Machine Learning-Based Scheduling for the Coordination of Automated Vehicles

Jeroen van Riel

2025-08-27

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.

Crossing time scheduling. 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

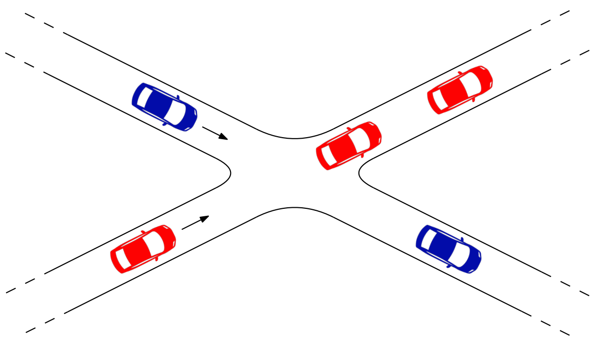


Figure 1.1: Stylized model of a single intersection with two conflicting flows that do not merge, so we assume vehicles must cross the intersection in a straight line, without turning.

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 Related work

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.

1.1.2 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. 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

1.3 Job shop scheduling

The job shop model provides a mathematical framework to study systems where a given set of—possibly distinct—facilities must be shared among a number of 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 [9, 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 1.1. 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 [10], see also [9, 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 1.2 shows an optimal schedule y^* with makespan $J(y^*) = 8$.

Disjunctive graph. A commonly used representation of job shop problems is through their disjunctive graph, which is a directed graph with vertices $\mathcal{N}$ corresponding to the operations and two types of arcs. The conjunctive arcs $\mathcal{C}$ are used to encode the predetermined machine sequence of each job. Each such arc $(i, j) \to (k, j)$ encodes that job j should first be processed 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 [9]. 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 1.1 are shown in Figure 1.3.

[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.

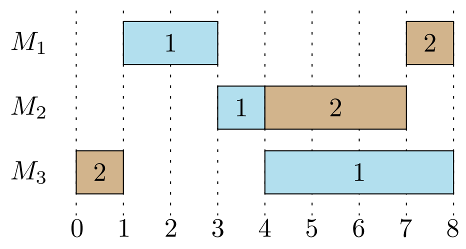


Figure 1.2: Example of an optimal schedule for Example 1.1, shown as a Gantt chart. Each row M_i corresponds to machine i and each block numbered j on this row represents the operation (i, j). The dashed lines indicate unit time steps. Note that machine 2 is kept idle, while operation $(2, 2)$ could have already been scheduled at time 1. Furthermore, for this particular instance, it can be checked that this is the unique optimal schedule.

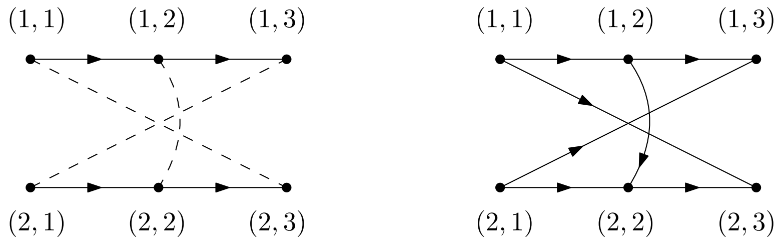


Figure 1.3: Illustration of disjunctive graphs for Example 1.1. Horizontal arrows represent conjunctive arcs. We used dashed lines to for the pairs of disjunctive arcs as dashed lines. The left graph corresponds to an empty selection $\mathcal{O} = \varnothing$ while the right graph shows the selection $\mathcal{O}$ that corresponds to the optimal schedule of Figure 1.2.

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 1.1 is known to be NP-hard [10], 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 1.1. Using the notation of the disjunctive graph, the problem can be concisely stated as

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

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.

1.4 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 [11] 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.

1.5 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 [12].

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 [13]. 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 [14] 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 [15]. 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 [16]. 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 [17] employs two differentiable processes, completion and correction, to solve any violations of equality or inequality constraints, respectively. The more recent HardNet framework [18] uses a closed-form projection to map to feasible solutions under affine constraints and relies on a differentiable convex optimization solver (e.g., OptNet [19]) when general convex constraints are considered.

1.5.1 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 [20] 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 [21], 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 [22] and Demirkol [23] 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 [9], 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.

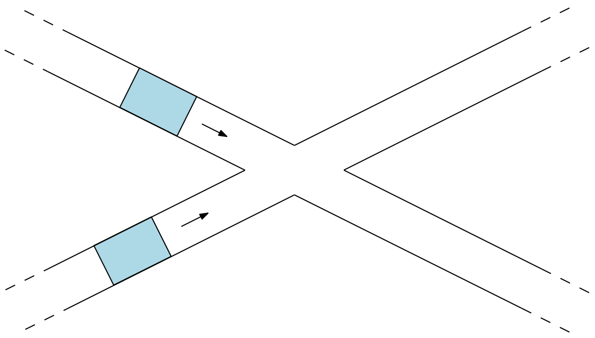
The seminal "Learning to Dispatch" (L2D) paper [24] proposes a construction heuristic for JSSP with makespan objective. Their method is based on a dispatching policy that is parameterized in terms of a graph neural network encoding of the disjunctive graph belonging to a partial solution. Again, each 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 consecutive 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 [25] 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 [26], 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.

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

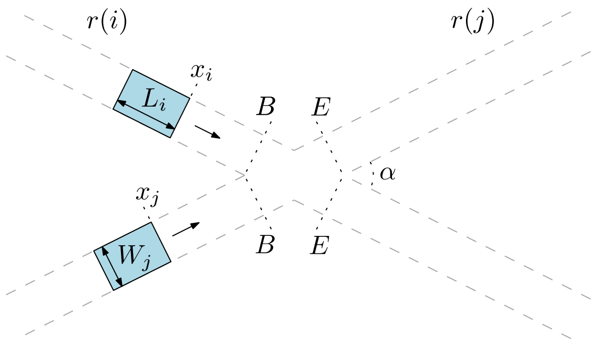


Figure 2.1: Example of two vehicle routes intersecting at some angle α. The indicated positions $x_i = B$ and $x_i = E$ for each vehicle i are chosen such that whenever the vehicle position x_i satisfies either $x_i \leq B$ or $x_i - L_i \geq E$, then the intersection area is not occupied by this vehicle 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 mention the possible role of local search heuristics in refining solutions.

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.*

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

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

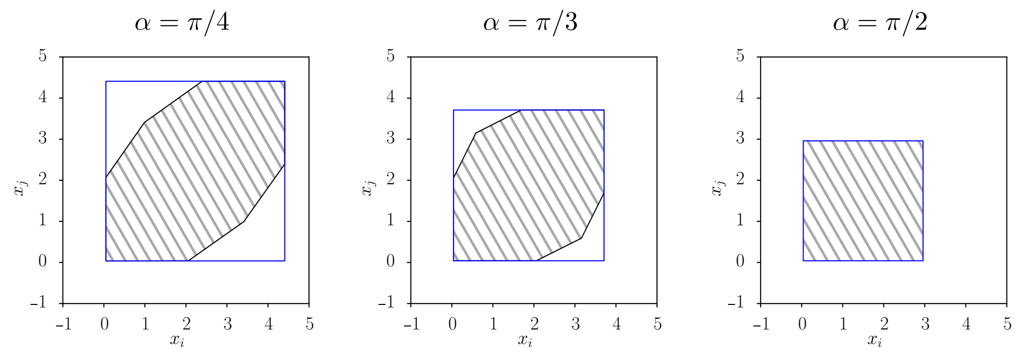


Figure 2.2: For three different intersection angles and fixed vehicle dimenions $W_i = W_j = 1$ and $L_i = L_j = 2$, we plotted the region $\mathcal{X}_{ij}^C$ in configuration space corresponding to collisions as the area marked in grey. The blue square regions correspond to the approximation of the 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 A.

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 consecutive 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

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

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). The acceleration is often considered to be the main input under control, so 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* model in optimal control literature.

We now present some different ways of measuring how desirable some individual vehicle trajectory $x_i(t)$ is, which we will do in terms of defining a cost functional $J(x_i)$ that we aim to minimize. First of all, we could use the absolute value of the acceleration as a proxy for energy consumption, by setting

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

As another example involving energy, consider the cost functional

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

where $v_d > 0$ is some reference velocity. This objective can be roughly interpreted as trying to maintain a velocity close to v_d, while simultaneously minimizing the square of acceleration as proxy of energy consumption. Alternatively, if we do not want to take into account energy consumption, but instead want to promote throughput of the overall system, a natural choice is to consider the negative cumulative distance

$$J(x_i) = -\int_0^{t_f} x_i(t) \, \mathrm{d}t. \tag{2.9}$$

We will also call the minimization of this cost functional the *haste objective*, which can be interpreted as trying to have each vehicle reach the intersection as fast as possible.

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

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

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. The next section will discuss ways of dealing with this non-convexity.

Remark 2.2. *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*. A more canonical way of writing such problems assumes $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ satisfy the dynamics*

$$\dot{x} = f(t, x, u), \quad x(t_0) = x_0, \quad u \in U \subset \mathbb{R}^m, \tag{2.10}$$

for some initial state x_0 and time t_0. Given some final time t_f, the goal is to minimize some running cost L and some terminal cost K in the following way:

$$\begin{aligned} \min_{x} \quad & \int_{t_0}^{t_f} L(t, x(t), u(t))\,\mathrm{d}t + K(t_f, x_f), && \text{(2.11a)} \\ \text{such that} \quad & \text{(2.10) holds}, && \text{(2.11b)} \\ & h(x) = 0, && \text{(2.11c)} \\ & g(x) \leq 0, && \text{(2.11d)} \end{aligned}$$

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 (2.11)*, most notably the necessary conditions given by the maximum principle of Pontryagin. However, the occurence of state constraints* (2.11c) *and* (2.11d) *make such analysis notoriously difficult and remains an area of ongoing research. For our current problem, note that our collision-avoidance constraints* (T.2) *and* (T.3)*, as well as the implicit speed constraint* (2.6a) *in* (T.1)*, are of the type* (2.11d)*.*

Feasibility. Note that 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

$$x_i^0 - x_j^0 > L \quad \text{for all} \quad (i, j) \in \mathcal{C}. \tag{2.12}$$

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

$$x_{(r,1)}^0 < B - 1/(2\omega) - 1/(2\bar{\omega}), \tag{2.13}$$

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 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

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

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 our current problem 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

$$\mathbb{T} := \{0, \Delta t, \dots, K\Delta t\}. \tag{2.14}$$

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

$$x_i(t + \Delta t) = x_i(t) + v_i(t)\Delta t, \tag{2.15a}$$

$$v_i(t + \Delta t) = v_i(t) + u_i(t)\Delta t, \tag{2.15b}$$

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 consecutive vehicles $(i, j) \in \mathcal{C}$ on the same route, the safe headway constraints can simply be added as

$$x_i(t) - x_j(t) \geq L \quad \text{for each } t \in \mathbb{T}. \tag{2.16}$$

Encoding of the safe crossing constraints needs some additional attention. 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:

$$x_i(t) \leq B + \delta_i(t)M, \tag{2.17a}$$

$$x_i(t) \geq E + L - \gamma_i(t)M, \tag{2.17b}$$

$$\delta_i(t) + \delta_j(t) + \gamma_i(t) + \gamma_j(t) \leq 3, \tag{2.17c}$$

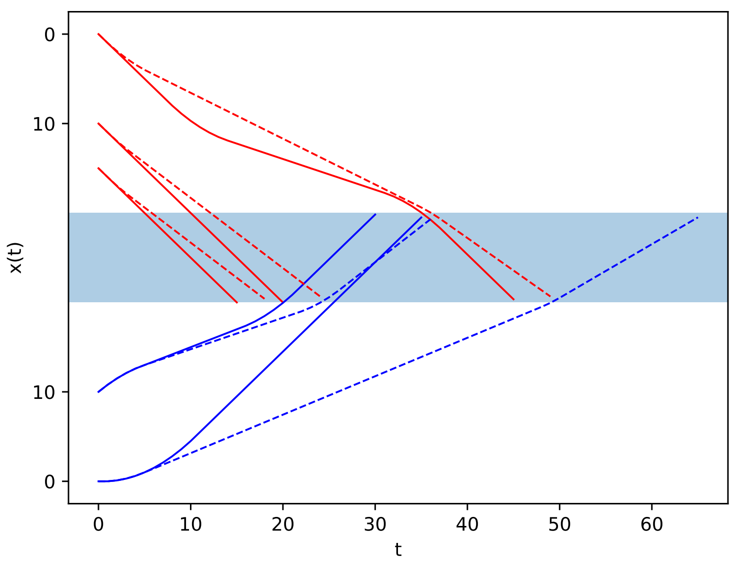
where M is some sufficiently large number. The idea behind this encoding is as follows. First, observe that setting $\delta_i(t)$ can be thought of as *deactivating* (2.17a), since M is chosen sufficiently large such that the inequality is trivially true. Analogously, setting $\gamma_i(t) = 1$ deactivates (2.17b). Hence, the constraint (2.17c) 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:

$$x_i(t) \leq B, \quad x_j(t) \leq B, \quad x_i(t) \geq E + L, \quad x_j(t) \geq E + L, \tag{2.18}$$

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 the two energy-based criteria for a small problem with five vehicles, for which the optimal trajectories are shown in Figure 2.3.

Remark 2.3. *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 [27] for a light introduction of trajectory optimization and [28] 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

Figure 2.3: Example of optimal trajectories for two objectives, obtained using the direct transcription method with parameters $L = 5$, $\mathcal{E} = [20, 30]$, $\bar{v} = 1$, $\omega = \bar{\omega} = 0.1$, $v_d = 0.6$, $T = 50$, $\Delta t = 1$ and initial conditions as given in the table. 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 stop drawing a trajectory after it leaves the intersection.

Bilevel decomposition. After applying the transcription method above, the safe crossing constraints of each conflict are imposed for every time step, requiring in total $2T$ 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, deciding which of the two vehicle crosses the intersection first. However, note that such an 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 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. The following presentation and notation is adapted from [5]. Throughout the following discussion, we assume the system parameters $(L, B, E, \bar{v}, \omega, \bar{\omega}, t_f)$ to be fixed.

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

$$\tau(x_i) := \min\{t \in [0, t_f] : x_i(t) = B\}, \tag{2.19a}$$
$$\xi(x_i) := \min\{t \in [0, t_f] : x_i(t) = E + L\}. \tag{2.19b}$$

Note that the sets in the definition are closed, because x_i is continuous, 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 (2.17). Similarly, observe that the collision-avoidance constraints

$$(x_i(t), x_j(t)) \notin \mathcal{E}^2 \quad \text{for all } \{i, j\} \in \mathcal{D} \tag{2.20}$$

are completely equivalent to the constraints

$$[\tau(x_i), \xi(x_i)] \cap [\tau(x_j), \xi(x_j)] = \varnothing \quad \text{for all } \{i, j\} \in \mathcal{D}. \tag{2.21}$$

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 (2.21), 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. Let $\mathcal{N}_r \subset \mathcal{N}$ be all vehicle indices of 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 *route planning 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 (2.21), 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 *occupancy scheduling problem*

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

Without further assumptions, problem (U) is not necessarily equivalent to the original problem (T). A major issue lies in the fact that some feasible solution x of (T) does not need to cross the intersection at all, i.e., it is not guaranteed that $\tau(x)$ and $\xi(x)$ are finite for x. To illustrate such situations, consider the following (pathological) example.

Example 2.1. Suppose we have a single vehicle on each route, both having initial speed $v_i^0 = \bar{v}$ and some initial distance x_i^0 satisfying the assumption (2.12), such that each vehicle can perform a full deceleration ($\ddot{x}_i = \omega$) and come to a stop somewhere before the start of the intersection, so that $x_i(1/\omega) < B$. Consider the cost functional

$$J(x_i) = \int_0^{t_f} x_i(t) \, \mathrm{d}t, \tag{2.22}$$

then it is easily seen that the optimal solution is to have both vehicles decelerate immediately as just described. In contrast, any solution x' obtained by solving (U) will cross the intersection sometime and thus has a strictly larger objective value.

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

$$\dot{x}_i(t) \geq \epsilon \quad \text{for all } t \in [0, t_f], \tag{2.23}$$

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

$$J(x_i) = \int_0^{t_f} \Lambda(x_i(\tau), \dot{x}_i(\tau), \ddot{x}_i(\tau))\, \mathrm{d}\tau, \tag{2.24}$$

for some convex and quadratic function $\Lambda(x, v, u)$, it has been argued that (T) and (U) are equivalent [29, Theorem 1]. Their argument relies on showing 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 (2.23), we will just explicitly restrict the set feasible trajectories to cross the intersection by focusing on the problem variant

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

Another issue that needs attention is the possibility of (T) not having an optimal solutions, even if feasible (see [30, Section 4.5]).

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

Proof. Every (y, z) has one or more solutions x. Every x has a single unique (y, z). □

Remark 2.4. *The constraints* (L.3)*,* (L.4)*,* (T*.4) *and* (T*.5) *are state constraints of a different type than those discussed in Remark 2.2. 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 prespecified intermediate time $y_i \in [t_0, t_f]$ and may be interpreted as some kind of "checkpoints". This situation 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 [31]. 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

$$\{(y, z) : y \in [T_i^l, T_i^h],\ l_i(y) \leq z \leq u_i(y)\} \subseteq \mathcal{T}(s_i), \tag{2.25}$$

[2]The authors of [5] refer to this as "strongly output monotone".

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 on the combinatorial part 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, because we avoid the need to approximate F, because we simply have

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

which means that the only remaining difficulty is how to deal with feasibility of the lower-level problem. Moreover, we assume that all vehicles start at full speed $v_i^0 = \bar{v}$ and introduce the additional constraint 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. First of all, because every x_i in a set of feasible trajectories for (L**) satisfies $x_i(y_i) = \bar{v}$, observe that we can let each vehicle continue at full speed across the intersection, which 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, this means we only have to care about finding

$$\mathcal{T}_y(s_r) := \{y_r : (y_r, y_r + \sigma) \in \mathcal{T}(s_r)\}, \tag{2.26}$$

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 using the mixed-integer linear programming framework after encoding the *disjunctive constraints* (C.3) using the big-M technique. This methodology will be explored in the next section.

2.3 Crossing time scheduling

The conclusion of the previous section is that, under certain assumptions, the trajectory optimization problem (T) reduces to a scheduling problem (C) of finding a crossing time schedule y that determines the time of crossing for each vehicle in the sytem. 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, we will focus on solving the crossing time scheduling problem. First, we show how to leverage the branch-and-cut framework by formulating (C) as a Mixed-Integer Linear Program (MILP) and defining three types of cutting planes. We investigate the computational complexity of this approach for different classes of problems in Section 2.3.2. As an illustration of an alternative solution approach, Section 2.3.3 briefly discusses a local search heuristic. The next chapter will discuss methods to learn heuristics using a data-driven methodology.

2.3.1 Branch-and-cut

To formulate the crossing time scheduling problem as a MILP, we rewrite the disjunctive constraints using the well-known big-M method by introducing a binary decision variable γ_{ij} for every disjunctive pair $\{i,j\} \in \mathcal{D}$. To avoid redundant variables, we first impose some arbitrary ordering of the disjunctive pairs by defining

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

such that for every $(i,j) \in \bar{\mathcal{D}}$, setting $\gamma_{ij} = 0$ corresponds to choosing disjunction $i \to j$ and $\gamma_{ij} = 1$ corresponds to $j \to i$. This yields the following MILP formulation

$$\begin{aligned}
\min_{y} \quad & \sum_{i \in \mathcal{N}} y_i - a_i & \\
\text{s.t.} \quad & a_i \leq y_i, & \text{for all } i \in \mathcal{N}, \\
& y_i + \rho \leq y_j, & \text{for all } (i,j) \in \mathcal{C}, \\
& y_i + \sigma \leq y_j + \gamma_{ij} M, & \text{for all } (i,j) \in \bar{\mathcal{D}}, \\
& y_j + \sigma \leq y_i + (1 - \gamma_{ij}) M, & \text{for all } (i,j) \in \bar{\mathcal{D}}, \\
& \gamma_{ij} \in \{0,1\}, & \text{for all } (i,j) \in \bar{\mathcal{D}},
\end{aligned}$$

where $M > 0$ is some sufficiently large number. This problem can be solved by any off-the-shelf MILP solving software. Next, we will discuss three types of cutting planes that can be added to this formulation, which we hope improves the solving time.

Cutting planes. Consider some disjunctive arc $(i,j) \in \bar{\mathcal{D}}$. Let $\mathit{pred}(i)$ denote the set of indices of vehicles that arrive no later than i on route $r(i)$. Alternatively, we could say these are all the vehicles from which there is a path of conjunctive arcs to i. Similarly, let $\mathit{succ}(j)$ denote the set of indices of vehicles that arrive no later than j on route $r(j)$. Now suppose $\gamma_{ij} = 0$, so the direction of the arc is $i \to j$, then any feasible solution must also satisfy

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

Written in terms of the disjunctive variables, this gives the following cutting planes

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

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

Apart from the redundancy that stems from the way the conflicts are encoded, we will now study some less obvious structure in the problem. Let us briefly consider some trivial examples to establish some better intuition for the problem.

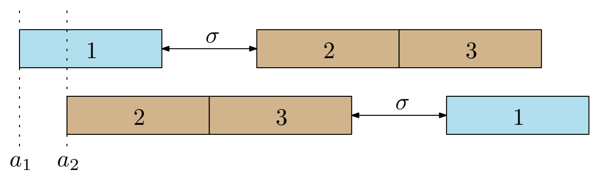


Figure 2.4: Illustration of the two possible sequences of vehicles in Example 2.3.

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

$$a_2 \leq \sigma/3, \tag{2.28}$$

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

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

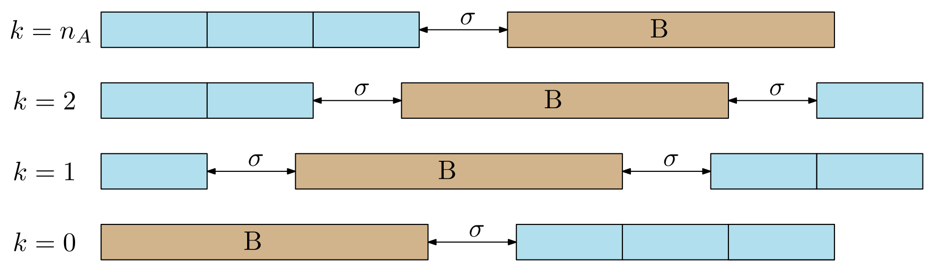


Figure 2.5: Illustration of different ways to split platoon A, regarding Example 2.4, assuming that $a_A = a_B$ with $n_A = 3$ and some arbitrary n_B.

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

$$\sum_{i \in \mathcal{N}} y_i(n_A) = \sigma n_B + \sum_{j=1}^{n_A+n_B} (j-1)\rho, \tag{2.29}$$

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 (2.29), 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 (2.28) that we derived for the case with $n_A = 1$ and $n_B = 2$.

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 [32])**.** *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 $\delta_{ij} = 1$, 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 analysis of branch-and-cut

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 consecutive 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 $A_0 = 0$.

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

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

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.

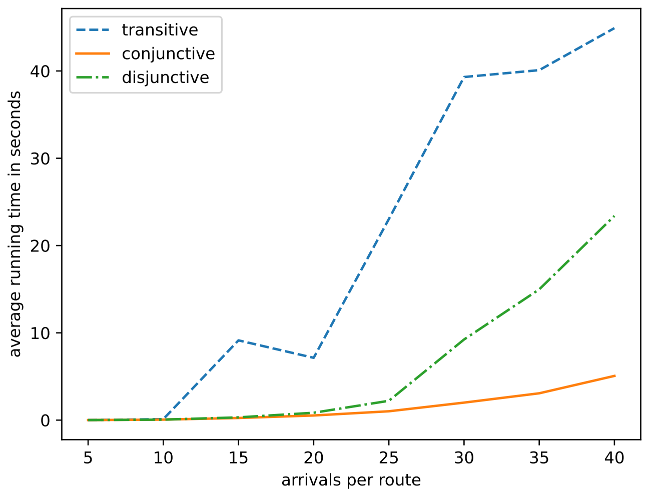


Figure 2.6: The average running time of the branch-and-cut procedure is plotted as a function of the number of arriving vehicles per route, for each of the three indicated cutting planes. Each average is computed over 20 problem instances. All instances use $\rho = 4$ and $\sigma = 1$. The arrivals of 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.

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, we can try to use some kind of local search. Specifically, compute a solution using one of the methods from the previous section and then explore some *neighboring solutions*, that we define next.

As seen in the previous sections, vehicles of the same route occur mostly in platoons. For example, consider for example the route order $\eta = (0, 1, 1, 0, 0, 1, 1, 1, 0, 0)$. This example has 5 platoons of consecutive vehicles from the same route. The second platoon consists of two vehicles from route 1. The basic idea is to make little changes in these platoons by moving vehicles at the start and end of a platoon to the previous and next platoon of the same route. More precisely, we define the following two types of modifications to a route order. A *right-shift* modification of platoon i moves the last vehicle of this platoon to the next platoon of this route. Similarly, a *left-shift* modification of platoon i moves the first vehicle of this platoon to the previous platoon of this route. We construct the neighborhood of a solution by performing every possible right-shift and left-shift with respect to every platoon in the route order. For illustration purposes, we have listed a full neighborhood for some example route order in Table 2.1.

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)	

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 [33, Chapter 3].

Chapter 3

Optimal schedule modeling

Methods that are based on the branch-and-cut framework are guaranteed to find an optimal solution, but as we illustrated in Section 2.3.2, their 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 be considering heuristics that can be formulated as autoregressive models [34] and show how these can be tuned by using example problem instances.

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?]

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

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.

Autoregressive modeling of 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

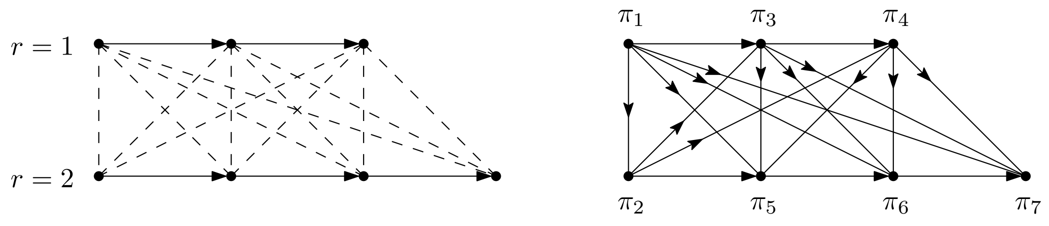


Figure 3.1: Illustration of disjunctive graphs for some instance with two routes with $n_1 = 3$ and $n_2 = 4$ vehicles. Horizontal arrows are the conjunctive arcs, the rest are disjunctive arcs. The left graph is and *empty* disjunctive graph, because $\mathcal{O} = \varnothing$. The selection of disjunctions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

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 [35].

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

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

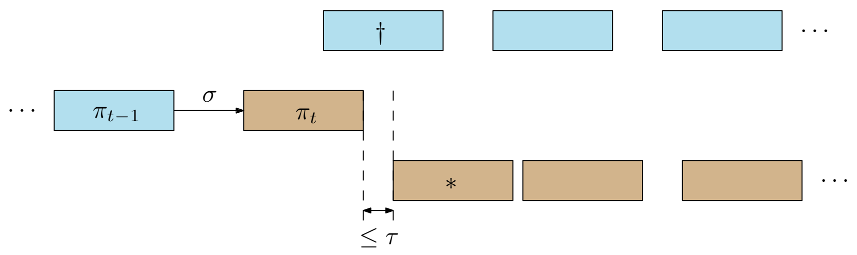


Figure 3.2: Illustration of how the threshold heuristic is evaluated at some intermediate step t to choose the next route η_{t+1}. The top and bottom row contains the unscheduled vehicles from route 1 and route 2, respectively, drawn at their earliest crossing times $\beta_t(i)$. The middel row represents 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 $\beta(\pi^r) = (\beta(\pi^r_1), \beta(\pi^r_2), \dots)$. Let the minimum crossing time lower bound among unscheduled vehicles be denoted by T, then we call $h_r = \beta(\pi^r) - T = (\beta(\pi^r_1) - T, \beta(\pi^r) - T, \dots)$ the *horizon* of route r.

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

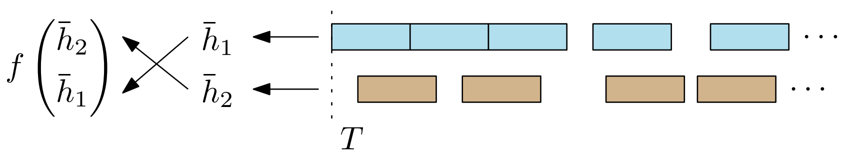


Figure 3.3: Schematic overview of parameterization of the neural heuristic. The distribution over the next route is parameterized as $p_\theta(\eta_{t+1}|G_t)$, which is computed from the current route horizons h_r via embeddings $\bar{h}_r$ and the cyling trick. Observe that the particular cycling shown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

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 MILP from Section 2.3. For each such optimal schedule, we can compute the sequence $G_0, \eta_1, G_1, \eta_2, \dots, \eta_N, G_N$. The resulting set of pairs $\{(G_t, \eta_{t+1}) : t = 1, \dots, N-1\}$ 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 autoregressive 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.

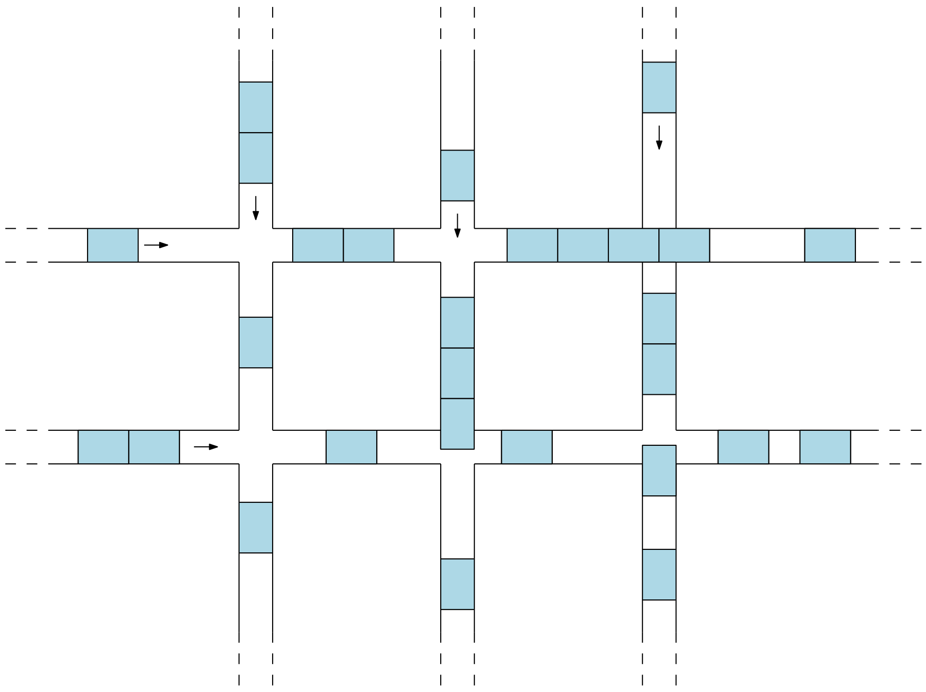
n	type	MILP	time	exhaustive (gap)	threshold (gap)	τ_{opt}	time
10	low	5.29	0.05	9.49 (79.45%)	7.78 (47.08%)	0.95	3.79
30	low	8.60	2.01	12.72 (47.88%)	11.31 (31.49%)	2.65	21.04
50	low	11.03	13.78	16.56 (50.20%)	14.60 (32.41%)	1.95	51.09
10	med	4.46	0.06	7.20 (61.32%)	6.39 (43.15%)	2.50	3.79
30	med	6.99	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

Network with 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.

Chapter 5

Learning to schedule

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

[10] 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.

[11] 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.

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

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

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

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

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

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

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

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

[20] 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.

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

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

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

[24] 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.

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

[26] 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.

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

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

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

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

[31] 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.

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

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

[34] Jakub M. Tomczak. *Deep Generative Modeling*. Springer International Publishing, Cham, 2024.

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

Appendix

Appendix A

Feasible configurations for single intersection model

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

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

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

With these definitions, in order for the intersection to be free for vehicle j, position x_i must satisfy either $x_i < l(x_j)$ or $x_i - L > u(x_j)$ and x_j must satisfy either $x_j < l(x_i)$ or

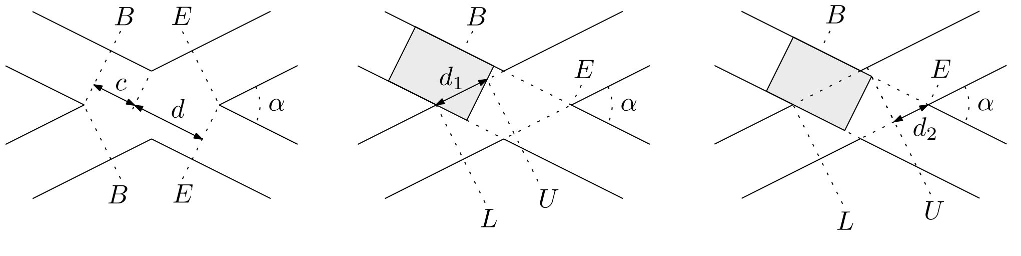


Figure A.1: Sketches to derive the feasible configurations of two vehicles in the intersecting routes model. Using some elementary trigonometry, the distances in the first figure can be shown to be $c = W/\tan(\alpha)$ and $d = W/\sin(\alpha)$. Furthermore, observe that we have $(x_i - B)/d_1 = \cos(\alpha)$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

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

$$\mathcal{X}_{ij} = \{(x_i, x_j) \in \mathbb{R}^j : [\, x_i - L, x_i \,] \cap [\, l(x_j), u(x_j)] = \varnothing \tag{A.3}$$
$$\text{and } \; [\, x_j - L, x_j] \cap [\, l(x_i), u(x_i) \,] = \varnothing\}. \tag{A.4}$$

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

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

such that the set of feasible configurations is simply given by

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

Appendix B

REINFORCE with baseline

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

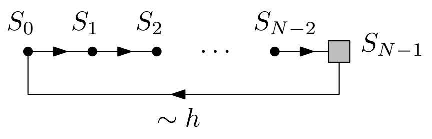


Figure B.1: Illustration of the induced Markov chain when dealing with finite episodes. The next state after the final state, indicated as the grey rectangle, is sampled according to initial state distribution h.

B.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

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

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

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

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

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

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

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

B.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 [11], 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 [11], the gradient of the value function for a fixed initial state s_0 with respect to the parameters is given by

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

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 [11]:

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

where the constant of proportionality is just the average episode length. Because we do not know μ or q_π explicitly, we would like to estimate ∇v_π based on samples. If we sample episodes according to h and π as explained above, we encounter states according to μ, so we have

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

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

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

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

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

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

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

then observe that

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

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

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

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

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

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

then applying the law of total expectation yields

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