponential interarrival times with $p = 0.5, \mu_s = 0.1, \mu_l = 10$. This figure clearly shows that the conjunctive cutting planes provide the most runtime improvement. [why not show the case without any cutting planes?]

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

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

to keep the arrival rate constant accross arrival distributions.

Comparison. We now assess which type of cutting planes yields the overall best performance. The running time of branch-and-cut is mainly determined by the total number of vehicles in the instance. Therefore, we consider instances with two routes and measure the running time of branch-and-cut as a function of the number of vehicles per route. In order to keep the total computation time limited, we set a time limit of 60 seconds for solving each instance. Therefore, we should be careful when calculating the average running time, because some observations may correspond to the algorithm reaching the time limit, in which case the observation is said to be *censored*. Although there are statistical methods to rigorously deal censored data, we do not need this for our purpose of picking the best type of cutting planes. Figure 2.6 shows the average (censored) running time for the three types of cutting planes. Observe that the necessary conjunctive cutting planes seem to lower the running time the most.

2.3.3 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. We will now give an example of such a neighborhood.

Table 2.1: Neighborhood of route order $\eta = (0, 1, 1, 0, 0, 1, 1, 1, 0, 0)$.

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)	

As seen in the previous sections, vehicles of the same route occur mostly in platoons. For example, consider for example the route order [this concept has not been introduced at this point!] $\eta = (0, 1, 1, 0, 0, 1, 1, 1, 0, 0)$. 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 2.1.

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*.

2.4 Notes and references

The single intersection model is mostly based on the model described in the PhD thesis of Hult [19, Chapter 3]. The bilevel decomposition and approximation scheme is further detailed in [5, 13]. For a textbook introduction of optimal control theory from the perspective of mathematical analysis, we recommend the book of Liberzon [9]. 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. [10].

A solid introduction to the algorithmic foundations for integer programming is provided by the book [20], whose introductory chapter also contains a clear description of the branch-and-cut methodology.

Chapter 3

Optimal schedule modeling

Most methods that are based on some kind of branch-and-bound tree search are guaranteed to find an optimal solution, but as we illustrated in Section 2.3.2, without further modifications, the running time generally scales very poorly with increased instance sizes. For this reason, we are interested in developing heuristics to obtain good approximations in limited time.

Specifically, we will show how (C) can be viewed as a sequence modeling problem. After precisely stating the model parameterization, we will show how to use randomly generated problem instances for model fitting. There are two general approaches: the most natural way is to use an existing solution method to solve these instances, then use the optimal or near-optimal solutions as training data points in a supervised learning setting. However, this assumes our ability to solve problems in the first place. An alternative approach is to collect data points in an online fashion by applying the current model and then updating, which is best understood in the framework of reinforcement learning.

Equivalent representations of schedules. Observe that the space of feasible solutions of (C) can be reduced to finitely many decisions regarding the disjunctive constraints (C.3). Specifically, for each feasible choice of the disjunctions $\mathcal{O}$, we consider the so-called *active schedule* y as the unique solution to the linear program [why unique?]

$$\begin{aligned} \min_{y} \quad & \sum_{i \in \mathcal{N}} y_i && && \text{(3.1a)} \\ \text{s.t.} \quad & a_i \leq y_i && \text{for all } i \in \mathcal{N}, && \text{(3.1b)} \\ & y_i + \rho_i \leq y_j, && \text{for all } (i,j) \in \mathcal{C}, && \text{(3.1c)} \\ & y_i + \sigma_i \leq y_j, && \text{for all } (i,j) \in \mathcal{O}. && \text{(3.1d)} \end{aligned}$$

To see when such a selection $\mathcal{O}$ is feasible, we can think of this linear program as a weighted directed graph on nodes $\mathcal{N}$ with conjunctive arcs $(i,j) \in \mathcal{C}$ with weights $w(i,j) = \rho_i$ and disjunctive arcs $(i,j) \in \mathcal{O}$ with weights $w(i,j) = \sigma_i$. Now observe that the definition of y through the linear program is equivalent to defining y through

$$y_j = \max\{a_j, \max_{i \in \mathcal{N}^-(j)} y_i + w(i,j)\}, \tag{3.2}$$

where $\mathcal{N}^-(j)$ denotes the set of in-neighbors of node j. It is now easy to see that y is well-defined if and only if the so-called *complete disjunctive graph* $G = (\mathcal{N}, \mathcal{C} \cup \mathcal{O})$ is acyclic. When G is acyclic, there is a unique topological ordering of its nodes. Hence, any valid selection $\mathcal{O}$ of disjunctive constraints is equivalent to a sequence π of vehicles. It is also equivalent to a sequence η of routes, because the ordering of vehicles on the same route is already fixed. See Figure 3.1 for an example of a complete disjunctive graph with corresponding vehicle order π and route order η. From now on, we will mainly be working with η, because it is in some sense the simplest representation of a schedule and we write y^η to denote the corresponding induced crossing times.

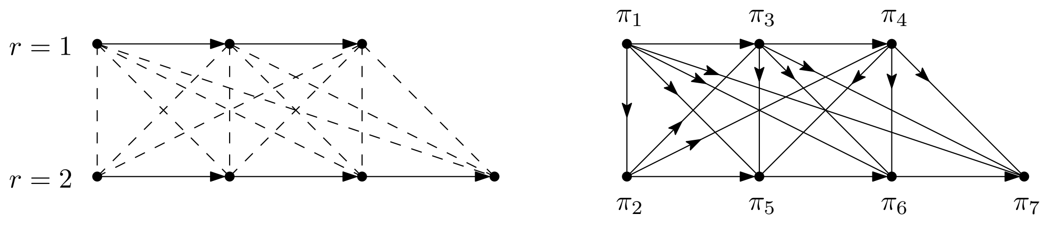


Figure 3.1: Illustration of disjunctive graphs for some instance with two routes with $n_1 = 3$ and $n_2 = 4$ vehicles. Horizontal arrows are the conjunctive arcs, the rest are disjunctive arcs. The left graph is and *empty* disjunctive graph, because $\mathcal{O} = \varnothing$. The selection of disjunctions shown in the *complete* graph on the right encodes the vehicle order $\pi = ((1,1),(2,1),(1,2),(1,3),(2,2),(2,3),(2,4))$ and route order $\eta = (1,2,1,1,2,2,2)$.

Sequence modeling for schedules. Given some problem instance s, we now want to model the sequence η such that y^η is an optimal schedule. To this end, we model the conditional probability of η given s by considering autoregressive models of the form

$$p(\eta|s) = \prod_{t=1}^{N} p(\eta_t | s, \eta_{1:t-1}), \tag{3.3}$$

where N denotes the total number of vehicles. Now consider some distribution of problem instances $\mathcal{X}$, then our goal is to minimize the expected value

$$\mathbb{E}_{s\sim\mathcal{X},\eta\sim p(\eta|s)} L(s,\eta),$$

of the total vehicle delay

$$L(s,\eta) = \sum_{i\in\mathcal{N}} y_i^\eta - a_i.$$

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

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

but this is generally very expensive to compute, because this could require $O(|\mathcal{R}|^N)$ evaluations of $p(\eta_t|s,\eta_{1:t-1})$ when we do not make additional structural model assumption. Therefore, one often uses *greedy rollout*, which means that we pick η_t with the highest probability at every step. Other inference strategies have been proposed in the context of modeling combinatorial optimization problems, see for example the “Sampling” and “Active Search” strategies in the seminal paper [21].

3.1 Model parameterization

We can consider different ways of parameterizing $p(\eta|s)$ in terms of $p(\eta_{t+1}|s,\eta_{1:t})$. Here, each partial sequence $\eta_{1:t}$ represents some partial schedule, which can equivalenty be defined in terms of the sequence of *scheduled* vehicles $\pi_{1:t}$. However, it is more convenient to define a partial schedule in terms of the *partial disjunctive graph* $G_t = (\mathcal{N}, \mathcal{C} \cup \mathcal{O}_t)$, which is defined by the unique selection $\mathcal{O}_t$ such that for each $i \in \pi_{1:t}$ and each $\{i,j\} \in \mathcal{D}$, we have either $(i,j) \in \mathcal{O}_t$ or $(j,i) \in \mathcal{O}_t$ with $j \in \pi_{1:t}$. To emphasize this parameterization in terms of the partial disjunctive graph, we can alternatively write (3.3) as

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

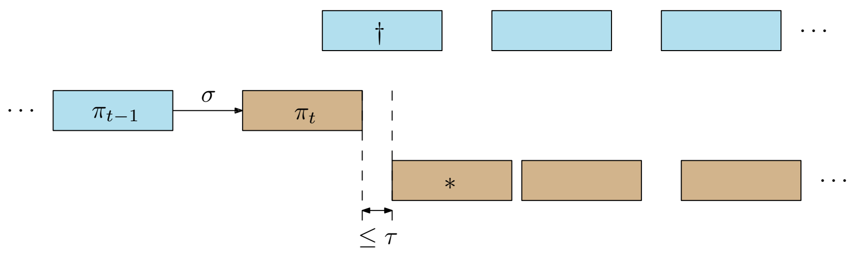


Figure 3.2: Illustration of how the threshold heuristic is evaluated at some intermediate step t to choose the next route η_{t+1}. The top and bottom row contains the unscheduled vehicles from route 1 and route 2, respectively, drawn at their earliest crossing times $\beta_t(i)$. The middel row represents the current partial schedule. Vehicle π_{t-1} is from route 1 and the last scheduled vehicle π_t is from route 2 and the disjunctive constraint for them happens to be tight in this case, illustrated by the arrow. Whenever the indicated distance is smaller than τ, the threshold rule selects vehicle $*$ to be scheduled next. Otherwise, vehicle † will be chosen.

There is a natural extensions of expression (3.2) for partial disjunctive graphs. Given G_t, let the *earliest crossing time* of each vehicle $i \in \mathcal{N}$ be recursively defined as

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

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

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

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

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

where $\texttt{next}(\eta_t)$ denotes some arbitrary route other than η_t with unscheduled vehicles left. The threshold heuristic is illustrated in Figure 3.2.

Neural parameterization. We will now consider a parameterizaton that directly generalizes the threshold heuristic. Instead of looking at the earliest crossing time of the next vehicle in the current lane, we now consider the earliest crossing times of all unscheduled vehicles across lanes. In the following definitions, we will drop the step index t to avoid cluttering the notation. For every route r, let π^r denote the sequence of unscheduled vehicles and consider their crossing time lower bounds $\beta(\pi^r) = (\beta(\pi_1^r), \beta(\pi_2^r), \dots)$. Let the minimum crossing time lower bound among unscheduled vehicles be denoted by T, then we call $h_r = \beta(\pi^r) - T = (\beta(\pi_1^r) - T, \beta(\pi^r) - T, \dots)$ the *horizon* of route r.

Next, we define some neural embedding $\bar{h}_r$ of each horizon. Observe that horizons can be variable length. We could fix the length by using padding, but this can be problematic

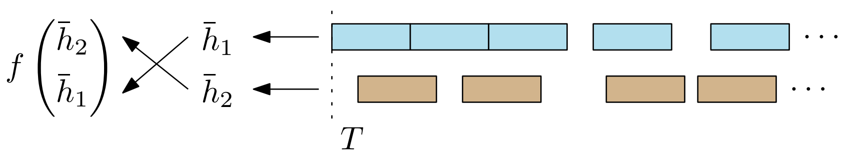


Figure 3.3: Schematic overview of parameterization of the neural heuristic. The distribution over the next route is parameterized as $p_\theta(\eta_{t+1}|G_t)$, which is computed from the current route horizons h_r via embeddings $\bar{h}_r$ and the cyling trick. Observe that the particular cycling shown in the figure corresponds to a situation in which the current route is $\eta_t = 2$.

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

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

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

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

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

3.2 Parameter fitting

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

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

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

Consider some instance $s \in X$ and let η^* denote some optimal route sequence, which can for example be computed by solving the integer program (C') from Section 2.3. For each such optimal schedule, we can compute the sequence $G_0, \eta_1, G_1, \eta_2, \ldots, \eta_N, G_N$. The resulting set of pairs $\{(G_t, \eta_{t+1}) : t = 1, \ldots, N-1\}$ can be used to learn p_θ in a supervised fashion by treating it as a classification task and computing the maximum likelihood estimator $\hat{\theta}$. Interpreting G_t as *states* and η_{t+1} as *actions*, we see that this approach is equivalent to *imitation learning* in the context of finding policies for Markov decision processes from so-called *expert demonstration.* Let Z denote the set of all state-action pairs collected from all training instances X. We make the procedure concrete for the case of two routes $\mathcal{R} = \{1, 2\}$, which is slightly simpler. Let $p_\theta(G_t)$ denote the probability of choosing the first route, then we can use the binary cross entropy loss, given by

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

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

3.3 Reinforcement learning

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

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

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

Let the return at step t be defined as

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

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

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

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

Policy-based methods work with an explicit parameterization of the policy. The model parameters are then tuned based on experience, often using some form of (stochastic) gradient descent to optimize the expected total return. Therefore, the gradient of the expected return plays a central role. The following identity is generally known as the Policy Gradient Theorem:

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

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

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

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

3.4 Results

The evalutation of model performance is roughly based on two aspects. Of course, the quality of the produced solutions is important. Second, we need to take into account the time that the algorithm requires to compute the solutions. We need to be careful here, because we have both training time as well as inference time for sequence models. We study the effect of the problem instance distribution $\mathcal{X}$ by varying the number of routes and number of arrivals per route, distribution of interarrival times, arrival intensity per route and degree of platooning.

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

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

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

The performance of the threshold heuristic is evaluated based on optimal solutions obtained using MILP in Table 3.1. With the specific choice $\tau = 0$, the threshold rule is related to the so-called *exhaustive policy* for polling systems, which is why we consider this case separately. We plot the average objective for the values of τ in the grid search, see Figure **??**.

The neural heuristics is trained for a fixed number of training steps. At regular intervals, we compute the average validation loss and store the current model parameters. At the end of the training, we pick the model parameters with the smallest validation loss. The results are listed in Table 3.2. For the neural heuristic with supervised (imitation) learning, we plot the training and validation loss, see Figure **??**. It can be seen that the model converges very steadily in all cases. For the policy gradient method using REINFORCE with baseline, the training loss with episodic baseline is shown in Figure **??** and for the stepwise baseline in Figure **??**.

Table 3.1: Performance evaluation of the branch-and-cut (MILP) approach and the threshold heuristic for different classes of instances with two routes. The first two columns specify the instance class based on the number of vehicles n per route and the type of arrival distribution for each route. These arrival distributions are chosen such that the arrival intensity is the same, only the degree of platooning varies. Performance is measured in terms of $L(s, \eta)/N(s)$, averaged over 100 test instances. The optimality gap is shown in parentheses for the heuristics. The threshold heuristic is fitted based on 100 training instances and the optimal threshold and training time is indicated. For branch-and-cut the average inference time is indicated. Note that we used a time limit of 60 seconds for all the branch-and-cut computations.

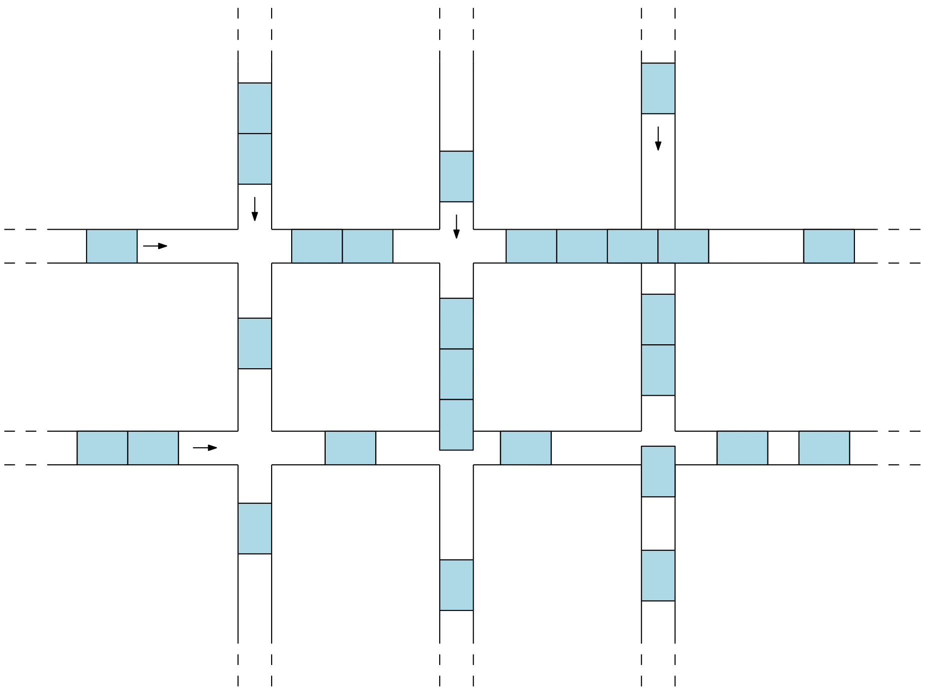
n	type	MILP	time	exhaustive (gap)	threshold (gap)	τ_{opt}	time
10	low	5.29	0.05	9.49 (79.45%)	7.78 (47.08%)	0.95	3.79
30	low	8.60	2.01	12.72 (47.88%)	11.31 (31.49%)	2.65	21.04
50	low	11.03	13.78	16.56 (50.20%)	14.60 (32.41%)	1.95	51.09
10	med	4.46	0.06	7.20 (61.32%)	6.39 (43.15%)	2.50	3.79
30	med	6.99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

Table 3.2: Comparison of neural heuristics with supervised learning and reinforcement learning, based on average delay per vehicle for different classes of instances with two routes. The first two columns specify the instance class based on the number of vehicles n per route and the type of arrival distribution for each route. These arrival distributions are chosen such that the arrival intensity is the same, only the degree of platooning varies. The supervised heuristic is fitted based on 100 train instances and results averaged over 100 test instances. We use two different ways to compute the baseline: single baseline per episode, or baseline for every step during the episode.

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

Part II

Network of Intersections



Chapter 4

Capacitated lanes

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

4.1 Model formulation

We will propose a model for a one-directional single-lane road of finite length where overtaking is not permitted. We will link such lanes together to construct a model representing a network of intersections. The fact that a lane between two intersections is of limited length—unlike the approaching lanes of the isolated intersection model—means that it can be occupied by a limited number of vehicles at the same time. This property 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 satisfies these *schedule times*. Loosely speaking, we want an easy way to answer this question for any set of schedule times. By requiring vehicles to enter and exit the lane at *full speed*, we will show that this feasibility question is precisely answered by a system of linear inequalities in terms of the schedule times.

Because there is generally not a single feasible set of trajectories, we can consider some performance criterion, for example by measuring some sort of smoothness to model passenger comfort or energy consumption. The resulting optimal control problem is straightforward to solve using a direct transcription method. However, when we seek to minimize each vehicle's distance to the end of the lane at all times, to which we refer as the *haste objective*, we will show that the optimal solution can be computed much more efficiently. We will see that the derivation of this algorithm is a direct byproduct of the feasibility analysis, because the latter will involve the construction of a certain set of upper bounding trajectories, which happen to be optimal under the haste objective.

In the remainder of this introducion, we will precisely establish the notion of a feasible set of trajectories in a lane. After precisely defining the optimal control problem under study, we show some examples of feasible trajectories for the haste objective and for the minimization of some proxy for energy consumption.

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

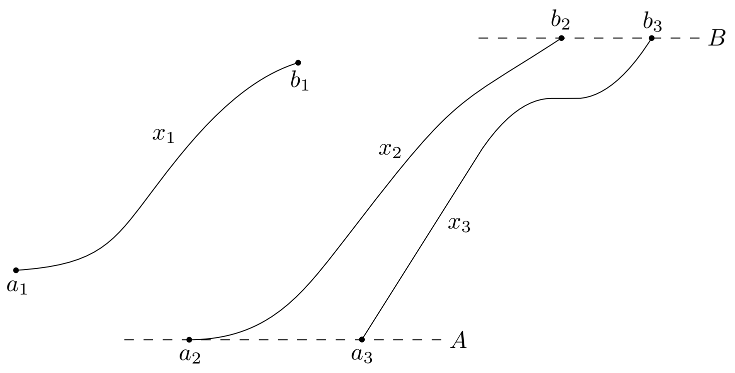


Figure 4.1: Example trajectories $x_1 \in \mathcal{D}[a_1, b_1]$, $x_2 \in D_1[a_2, b_2]$ and $x_3 \in D_2[a_3, b_3]$ for each the three classes of trajectories that are used throughout this chapter (horizontal axis is time, vertical axis is position).

the set of valid *trajectories*, which we define to be all continuously differentiable functions $x : [a, b] \to \mathbb{R}$ satisfying the constraints

$$\dot{x}(t) \in [0, 1] \quad \text{and} \quad \ddot{x}(t) \in [-\omega, \bar{\omega}], \quad \text{for all } t \in [a, b], \tag{4.1}$$

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

$$x(a) = A \text{ and } x(b) = B \tag{4.2}$$

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

$$\dot{x}(a) = \dot{x}(b) = 1. \tag{4.3}$$

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 4.1 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 4.1, then $x_1 \preceq x_3$ (void) and $x_3 \preceq x_2$, but clearly $x_1 \not\preceq x_2$.

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

Definition 4.1. 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

$$x_i \in D_2[a_i, b_i] \qquad \text{for each } i \in \{1, \dots, N\}, \tag{4.4a}$$

$$x_i \preceq x_{i-1} - L \qquad \text{for each } i \in \{2, \dots, N\}. \tag{4.4b}$$

We will refer to (4.4b) 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

$$\max \sum_{i=1}^{N} J(x_i). \tag{4.5}$$

We emphasize again that (4.4a) requires vehicles to enter and exit the lane at full speed. The feasibility characterization that we will derive can now be roughly stated as follows. Assuming the system parameters $(\omega, \bar{\omega}, A, B, L)$ to be fixed, with lane length $B-A$ sufficiently large and following distance L sufficiently small, feasibility of the lane planning problem is characterized by a system of linear inequalities in terms of the schedule times a_i and b_i.

Choice of objective. Probably better to introduce these examples earlier on, i.e., in single intersection chapter. We obtain what we call the *haste objective* by choosing

$$J(x_i) = \int_{a_i}^{b_i} x_i(t)\,\mathrm{d}t. \tag{4.6}$$

Roughly speaking, this objective seeks to keep all vehicles as close to the end of the lane at all times, but it does not capture energy efficiency in any way. Introduce energy objective and present some examples for both for comparison. In the rest of this chapter, we will show that optimal trajectories under the haste objective can be understood as the concatenation of at most four different types of trajectory parts, which we might call *bang-off-bang.* Based on this observation, we present an algorithm to compute optimal trajectories. Extending this algorithm to objectives like [the energy objective], is an interesting topic for further research.

4.2 Single vehicle with arbitary lead vehicle constraint

Before we analyze the feasibility of the lane planning problem as a whole, we focus on the lead vehicle constraint (4.4b) 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 4.2. 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 $x \preceq u$.

4.2.1 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 4.2. 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

$$x^1(t) = A + t - a, \tag{4.7}$$

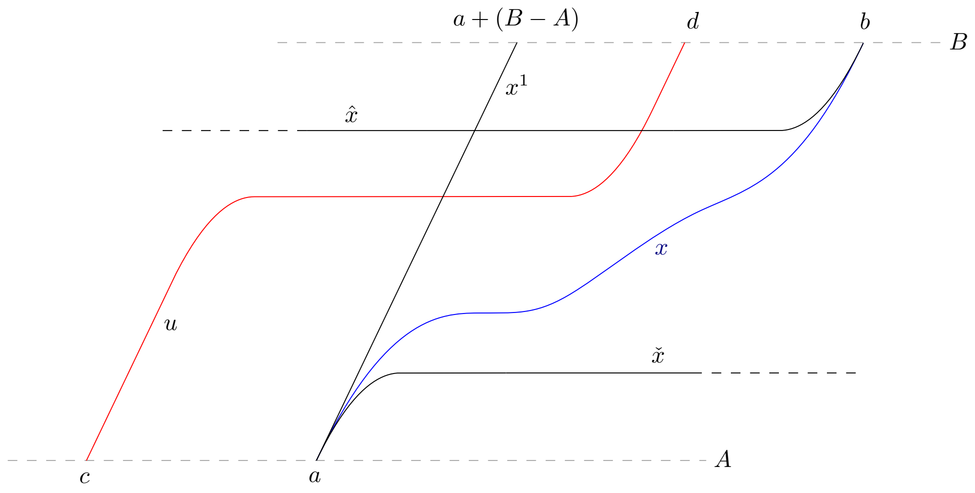


Figure 4.2: Illustration of the four bounding trajectories $u, x^1, \hat{x}, \check{x}$ that bound feasible trajectories from above and below. We also drew an example of a feasible trajectory x in blue. The horizontal axis represents time and the vertical axis corresponds to the position on the lane, so the vertical dashed grey lines correspond to the start and end of the lane.

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 4.1. *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

$$x(t) = x(a) + \int_a^t \dot{x}(\tau)\,\mathrm{d}\tau \geq A + \int_a^t \max\{0, 1 - \omega(\tau - a)\}\,\mathrm{d}\tau =: \check{x}(t), \tag{4.8}$$

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

$$x(t) = x(b) - \int_t^b \dot{x}(\tau)\,\mathrm{d}\tau \leq B - \int_t^b \max\{0, 1 - \bar{\omega}(b - \tau)\}\,\mathrm{d}\tau =: \hat{x}(t), \tag{4.9}$$

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

Lemma 4.2. *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 $x \in D_2[a, b]$. $\square$

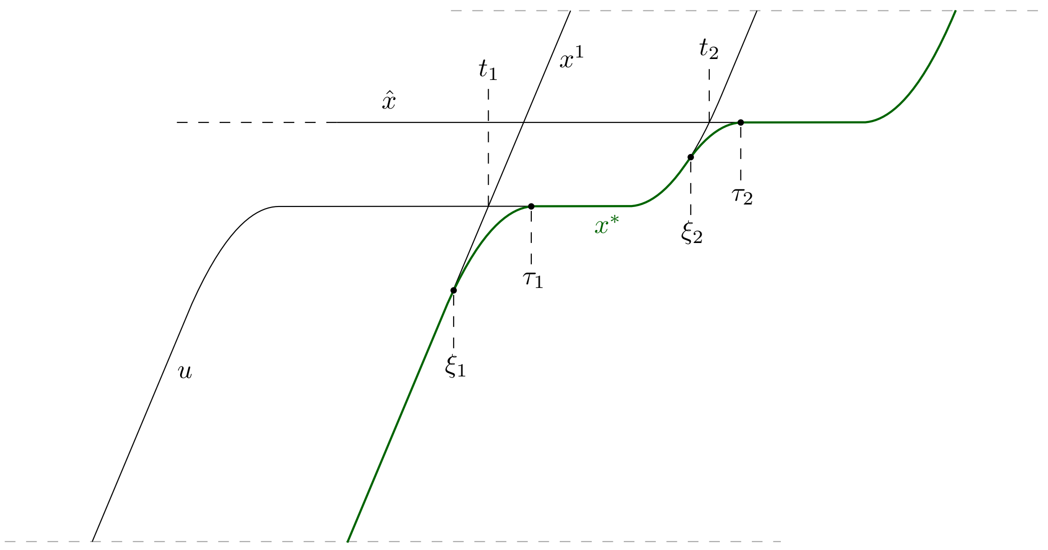


Figure 4.3: The minimum boundary γ, induced by three upper boundaries u, $\hat{x}$ and x^1, is smoothened around time t_1 and t_2, where the derivative is discontinuous, to obtain the smooth optimal trajectory x^*, drawn in green. The times ξ_i and τ_i correspond to the start and end of the connecting deceleration as defined in Section 4.2.4.

Remark 4.1. *The assumption of non-empty domains is required in the previous lemma, because otherwise we include the situation in which x lies completely to the left of u, in which case the stated conditions are obviously not necessary anymore.*

We note that the boundaries $\hat{x}$ and $\check{x}$ can be combined to yield yet another necessary condition. It is straightforward to verify from equations (4.8) and (4.9) that $\hat{x}(t) \geq B - 1/(2\bar{\omega})$ and $\check{x}(t) \leq A + 1/(2\omega)$. Therefore, whenever $B - A < 1/(2\bar{\omega}) + 1/(2\omega)$, these boundaries 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 4.1. *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 4.1 as requiring enough space in the lane such that there is at least one *waiting position.* We will return to this observation in Section 4.3. (check that we do this)

4.2.2 Sufficient conditions

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

Theorem 4.1 (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 4.1, 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 4.1 and Lemma 4.2 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

$$\gamma(t) := \min\{u(t), \hat{x}(t), x^1(t)\}. \tag{4.10}$$

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

Definition 4.3. Let $\mathcal{P}[a, b]$ be the set of functions $\mu : [a, b] \to \mathbb{R}$ for which there is a finite subdivision $a = t_0 < \cdots < t_{n+1} = b$ such that the truncation $\mu|_{[t_i,t_{i+1}]} \in \mathcal{D}[t_i, t_{i+1}]$ is a smooth trajectory, for each $i \in \{0, \ldots, n\}$, and for which the one-sided limits of $\dot{\mu}$ satisfy

$$\dot{\mu}(t_i^-) := \lim_{t \uparrow t_i} \dot{\mu}(t) > \lim_{t \downarrow t_i} \dot{\mu}(t) =: \dot{\mu}(t_i^+), \tag{4.11}$$

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

Under the conditions of Theorem 4.1, it is not difficult to see from Figure 4.2 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 4.3. We will argue that this procedure can be repeated as many times as necessary to smoothen out every discontinuity.

In Section 4.2.3, 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 4.2.4. 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 4.2.5.

4.2.3 Deceleration boundary

Recall the derivation of $\check{x}$ in equation (4.8) and the discussion preceding it, which we will now generalize a bit. Let $x \in \mathcal{D}[a, b]$ be some smooth trajectory, then observe that $\dot{x}(t) \geq \dot{x}(\xi) - \omega(t - \xi)$ for all $t \in [a, b]$. Combining this with the constraint $\dot{x}(t) \in [0, 1]$, this yields

$$\dot{x}(t) \geq \max\{0, \min\{1, \dot{x}(\xi) - \omega(t - \xi)\}\} =: \{\dot{x}(\xi) - \omega(t - \xi)\}_{[0,1]}, \tag{4.12}$$

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

$$x(t) = x(\xi) + \int_\xi^t \dot{x}(\tau)\, \mathrm{d}\tau \geq x(\xi) + \int_\xi^t \{\dot{x}(\xi) - \omega(\tau - \xi)\}_{[0,1]}\, \mathrm{d}\tau =: x[\xi](t), \tag{4.13}$$

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

Note that $x[\xi]$ depends on x only through the two real numbers $x(\xi)$ and $\dot{x}(\xi)$. It will be convenient later to rewrite the right-hand side of (4.13) as

$$x^-[p, v, \xi](t) := p + \int_\xi^t \{v - \omega(\tau - \xi)\}_{[0,1]}\, \mathrm{d}\tau, \tag{4.14}$$

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

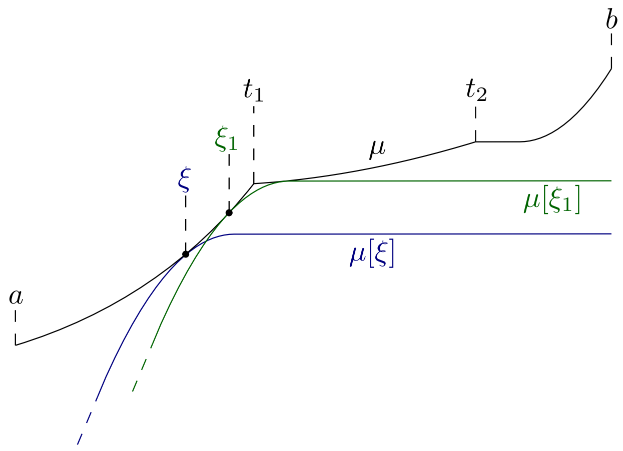


Figure 4.4: Illustration of some piecewise trajectory $\mu \in \mathcal{P}[a, b]$ with a discontinuous derivative at times t_1 and t_2. Furthermore, the figure shows some arbitrary deceleration boundary $\mu[\xi]$ at time ξ in blue and the unique connecting deceleration $\mu[\xi_1]$ to the cover discontinuity 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 (4.10), but please note that the class of piecewise trajectories $\mathcal{P}[a, b]$ is just slightly more general than necessary for our current purposes.

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

$$x^-[p, v, \xi](t) = p + \begin{cases} (1-v)^2/(2\omega) + (t - \xi) & \text{for } t \leq \delta_1, \\ v(t - \xi) - \omega(t - \xi)^2/2 & \text{for } t \in [\delta_1, \delta_0], \\ v^2/(2\omega) & \text{for } t \geq \delta_0. \end{cases} \tag{4.15}$$

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

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 4.3. It is straightforward to generalize the definition of a deceleration boundary to μ. Whenever $\xi \in [a, b] \setminus \{t_1, \dots, t_n\}$, we just define $\mu[\xi] := x^-[\mu(\xi), \dot{\mu}(\xi), \xi]$, exactly like we did for x. However, at the points of discontinuity $\xi \in \{t_1, \dots, t_n\}$, the derivative $\dot{\mu}(\xi)$ is not defined, so we choose to use the left-sided limit instead, by defining $\mu[\xi] := x^-[\mu(\xi), \dot{\mu}(\xi^-), \xi]$.

Remark 4.2. *Please note that we cannot just replace x with μ in inequality* (4.13) *to obtain a similar bound for μ on the its full interval $[a, b]$. Instead, we get the following* piecewise lower bounding *property. Consider some interval $I \in \{[a, t_1], (t_1, t_2], \dots, (t_n, b]\}$, then what remains true is that $\xi \in I$ implies $\mu(t) \geq \mu[\xi](t)$ for every $t \in I$.*

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

4.2.4 Smoothing procedure

Let $\mu \in \mathcal{P}[a,b]$ be some piecewise trajectory and let $a = t_0 < \cdots < t_{n+1} = b$ denote the subdivision as in Definition 4.3. 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 4.4 for an example. The construction relies on the following technical assumption.

Assumption 4.2. *Throughout the following discussion, we assume $\mu \succeq \mu[a]$ and $\mu \succeq \mu[b]$.*

Touching. Recall Remark 4.2, which asserts that we have $\mu[\xi](t) \leq \mu(t)$ for every $t \in [a, t_1]$ for any $\xi \in [a, t_1]$. After the discontinuity, so for every $t \in [t_1, b]$, we want $\mu[\xi](t) \leq \mu(t)$ and equality at least somewhere, so we measure the relative position of $\mu[\xi]$ with respect to μ here, by considering

$$d(\xi) := \min_{t \in [t_1, b]} \mu(t) - \mu[\xi](t). \tag{4.16}$$

Since $\mu(t)$ and $\mu[\xi](t)$ are both continuous as a function of t on the interval $[t_1, b]$, this minimum actually exists (extreme value theorem). Furthermore, since d is the minimum of a continuous function over a closed interval, it is continuous as well (see Lemma E.1). Observe that $d(a) \geq 0$, because $\mu \succeq \mu[a]$ by Assumption 4.2. 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

$$X := \{\xi \in [a, t_1) : d(\xi) = 0\} \tag{4.17}$$

is a closed interval. To this end, we show that d is non-increasing on $[a, t_1)$, which together with continuity implies the desired result (see Lemma E.2). To show that d is non-increasing, it suffices to show that $\mu[\xi](t)$ is non-decreasing as a function of ξ, for every $t \in [t_1, b]$. We can do this by computing the partial derivative of $\mu[\xi]$ with respect to ξ and verifying it is non-negativity. Recall the definition of $\mu[\xi]$, based on x^- in equation (4.15). Using similar notation, we write $\delta_1(\xi) = \xi - (1 - \dot{\mu}(\xi))/\omega$ and $\delta_0(\xi) = \xi + \dot{\mu}(\xi)/\omega$ and compute

$$\frac{\partial}{\partial \xi} \mu[\xi](t) = \dot{\mu}(\xi) + \begin{cases} \ddot{\mu}(\xi)(\dot{\mu}(\xi) - 1)/\omega - 1 & \text{for } t \leq \delta_1(\xi), \\ \ddot{\mu}(\xi)(t - \xi) - \dot{\mu}(\xi) + \omega(t - \xi) & \text{for } t \in [\delta_1(\xi), \delta_0(\xi)], \\ \ddot{\mu}(\xi)\dot{\mu}(\xi)/\omega & \text{for } t \geq \delta_0(\xi). \end{cases} \tag{4.18}$$

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

$$\frac{\partial}{\partial \xi} \mu[\xi](t) = (\ddot{\mu}(\xi) + \omega)(t - \xi) \geq 0 \qquad \text{for } t \in [\delta_1(\xi), \delta_0(\xi)], \tag{4.19a}$$

$$\frac{\partial}{\partial \xi} \mu[\xi](t) \geq \dot{\mu}(\xi) + (-\omega)\dot{\mu}(\xi)/\omega = 0 \qquad \text{for } t \geq \delta_0(\xi). \tag{4.19b}$$

This concludes the argument for X being a closed interval.

Assuming ξ to be fixed, observe that there is equality in (4.19a) for some $t \in [\delta_1(\xi), \delta_0(\xi)]$ if and only if there is equality in (4.19b) for some other $t' \geq \delta_0(\xi)$. Note that this happens precisely when $\ddot{\mu}(\xi) = -\omega$. Therefore, whenever μ is fully deceleration, so $\dot{\mu}(t) = -\omega$ on some open interval $U \subset (a, t_1)$, we have $(\partial/\partial\xi)\mu[\xi](t) = 0$ for all $t \geq \delta_1(\xi)$. This essentially means that any choice of $\xi \in U$ produces the same trajectory $\mu[\xi]$. Please see Figure 4.5 for an example of this case. This observation is key to the remaining uniqueness argument.

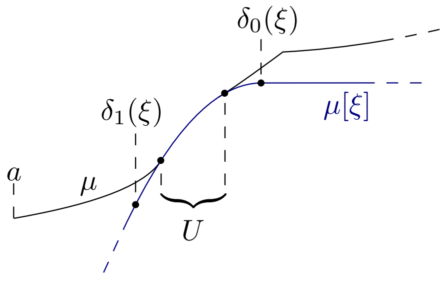


Figure 4.5: Example of a piecewise trajectory μ with a part of full deceleration over some interval U such that any choice of $\xi \in U$ produces the same deceleration boundary $\mu[\xi]$, which naturally coincides with μ on U.

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

$$\frac{\partial}{\partial\xi}\mu[\xi](t') > 0 \text{ for all } \xi \in U. \tag{4.20}$$

However, we argued in the previous paragraph that this actually holds for any $t' \geq \delta_1(\xi)$. In particular, let $t^* \in [t_1, b]$ be such that $\mu(t^*) = \mu[\xi_0](t^*)$, then $t^* \geq t_1 \geq \xi \geq \delta_1(\xi)$, so (4.20) yields $\mu[\xi'](t^*) > \mu[\xi_0](t^*)$, but then $d(\xi') > d(\xi_0) = 0$, so $\xi' \notin X$, a contradiction.

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

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

$$\dot{\mu}(t_1^+) > \dot{\mu}[\xi_0](t_1^+) = \dot{\mu}[\xi_0](t_1), \tag{4.21}$$

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

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

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

$$\dot{\mu}[b](t) \leq \dot{\mu}[\xi_0](t) \text{ for } t \leq b, \tag{4.22}$$

but then $\mu[b](t) > \mu[\xi_0](t)$ for $t < b$. In particular, for $t = \xi_0$, this shows $\mu[b](\xi_0) > \mu[\xi_0](\xi_0) = \mu(\xi_0)$, which contradicts part $\mu[b] \preceq \mu$ of Assumption 4.2.

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

$$\mu[\xi_0](t) \leq \mu(t) \quad \text{for} \quad t \in [\xi_0, \tau_1], \tag{4.23a}$$

$$\dot{\mu}[\xi_0](\xi_0) = \dot{\mu}(\xi_0) \quad \text{and} \quad \dot{\mu}[\xi_0](\tau_1) = \dot{\mu}(\tau_1). \tag{4.23b}$$

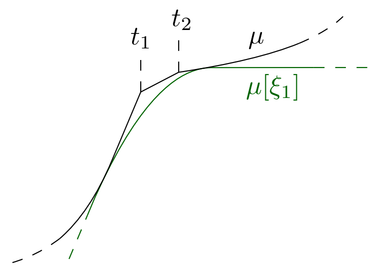


Figure 4.6: Part of a piecewise trajectory μ on which a single connecting deceleration covers the two discontinuities at t_1 and t_2 at once.

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

$$\mu_1(t) = \begin{cases} \mu(t) & \text{for } t \in [a, \xi_1] \cup [\tau_1, b], \\ \mu[\xi_1](t) & \text{for } t \in [\xi_1, \tau_1]. \end{cases} \tag{4.24}$$

From the way we constructed $\mu[\xi_1]$, it follows from (4.23) that we have $\mu_1 \in \mathcal{P}[a, b]$, but without the discontinuity at t_1. Observe that a single connecting deceleration may cover more than one discontinuity, as illustrated in Figure 4.6. 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 4.2, 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 4.1. Let us now return to the minimum boundary γ defined in (4.10). From Figure 4.2 and the conditions of Theorem 4.1, it is clear that γ must satisfy $\gamma(a) = A$, $\gamma(b) = B$ and $\dot{\gamma}(a) = \dot{\gamma}(b) = 1$, so whenever we have $\gamma \in \mathcal{D}[a, b]$, i.e., γ does not contain discontinuities, we automatically have $\gamma \in D_2[a, b]$ so that γ itself is already a feasible solution. Explain why Assumption 4.2 holds. Otherwise, we perform the smoothing procedure presented above to obtain the smoothed trajectory $\gamma^* \in \mathcal{D}[a, b]$. This completes the proof of Theorem 4.1.

4.2.5 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 4.3. *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

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

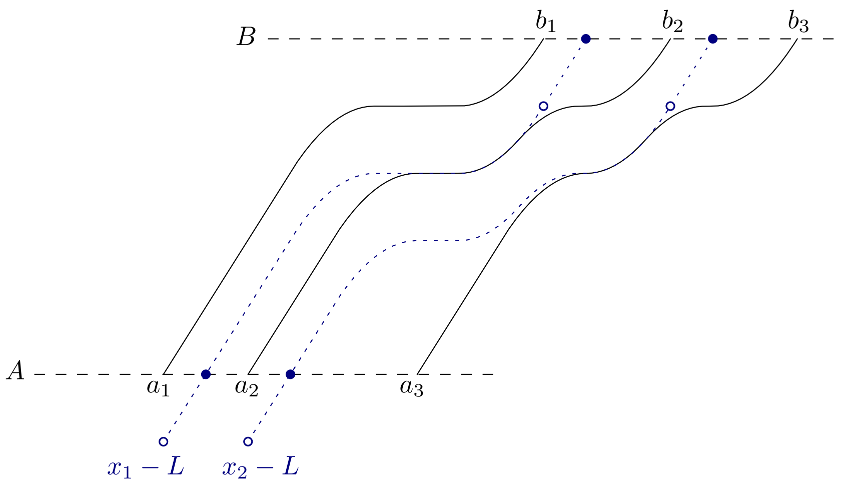


Figure 4.7: Optimal trajectories x_i for three vehicles. The dotted blue trajectories between the little open circles illustrates the safe following constraints (4.27b). The dotted blue trajectories between the solid dots are the following boundaries $\bar{x}_2 \in \bar{D}[\bar{a}_2, \bar{b}_2]$ and $\bar{x}_3 \in \bar{D}[\bar{a}_3, \bar{b}_3]$.

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

Remark 4.3. *The above upper boundary property has the following interesting consequence if we extend the single vehicle problem to an optimal control problem by considering maximizing the haste criterion $J(x)$ defined in* (4.6) *as optimization objective. In particular, observe that it follows from the above lemma that any other $x \in D_2[a, b]$ satisfying $x \preceq u$ must also satisfy*

$$\int_a^b x(t)\,\mathrm{d}t \leq \int_a^b \gamma^*(t)\,\mathrm{d}t \tag{4.25}$$

Consequently, γ^* is an optimal solution to the single vehicle optimal control problem

$$\max_{x \in D_2[a,b]} J(x) \;\; \textit{such that} \;\; x \preceq u. \tag{4.26}$$

4.3 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

$$\begin{aligned} &x_i \in D[a_i, b_i] && \text{for each } i \in \{1, \dots, N\}, && \text{(4.27a)} \\ &x_i \preceq x_{i-1} - L && \text{for each } i \in \{2, \dots, N\}. && \text{(4.27b)} \end{aligned}$$

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, which is very reasonable to assume in practice.

Assumption 4.3. *Assume that vehicle lengths are limited by $L < B - A$.*

Now consider the safe following constraints (4.27b). 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 4.1. It becomes clear from Figure 4.7 that inequality (4.27b) 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

$$y^{-1}(p) = \inf\{t \in [\alpha, \beta] : y(t) = p\}. \tag{4.28}$$

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

$$\bar{u}(t) = \begin{cases} u(t) - L & \text{for } t \in [u^{-1}(A + L), d], \\ B - L + t - d & \text{for } t \in [d, d + L]. \end{cases} \tag{4.29}$$

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

Lemma 4.4 (Boundary extension)**.** *Consider some trajectory $u \in \mathcal{D}[c, d]$ such that $u(d) \geq A$. If $x \in \mathcal{D}[a, b]$ is such that $x(a) = A$ and $x \preceq u$, then it satisfies $x \preceq (u(d) + t - d)|_{[d,\infty)}$, which may be interpreted as extending the upper boundary u to the right at full speed.*

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

Lemma 4.5 (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 4.4 to $u - L$ (using Assumption 4.3 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 concludes the summarizes what we have established so far.

Lemma 4.6. *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 $x_i \leq x_{i-1} - L$ for all $i \in \{2, \dots, N\}$.

(C2) There exists $x \in D_2^N[a, b]$ such that $x_i \leq \bar{x}_{i-1}$ 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)	$\bar{x}_i \geq \check{x}_i$.	*(entry space constraint)*

Proof. Of course, (C0) and (C1) are equivalent by definition. Note that equivalence of (C1) and (C2) is handled by Lemma 4.5. We show "(C2) $\implies$ (C3)". (simple) We show "(C3) $\implies$ (C2)". (induction) □

Simpler entry space constraints. [we will not proof this, but at least derive a^* and maybe sketch what we would need for the proof (why we find it difficult)]

The goal of the remainder of this section is to get rid of the entry space constraints and replace them with inequality constraints. More specifically, we will show that the above statements are further equivalent to

(C4) We have $b_i - a_i \geq B - A$ 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

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

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

Earliest arrival. We show that the entry order constraint $\bar{a}_i \leq a_i$ and the entry space constraint $\bar{x}_i \geq \check{x}_i$ together are equivalent to the two constraints

$$a_i \geq a_{i-1} + L, \tag{4.30a}$$
$$a_i \geq \check{a}_i(a, b), \tag{4.30b}$$

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 arrival time for vehicle i. Recall that $\bar{a}_i = x_{i-1}^{-1}(A + L)$.

Recall the definition of $\hat{x}$ in equation (4.9). By carefully handling the $\max\{\cdot\}$, we can expand this expression as

$$\hat{x}(t) = \begin{cases} B - b + t + \bar{\omega}(b - t)^2/2 & \text{for } t \geq b - 1/\bar{\omega}, \\ B - 1/(2\bar{\omega}) & \text{for } t \leq b - 1/\bar{\omega}. \end{cases} \tag{4.31}$$

Waiting capacity is given by

$$C = \left\lfloor \frac{B - A - 1/(2\bar{\omega}) - 1/(2\omega)}{L} \right\rfloor + 1, \tag{4.32}$$

where the brackets indicate the floor function.

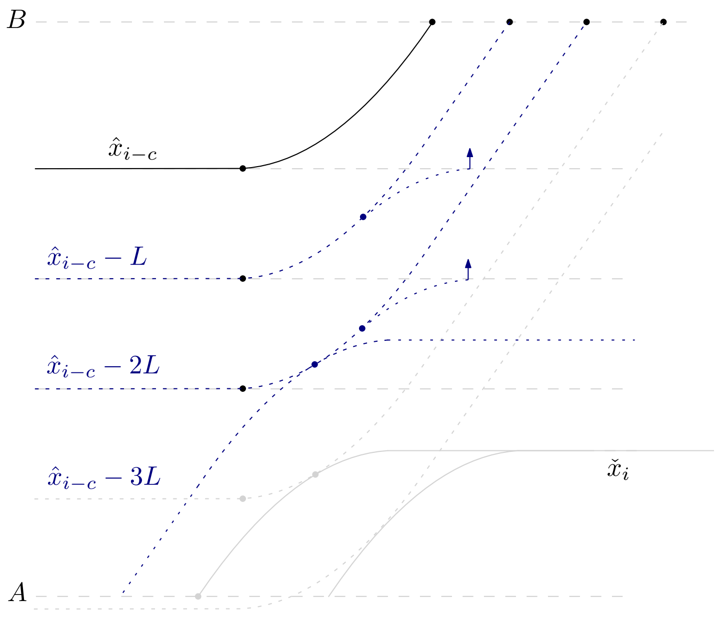


Figure 4.8: Earliest arrival due to entry order constraint and entry space constraint.

4.4 Optimal solution for haste objective

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

Definition 4.4. Let a trajectory $\gamma \in \mathcal{D}[a, b]$ be called *alternating* if for all $t \in [a, b]$, we have

$$\ddot{\gamma}(t) \in \{-\omega, 0, \bar{\omega}\} \quad \text{and} \quad \ddot{\gamma}(t) = 0 \implies \dot{\gamma}(t) \in \{0, 1\}. \tag{4.33}$$

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

Observe that an alternating trajectory $\gamma \in \mathcal{D}[a, b]$ can be described as a sequence of four types of consecutive repeating phases, see Figure 4.9 for an example. In general, there exists a partition of $[a, b]$, denoted by

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

such that we have the consecutive *alternating intervals*

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

such that on these intervals, γ satisfies

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

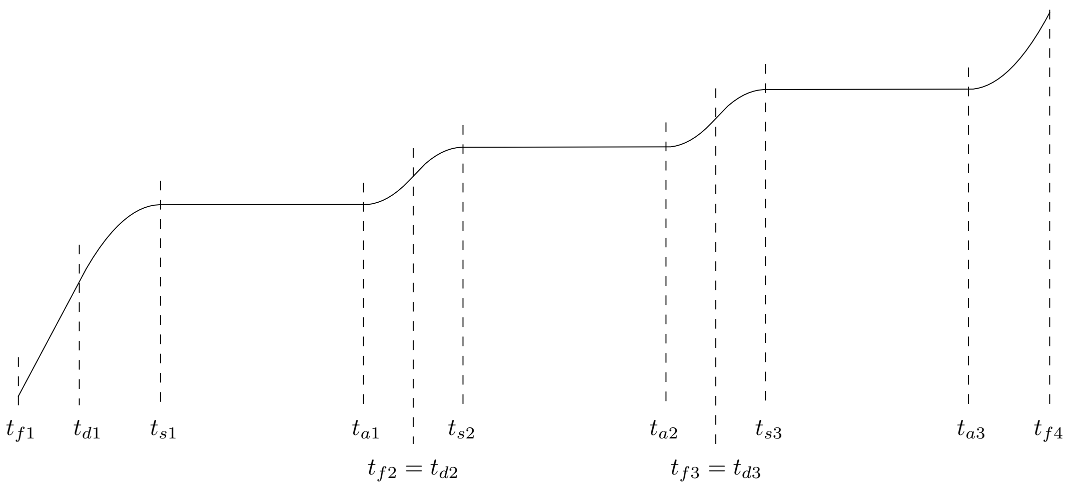


Figure 4.9: Example of an alternating vehicle trajectory with its defining time intervals. The particular shape of this trajectory is due to two leading vehicles, which causes the two start-stop *bumps* around the times where these leading vehicles depart from the lane.

Partial trajectories. Next, we will define parameterized functions x^f, x^d, x^s, x^a to describe alternating trajectory γ on each of these alternating intervals. Given some initial position $p \in [A, B]$, velocity $v \in [0, 1]$, start and end times a and b such that $a \leq b$ and $v + \bar{\omega}(b - a) \leq 1$, we define the acceleration trajectory $x^a[p, v, a, b] : [a, b] \to \mathbb{R}$ by setting

$$x^a[p, v, a, b](\tau) := p + v(\tau - a) + \bar{\omega}(\tau - a)^2/2, \tag{4.34a}$$

$$\dot{x}^a[p, v, a, b](\tau) := v + \bar{\omega}(\tau - a). \tag{4.34b}$$

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

$$x^d[p, v, a, b](\tau) := p + v(\tau - a) - \omega(\tau - a)^2/2, \tag{4.35a}$$

$$\dot{x}^d[p, v, a, b](\tau) := v - \omega(\tau - a). \tag{4.35b}$$

One may notice that x^d is essentially the same as the deceleration boundary x^-, which we defined in Section 4.2.3. However, note that the condition $v - \omega(b - a) \geq 0$ restricts the domain such that we do not need the clipping operation. Furthermore, the parameterization of x^d will be more convenient in the next section.

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

4.4.1 Connecting partial trajectories

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

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

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

Lemma 4.7 ($x^f \to x^s$)**.** *Let $x^f[p, a, b]$ and $x^s[q, c, d]$ be two trajectories. Considering τ_1 and τ_2 as variables in the equation*

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

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

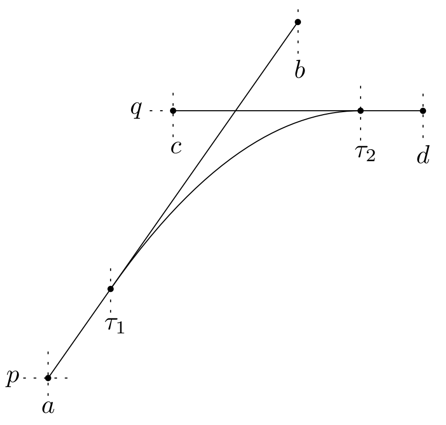


Figure 4.10: $x^f \to x^s$

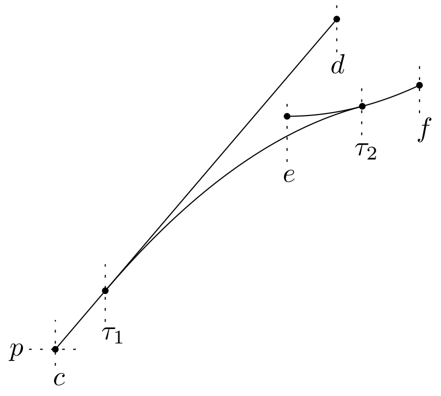


Figure 4.11: $x^f \to x^a$

Proof. The expanded system of state equations is given by

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

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

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

Lemma 4.8. *Consider some full acceleration trajectory $x^a[(p,0), a, a + 1/\omega]$ and the line through $(\lambda, 0)$ with slope 1. Whenever λ, which can be interpreted as a time epoch, satisfies $\lambda \in [a - p - 1/2\omega, a - p + 1/2\omega]$, then the equation*

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

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

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

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

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

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

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

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

which we substitute in the first equation to derive the equation

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

This is a quadratic equation in τ, with solutions

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

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

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

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

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

$\square$

Lemma 4.9 ($x^f \to x^a$). *Consider partial trajectories $x^f[p, c, d]$ and $x^a[x, e, f]$.*

Proof. First of all, observe that $x^f[p, c, d]$ lies on the line with slope 1 through $(\lambda, 0) := (c - p, 0)$ and $x^a[x, e, f]$ lies on the full acceleration curve $x^a[(x_1 - x_2^2/(2\omega), 0), e - x_2/\omega, e - x_2/\omega + 1/\omega]$, see Figure 4.11. Now apply Lemma 4.8 to $p = x_1 - x_2^2/(2\omega)$, $a = e - x_2/\omega$ and $\lambda = c - p$ yields some solutions τ and q. Let $\tau_2 := \tau$ and let τ_1 denote the time where the line and x^a join, given by $\tau_1 = \lambda + q$. Now we simply check whether this solution is also feasible for the smaller trajectories. We must have $\tau_1 \in [c, d]$ and $\tau_2 \in [e, f]$. $\square$

Lemma 4.10. *Consider the acceleration trajectory $x^a[(p, 0), a, b]$ and the horizontal line through $(0, q)$. Let $\tau_1 = a + \sqrt{(q - p)/\omega}$ and $\tau_2 = a + 2\sqrt{(q - p)/\omega}$. If τ_1 satisfies $\tau_1 \in [a, b]$, then both trajectories are joined by deceleration trajectory $x^d[x^a[(p, 0), a, b](\tau_1), \tau_1, \tau_2]$*

Proof. Consider the following equation

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

The expanded system of state equations is given by

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

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

Lemma 4.11 ($x^a \to x^s$). *Consider partial trajectories $x^a[x, c, d]$ and $x^s[q, e, f]$.*

Proof. Observe that $x^a[x, c, d]$ lies on the full acceleration curve $x^a[(x_1 - x_2^2/(2\omega), 0), c - x_2/\omega, c - x_2/\omega + 1/\omega]$. Hence, we can apply Lemma 4.10 with $p = x_1 - x_2^2/(2\omega)$, $a = c - x_2/\omega$, which yields some solutions τ_1 and τ_2, which are feasible solutions if $\tau_1 \in [c, d]$ and $\tau_2 \in [e, f]$. $\square$

Lemma 4.12. *Consider full acceleration trajectories $x^a[(p, 0), a, b]$ and $x^a[(q, 0), c, d]$.*

Proof. Consider the equation

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

expanded to the system of equations

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

$\square$

Lemma 4.13 ($x^a \to x^a$). *Consider partial trajectories $x^a[x, a, b]$ and $x^a[y, c, d]$.*

Proof. $\square$

Algorithm 1 Computing connecting deceleration for alternating trajectories.

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

4.4.2 Algorithm

Put everything together into pseudocode.

4.5 Notes and references

The analysis of feasibility conditions in this chapter is very much related to the work [22]. They study an online situation, in which new vehicles arrive to the system at later times. They also alow changing the crossing order with respect to conflicting lanes, which might require the current planned vehicle trajectories to change (slow down) to anticipate this change. We do not study such an online setting, but we take into account the fact that lanes are of finite length. [while revisiting the figures in their paper, it seems what they do is very related to my own attempt; I couldn't get through their appendix earlier]

There are different ways of relaxing the second-order constraints 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.,

$$\eta \leq \dot{x}(a) \leq 1 \quad \text{and} \quad \eta \leq \dot{x}(b) \leq 1, \tag{4.36}$$

for some $\eta > 0$.

Chapter 5

Learning to schedule in networks

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

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

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

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

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

This assumption ensures that each route $\bar{V}_r$ can be modeled by connecting a sequence of lanes together, with some *intersection areas* of some fixed size W in between them, see Figure 5.1. Hence, we set the longitudinal start and end position of each lane model as follows. Let $d(v, w)$ denote the length of edge $(v, w) \in E_r$, then we recursively define

$$A_{r1} = 0, \tag{5.1a}$$

$$A_{rk} = B_{r,k-1} + W + L, \tag{5.1b}$$

$$B_{rk} = A_{rk} + d(v_r(k-1), v_r(k)), \tag{5.1c}$$

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

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

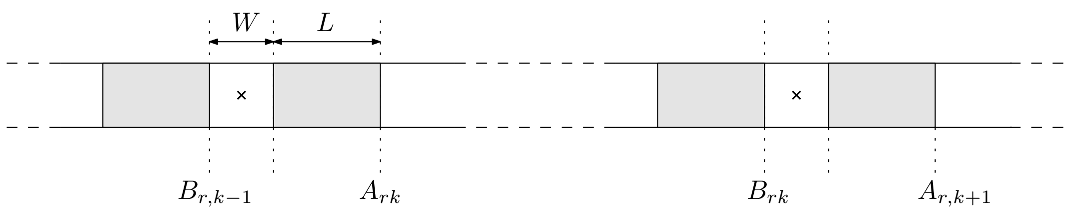


Figure 5.1: Illustration of how the individual lane models are connected to form a route with intersections, marked with a little cross. The four shaded rectangles illustrate four possible vehicle positions. The length of the intersection is W. The longitudinal positions A_{rk} and B_{rk} denote the start and end, respectively, of the kth lane on route r.

Assumption 5.1 ensures that the order of vehicles on each lane is completely determined by the order of vehicles on the corresponding lane.

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

Occupancy time slot scheduling

Chapter 6

Conclusion and discussion

Bibliography

[1] Stefano Mariani, Giacomo Cabri, and Franco Zambonelli. Coordination of Autonomous Vehicles: Taxonomy and Survey. *ACM Computing Surveys*, 54(1):1–33, January 2022.

[2] K. Dresner and P. Stone. A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research*, 31:591–656, March 2008.

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

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

[5] Robert Hult, Gabriel R. Campos, Paolo Falcone, and Henk Wymeersch. An approximate solution to the optimal coordination problem for autonomous vehicles at intersections. In *2015 American Control Conference (ACC)*, pages 763–768, Chicago, IL, USA, July 2015. IEEE.

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

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

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

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

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

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

[12] Matthew Kelly. An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation. *SIAM Review*, 59(4):849–904, January 2017.

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

[14] A. V. Dmitruk and A. M. Kaganovich. Maximum principle for optimal control problems with intermediate constraints. *Computational Mathematics and Modeling*, 22(2):180–215, April 2011.

[15] Ding-Zhu Du, Panos M. Pardalos, Xiaodong Hu, and Weili Wu. *Introduction to Combinatorial Optimization*, volume 196 of *Springer Optimization and Its Applications*. Springer International Publishing, Cham, 2022.

[16] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer International Publishing, Cham, 2016.

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

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

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

[20] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer Programming*, volume 271 of *Graduate Texts in Mathematics.* Springer International Publishing, Cham, 2014.

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

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

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

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

[25] Andrea Lodi and Giulia Zarpellon. On learning and branching: A survey. *TOP*, 25(2):207–236, July 2017.

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

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

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

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

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

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

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

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

[34] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

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

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

[37] 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.

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

Appendix

Appendix A

Job shop scheduling

["Readers unfamiliar with this topic may appreciate the brief introduction here."]

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 heterogenous 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 [16, 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 A.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 [16, Chapter 2]. Now consider a specific problem instance with $m = 3$ machines and $n = 2$ jobs. We specify the order in which jobs visit machines by providing the corresponding ordering of operations, which we choose to be $(1, 1) \to (2, 1) \to (3, 1)$ and $(3, 2) \to (2, 2) \to (1, 2)$. Using matrix notation $r(j) \equiv r_j$ and $p(i, j) \equiv p_{ij}$, the release dates and processing times are given by

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

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

[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.

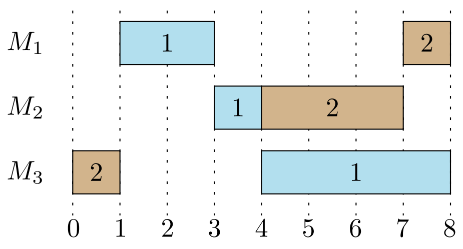


Figure A.1: Example of an optimal schedule for Example A.1, shown as a Gantt chart. Each row M_i corresponds to machine i and each block numbered j on this row represents the operation (i, j). The dashed lines indicate unit time steps. Note that machine 2 is kept idle, while operation $(2, 2)$ could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

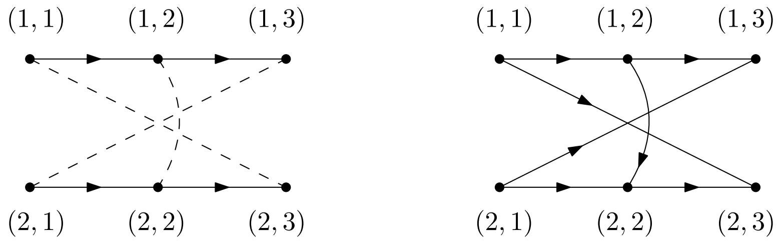


Figure A.2: Illustration of disjunctive graphs for Example A.1. Horizontal arrows represent conjunctive arcs. We used dashed lines to for the pairs of disjunctive arcs as dashed lines. The left graph corresponds to an empty selection $\mathcal{O} = \varnothing$ while the right graph shows the selection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

Disjunctive graph. A commonly used representation of job shop problems is through their disjunctive graph, which is a directed graph with vertices $\mathcal{N}$ corresponding to the operations and two types of arcs. The conjunctive arcs $\mathcal{C}$ are used to encode the predetermined machine sequence of each job. Each such arc $(i, j) \to (k, j)$ encodes that job j should first be processed 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 [16]. 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 A.1 are shown in Figure A.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 A.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 A.1. Using the notation of the disjunctive graph, the problem can be concisely stated as

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

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

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

which is generally refered to as the *big-M method.* The resulting MILP can be solved by any off-the-shelf solver, e.g., we used the commerical Gurobi Optimizer software [4] for this thesis.

Appendix B

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 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 [23].

Algorithm execution as MDPs. It is very natural to see the sequential decision-making process of any optimization algorithm in terms of the 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 [24]. 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 [25] 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 [26]. 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 [27]. Constraints can also be enforced by the factorization of the mapping into repeated application of some policy. For example, in methods for TSP, 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 [28] employs two differentiable processes, completion and correction, to solve any violations of equality or inequality constraints, respectively. The more recent HardNet framework [29] uses a closed-form projection to map to feasible solutions under affine constraints and relies on a differentiable convex optimization solver (e.g., OptNet [30]) when general convex constraints are considered.

Neural job shop scheduling Various NCO methods have already been studied for JSSP with makespan objective, of 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 [31] 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 [32], 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 [33] and Demirkol [34] 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 [16], 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 [35] 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 [36] 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 [37], 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 C

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

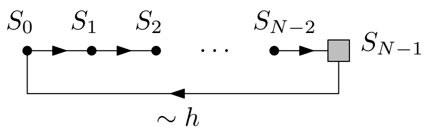


Figure C.1: Illustration of the induced Markov chain when dealing with finite episodes. The next state after the final state, indicated as the grey rectangle, is sampled according to initial state distribution h.

C.1 Stationary distribution for finite episodes.

Consider an absorbing Markov chain with transition matrix

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

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

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

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

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

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

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

Now we can derive two equivalent equations

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

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

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

and similarly, the second equation gives

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

C.2 Policy gradient estimation

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

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

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

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

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

```
1. sample $M$ episodes $\tau^{(1)},\dots,\tau^{(M)}$ 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 [38], 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 [38], the gradient of the value function for a fixed initial state s_0 with respect to the parameters is given by

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

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 [38]:

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

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

$$\nabla v_\pi \propto \mathbb{E}_{h,\pi}\left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t)\right] \tag{C.3a}$$
$$= \mathbb{E}_{h,\pi}\left[\sum_a \pi(a|S_t) q_\pi(S_t,a) \frac{\nabla \pi(a|S_t)}{\pi(a|S_t)}\right] \tag{C.3b}$$
$$= \mathbb{E}_{h,\pi}\left[q_\pi(S_t,A_t) \frac{\nabla \pi(A_t|S_t)}{\pi(A_t|S_t)}\right] \tag{C.3c}$$
$$= \mathbb{E}_{h,\pi}\left[G_t \nabla \log \pi(A_t|S_t)\right]. \tag{C.3d}$$

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

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

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

$$\nabla v_\pi \propto \sum_s \mu(s) \sum_a (q_\pi(s,a) + b(s)) \nabla \pi(a|s) \tag{C.5a}$$
$$= \mathbb{E}_{h,\pi}\left[\big(q_\pi(S_t,A_t) + b(S_t)\big) \nabla \log \pi(A_t|S_t)\right] \tag{C.5b}$$
$$= \mathbb{E}_{h,\pi}\left[\big(G_t + b(S_t)\big) \nabla \log \pi(A_t|S_t)\right]. \tag{C.5c}$$

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

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

then observe that

$$\begin{aligned} q_\pi(s,a) + b(s) &= \mathbb{E}_{h,\pi}\left[\sum_{k=t+1}^{\infty} R_k \middle| S_t = s, A_t = a\right] + \mathbb{E}_{h,\pi}\left[\sum_{k=1}^{t} R_k \middle| S_t = s\right] && \text{(C.7a)} \\ &= \mathbb{E}_{h,\pi}\left[\sum_{k=1}^{\infty} R_k \middle| S_t = s, A_t = a\right] && \text{(C.7b)} \\ &= \mathbb{E}_{h,\pi}[G_0 | S_t = s, A_t = a], && \text{(C.7c)} \end{aligned}$$

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

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

then applying the law of total expectation yields

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

Appendix D

Feasible configurations for single intersection model

We present a way to derive the feasible configurations of the two routes that intersect at some arbitrary angle, as shown in Figure 2.1. Assume that $\alpha < \pi/2$ is the acute angle between the two intersections. Furthermore, we consider uniform rectangular vehicle geometries with $L_i \equiv L$ and $W_i \equiv W$, but the analysis is easily extended to arbitrary dimensions. We skip a thorough derivation of the following expressions, but we note that it is based on the type of the distances illustrated in Figure D.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

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

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

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

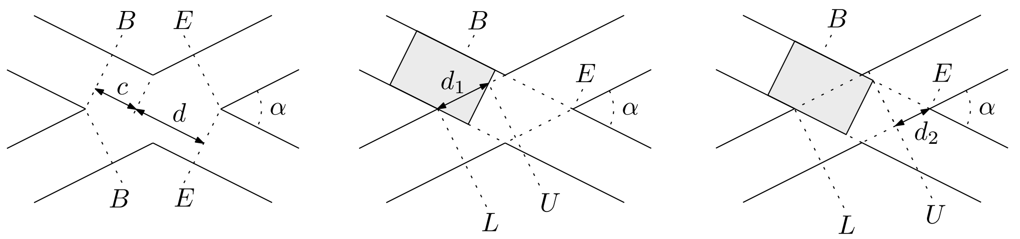


Figure D.1: Sketches to derive the feasible configurations of two vehicles in the intersecting routes model. Using some elementary trigonometry, the distances in the first figure can be shown to be $c = W/\tan(\alpha)$ and $d = W/\sin(\alpha)$. Furthermore, observe that we have $(x_i - B)/d_1 = \cos(\alpha)$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

$x_j - L > u(x_i)$. Hence, these two pairs of equations completely determine the set of feasible configurations, which can now be written as

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

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

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

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

such that the set of feasible configurations is simply given by

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

Appendix E

Miscellaneous

Lemma E.1. *Let $f : X \times Y \to \mathbb{R}$ be some continuous function. If Y is compact, then the function $g : X \to \mathbb{R}$, defined as $g(x) = \inf\{f(x, y) : y \in Y\}$, is also continuous.*

Lemma E.2. *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be continuous and $y \in \mathbb{R}^m$, then the level set $N := f^{-1}(\{y\})$ is a closed subset of $\mathbb{R}^n$.*

Proof. For any $y' \neq y$, there exists an open neighborhood $M(y')$ such that $y \notin M(y')$. The preimage $f^{-1}(M(y'))$ is open by continuity. Therefore, the complement $N^c = \{x : f(x) \neq y\} = \cup_{y' \neq y} f^{-1}(\{y'\}) = \cup_{y' \neq y} f^{-1}(M(y'))$ is open. □

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 4.2.3, we now generalize the exit boundary $\hat{x}$ to obtain the *acceleration boundary.* Because the derivation is completely analogous, we will only present the resulting expressions. Let $x \in \mathcal{D}[a, b]$ be some smooth trajectory, then the acceleration boundary $x^+[\xi]$ of x at some $\xi \in [a, b]$ is defined as the right-hand side of the inequality

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

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 (4.14), we define

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

such that $x^+[\xi](t) = x^+[x(\xi), \dot{x}(\xi), \xi](t)$ and similar to (4.15), we calculate

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

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