# Neural Combinatorial Optimization for Scalable Coordination of Autonomous Vehicles in Networks

**Jeroen van Riel**
Eindhoven University of Technology
j.a.a.v.riel@student.tue.nl

December 17, 2024

## Abstract

Coordination of autonomous vehicles through networks of intersections has the potential of reducing risk of accidents while providing huge savings in economic costs. We adopt an optimal control formulation with hard collision-avoidance constraints as a model to study the fundamental complexity of centralized coordination schemes for autonomous traffic without traffic signals. With delay as the main performance criterion, we show that our problem decomposes into an upper-level problem that is a variant of the classical job-shop scheduling problem and a lower-level trajectory optimization problem that can be efficiently solved by linear programming. Previous works have successfully proposed reinforcement learning methods to train scheduling policies based on the well-known disjunctive graph formalism of job-shop problems. We propose to adapt this formalism to our job-shop variant and, like previous works, to use a graph neural network to parameterize a scheduling policy that is tuned using reinforcement learning in order to obtain a scalable approximation scheme for the high-dimensional coordination problem.

## 1 Overall aim and goals

### 1.1 Motivation and challenges

Given the ongoing development of self-driving vehicle technology, it is not difficult to imagine a future vehicular mobility system without human drivers. Besides the obvious advantage of relieving us from the task of driving, freeing time that can be spent on more meaningful activities, other benefits include increased safety and efficiency. Human error is one of the major causes of most accidents in vehicular traffic, which motivates the design of automated systems with guarantees on collision avoidance, with the potential of saving millions of injuries and lives. Furthermore, traffic coordination methods promise to reduce economic costs such as travel delay, energy consumption and pollution [1], by optimizing the shared use of resources like parking lots and road intersections. In general, coordination of autonomous vehicles has been studied at different levels [2]. Local coordination like platooning of vehicles may lower energy consumption by reducing aerodynamic resistance and may result in more efficient use of intersections. On a larger scale, methods like network-wide dynamic route optimization have been proposed to reduce travel delay for all users in the traffic network.

We choose to focus on coordination algorithms for optimizing the shared use of a network of intersections without traffic lights. We consider a model in which every vehicle in the network is controlled by a single *controller* and we assume that every vehicle follows a fixed predetermined route. In this context, we identify the following three key challenges that are not yet fully addressed in the current literature.

- **Safety.** It is nontrivial to design systems that are guaranteed to be safe with respect to collisions, because the potentially complex vehicle dynamics must be taken into account. Because safety is so crucial in real-world application, algorithms must guarantee this by design.
- **Scalability.** Most current works consider coordination at a single intersection [3], so an important question remains how to develop systems for networks of intersections. Coordination algorithms should naturally scale

to complex networks of intersections with large numbers of vehicles, while balancing the quality of solutions with computational complexity, motivating the study of good approximation schemes.

- **Learning.** Given the successful applications of deep reinforcement learning to highly complex control tasks, it is natural to ask how coordination of autonomous vehicles can be formulated such that a similar methodology can be applied for automatically learning good policies for this complex decision-making problem. It is nontrivial to apply current reinforcement learning formulations to this setting, because satisfaction of hard constraints, like collision avoidance, is still an area of active research [4].

We observe that multi-intersection access control in a network of intersections shows some resemblance to the classical job-shop scheduling problem, which is a canonical example of a resource-sharing problem. Recently, deep reinforcement learning methods have been successfully proposed for this problem [5, 6, 7], often based on efficient representations of problem instances using graph neural networks. This motivates us to investigate whether a similar approach can be applied in our setting by adapting the job-shop scheduling formulation to the traffic coordination problem.

### 1.2 Overview of related literature

Traffic lights are currently one of the most effective way of influencing traffic, so current coordination methods often rely on some sort of synchronization of traffic light settings at neighboring intersections [8, 9], with *green waves* on arterial roads being a well-known example. Since the complex interactions in signalized networks are difficult to model explicitly, recent years have seen an increased interest in the application of deep learning models to traffic-related problems. In particular, numerous works have successfully proposed deep reinforcement learning methods for traffic signal control that scale to large networks of intersections [10, 11]. Furthermore, given the increasing number of fully autonomous vehicles, new types of traffic coordination scheme become conceivable [2], with notable examples being services like smart parking and ride sharing, local coordination methods like ramp merging and platooning, or network-wide traffic flow optimization through smart route suggestions.

Intersection access management without traffic signals is one type of setting in which coordination of autonomous vehicles can potentially bring huge benefits, both in terms of reduced risk and economic costs. While early works were mostly based on reservation-based protocols [12], current coordination schemes are studied under very different model assumptions regarding vehicle dynamics and modes of communication and control [3]. An important distinction can be made based on the used communication models, which can be roughly categorized into distributed approaches, where groups of vehicles coordinate trajectories together, and centralized approaches, where individual vehicles receive instructions from a central controller.

Centralized solutions for access control at a single intersection have be studied by framing the problem in terms of optimal control, providing a sound theoretical foundation. Most works from this perspective use a simple one-dimensional vehicle model known as the double integrator in optimal control literature. Although solutions can be obtained using so-called *direct transcription* methods, the high-dimensionality of the problem calls for good approximation schemes. A common observation is that the optimization problem may be thought of as two coupled optimization problems [13, 14, 15], where the upper-level problem is to determine when and in which order vehicles enter and exit each intersection on their route. The lower-level problem is to find optimal trajectories that match these time slots.

Extensions to multi-intersection settings have received much less attention. Existing methods are extensions of reservation-based protocols [16] or are based on mixed-integer linear programming [17]. We recognize that such a multi-intersection extensions is in large part of a combinatorial nature. Recently, there has been an increasing interest in applying a machine learning perspective on many classical combinatorial problems [18], to which we refer as *neural combinatorial optimization*. Realistic instances of many combinatorial problems are often too complex to solve to optimality, so researchers try to propose good approximation schemes and heuristics, based on some kind of structure in the problems of interest. However, manually designing such heuristics is a tedious task, requiring a lot of experience and deep insight into the problem. Tailored deep reinforcement learning methods have shown to be able to automatically derive good heuristics from scratch for many classical combinatorial optimization problems [19].

### 1.3 Problem formulation and research questions

We study a centralized model for the network-wide coordination of autonomous vehicles where vehicles are directly controlled, without relying on traffic lights. Specifically, we model coordination as an optimal control problem with hard constraints on collision avoidance and with assumptions of perfect communication and control. In this framework, a central controller sets the acceleration of all individual vehicles, with each vehicle modeled as a double integrator. Once a vehicle enters the network, it follows a predetermined route until it exists, as illustrated in Figure 1. The goal is

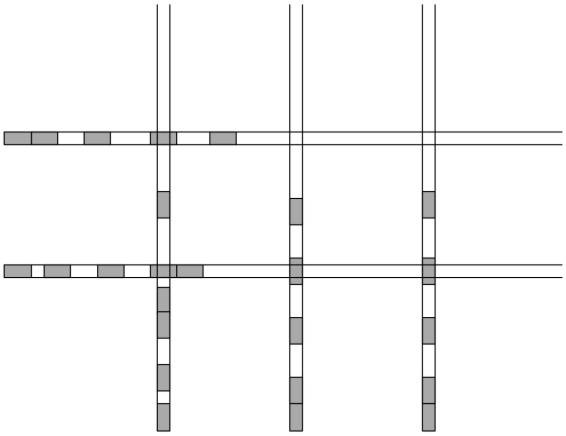

Figure 1: Illustration of some simple grid-like network of intersections with vehicles drawn as grey rectangles. There are five vehicle routes that only intersect at intersections (so no two routes share the same lane between intersections): two from east to west and three from south to north. Turning at intersections is not allowed in this model.

to control the trajectory of each vehicle in the network, ensuring collision-free movement while optimizing a global measure of efficiency. One common measure of efficiency in the literature is the total delay experienced by all vehicles. However, it may also be desirable to penalize acceleration, which can serve as a proxy for energy consumption. In scenarios where all vehicle arrival times are known in advance, we assume that the trajectories can be computed offline, without any prior interaction with the system. This setup is referred to as *offline trajectory optimization*.

In general, the offline trajectory optimization problem is of infinite dimension, because vehicle trajectories are smooth functions of time. Therefore, every numerical algorithm is based on some kind of dimensionality reduction. However, we recognize some basic structure in the trajectories that allows us to define a lower-dimensional representation. By assuming that the vehicle dynamics are known precisely, it is possible to formulate hard constraints on the trajectories of vehicles in order to avoid collisions. One of our key observation is that these constraints are somewhat similar to those found in classical job-shop scheduling problems, for which efficient solution methods are available that scale reasonably well to larger instances. Therefore, we aim to address the first two challenges of safety and scalability by answering our first research question:

**Q1**. How can we model the offline trajectory optimization problem, modeling coordination of autonomous vehicles in networks of unsignalized intersections, as a variant of job-shop scheduling to effectively reduce the dimensionality of the problem, while guaranteeing the generation of collision-free trajectories?

Apart from the similarity to job-shop scheduling, we believe that the offline trajectory optimization problem has more hidden structure that stems from the way vehicles interact with each other in large networks of intersections. Our second aim is to investigate how deep reinforcement learning can be applied to the offline trajectory optimization problem, because we believe that a machine learning perspective [18] can help discover structure in complex decision-making problems. Our hypothesis is that a lot of this structure is hard to formulate precisely, so the coordination algorithm should automatically learn how to exploit the structure in some class of problem instances. To further improve the scalability of our solutions method and to address the remaining challenge of learning, we formulate our second research question as:

**Q2**. Can recent deep reinforcement learning formulations for combinatorial optimization problems, like job-shop scheduling, be adapted to the offline trajectory optimization problem, formulated in terms of job-shop scheduling, in order to provide a scalable optimization algorithm that automatically learns to exploit hidden structures in problem instances?

## 2 Research approach

### 2.1 Overall methodology and decomposition

Because trajectories are smooth functions, the offline trajectory optimization problem is generally of infinite dimension. In order to solve it numerically, some dimensionality reduction scheme is required. A straightforward way to represent trajectories is based on a discrete time grid. Based on this, it is possible to formulate a mixed-integer linear program, whose optimal solutions approximate solutions to the original problem. Such so-called *direct transcription* methods are very common for optimal control problems, but they are still very high-dimensional due to the time discretization.

We show how the dimension can be reduced further, based on the following observation. A key issue the controller has to decide is the order of crossing at intersections, which is precisely what makes the optimal control problem non-convex and thus hard to solve with standard methods. When considering an optimization objective in terms of vehicle delay and some additional assumptions, the problem can be shown to decompose as a *bilevel* optimization problem with an *upper-level* combinatorial problem to determine a crossing time schedule — indicating when vehicles cross the intersections on their route — and a *lower-level* optimal control problem to find trajectories that match these crossing times, see Figure 2 for an illustration.

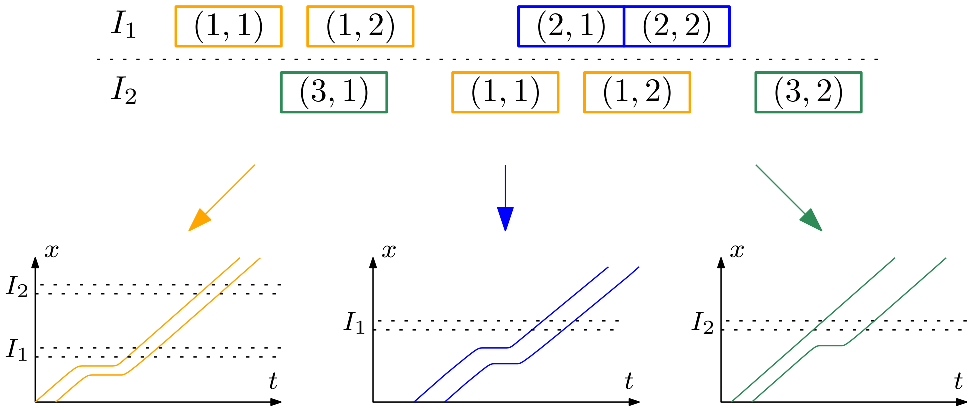

Figure 2: Illustration of the proposed bilevel decomposition. The upper-level problem is assign each vehicle a time slot for each of the intersections (here denoted $I_1$ and $I_2$) on its route. Based on this schedule, the lower-level problem is to generate trajectories satisfying these time slots.

Under some assumptions on the routes that vehicles take, it can be shown that the upper-level problem, to which we will refer as the Vehicle Scheduling Problem (VSP), is an extension of the classical Job-Shop Scheduling Problem (JSSP). Intersections are modeled as machines (the shared resources) and each vehicle corresponds to a job, whose operations model the crossing of intersections on the vehicle's route. In addition to the regular JSSP constraints, three types of additional constraints are defined to model some necessary clearance time at intersections and to take into account the travel time between intersections and safety constraints to prevent rear-end collisions. Given a solution to the VSP, computing the lower-level trajectories can be done reasonably efficiently, as we show in the next section. Furthermore, it can be shown that trajectories that satisfy the schedule are guaranteed to be collision-free, satisfying our first objective of guaranteed safety. Therefore, the bilevel decomposition enables us to focus on solving the scheduling problem.

Like plain JSSP problems, VSP can be solved by formulating it as a Mixed-Integer Linear Program (MILP) and using an off-the-shelf solver. Although being a powerful optimization framework, this approach is not likely to scale well, because of the inherent complexity of larger instances. However, by setting a time limit on the solving time, this method yields a good heuristic solution method.

To tackle the scalability issue, we propose to leverage the recent progress made in applying Deep Reinforcement Learning (DRL) methods to Combinatorial Optimization Problems (COP), of which JSSP is a classical example. We show that the disjunctive graph encoding of job-shop problems can be extended to our VSP variant. Following the approach of previous successful deep reinforcement learning methods for job-shop problems, we propose a policy for generating schedules in a step-by-step fashion. The policy is parameterized based on a Graph Neural Network (GNN) encoding of the adapted disjunctive graph. We verify whether the resulting policy space is general enough by using imiation learning on expert demonstration obtained from optimal solutions by backtracking the required step-by-step decisions. Finally, we train the policy using reinforcement learning in a step-by-step scheduling environment.

### 2.2 Models and methods

We now provide a more detailed introduction of the most important models and methods that we plan to use and discuss how they can be adapted to the current problem formulation. The first three sections are mainly about answering research question Q1 by defining the upper-level Vehicle Scheduling Problem. Section 2.2.4 shows how to solve this problem using deep reinforcement learning, thereby answering question Q2.

#### 2.2.1 Trajectory optimization

The bilevel decomposition sketched above depends heavily on the feasibility of the lower-level trajectory optimization problem. More precisely, we need to be sure that a crossing time schedule always allows trajectories that respect the vehicle dynamics and are collision-free. Therefore, we distinguish three key moments in time related to a particular crossing. For some combination of an intersection and a vehicle with this intersection on its route, let the *crossing time* $y$ be the first moment when the front bumper of the vehicle $i$ enters the intersection area, as depicted by the first situation in Figure 3. Next, let $\rho > 0$ denote the time after which another vehicle on the same route can start crossing and let $\sigma > \rho$ denote the of a *full crossing*, which is time after which any vehicle from a conflicting routes can start crossing the intersection. To simplify matters, we assume that vehicles drive at *full speed* when crossing the intersection, so for all $t$ between $y$ and $y + \sigma$.

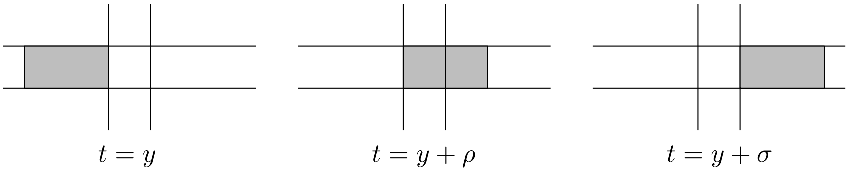

Figure 3: Three different snapshots of a vehicle crossing an intersection.

The upper-level scheduling problem provides the crossing times $y$, which the trajectories must satisfy. Together with the constraints from the double integrator vehicle dynamics, this yields a separate optimal control problem for each group of vehicles with the same route. Each of these can be straightforwardly solved by direction transcription to a linear program by introducing a discrete time grid to represent each vehicle's trajectory. The vehicle dynamics can be encoded using a forward Euler scheme or higher-order numerical integration methods. Rear-end collision-avoidance constraints can simply be added for each timestep. Finally, the position at timestep $y$ is bound to the start of the intersection.

#### 2.2.2 Job-shop scheduling

We briefly introduce the Job-Shop Scheduling Problem (JSSP), because we formulate our Vehicle Scheduling Problem (VSP) as a direct extension. For a textbook introduction, we recommend the very accessible book on scheduling by Pinedo [20]. The classical JSSP problem considers a set of $n$ jobs that must be assigned to non-overlapping time slots on a set of $m$ machines. Each job $i$ has a set of $n_i$ operations $O_{i1}, \dots, O_{in_i}$ that need to be executed in this order. Each operation $O_{ij}$ requires $p_{ij}$ processing time on machine $M_{ij}$. Each machine can process at most one operation and early preemption is not allowed. The task of the scheduler is to determine a valid schedule of start times $y_{ij}$ for each operation, while minimizing some objective function. Let $C_{ij} = y_{ij} + p_{ij}$ denote the *completion time* of operation $O_{ij}$. Common optimization objectives are a function of these completion times, e.g., minimizing the total completion time among operations or minimizing the maximum completion time, also known as the *makespan*. Objectives that are a non-decreasing function of completion times are called *regular*.

A commonly used representation of JSSP instances is the *disjunctive graph*, with vertices $\{O_{ik} : 1 \leq i \leq n, 1 \leq k \leq n_i\}$ corresponding to all the operations. The set of *conjunctive arcs* encodes all the precedence constraints $O_{i,k} \rightarrow O_{i,k+1}$ among each job's operations. The set of *disjunctive edges* consists of undirected edges between each pair of operations from distinct jobs that need to be processed on the same machine, effectively encoding all such *conflicts*. Each valid schedule induces an ordering of operations on machines that is encoded by fixing the direction of each disjunctive edge such that we obtain a directed acyclic graph.

We now explain how JSSP can be extended to VSP, where intersection are modeled as machines, vehicles correspond to jobs. The road network is represented as a simple directed graph with nodes representing intersections. Each vehicle has a fixed route consisting of a series of intersections. The act of *crossing* an intersection on this route is modeled as an operation. The processing time of a crossing corresponds to $\rho$ in Figure 3. We assume that routes are *edge-disjoint*, in the sense that they do not share edges but may possibly overlap at nodes (intersections). Furthermore, we assume that

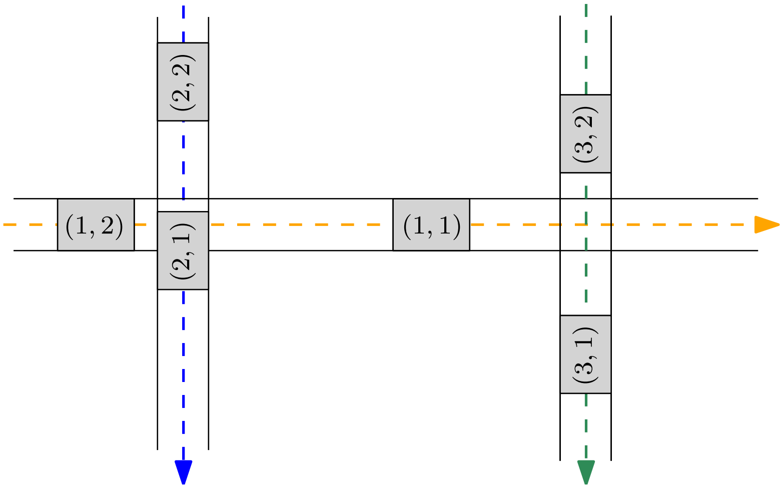

Figure 4: Example network with three different routes satisfying the edge-disjoint assumption (such that vehicle flows never merge) and vehicle indices $(r, k)$.

vehicles on the same lane are not able to overtake. Therefore, each vehicle can be identified as a tuple $(r, k)$, with some route identifier $r$ and an integer $k$ that indicates the relative order on this route, see Figure 4 for an example. In addition to the regular constraints of JSSP, we define the following three constraints:

- *Clearance time.* To guarantee collision-free crossing at intersections, some additional time is required between crossing times for vehicles that approach an intersection from different lanes. More precisely, we require at least $\sigma$ time (see Figure 3) between crossings of vehicles belonging to different routes.
- *Travel constraints.* Vehicles need to physically drive towards the next intersection on their route, so there is a lower bound on the travel time. Therefore, we need a constraint between the crossing times at every two consecutive intersections on each vehicle's route.
- *Buffer constraints.* Each lane between intersections has only space for a limited number of vehicles, depending on their position and velocity. To avoid rear-end collisions, the number of vehicles in each lane needs to be limited. This type of constraint is the most difficult to formulate, so the basic principle is illustrated by the example in Figure 5.

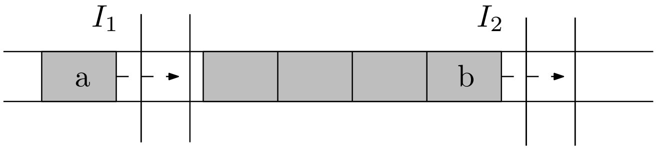

Figure 5: Illustration of buffer constraint for a pair of vehicles on the same route. Suppose that the vehicles on the lane between the two intersections are currently not moving and waiting to cross $I_2$. It is clear that the crossing time of vehicle $a$ at $I_1$ must be happen at least after vehicle $b$ starts crossing $I_2$. Since this particular lane segment has capacity for 4 waiting vehicles, we need such a buffer constraint between every pair of vehicles $(r, k_1)$ and $(r, k_2)$ that satisfy $k_2 = k_1 + 3$.

### 2.2.3 Mixed-integer linear programming

Like the original JSSP, our VSP can be formulated as a Mixed-Integer Linear Program (MILP). For a single isolated intersection, it is straightforward to do this. The crossing order decisions can be modeled by introducing a binary decision variable for each pair of conflicting vehicles that approach the intersection from different lanes. Note that the number of these so-called *disjunctive decisions* grows exponentially in the number of vehicles. Whenever two consecutive vehicles on the same lane are able to cross the intersection without a gap, it has been shown that they will always do so in any optimal schedule [21] (with respect to the total delay objective).

When considering a network of intersections, the two additional types of constraints are necessary to guarantee feasibility of the lower-level trajectory optimization. Both constraint types can be naturally encoded in the disjunctive graph. When we assume that there is no merging of routes, which means they only overlap at intersections, we expect that VSP is still computationally tractable for reasonably sized instances, using modern solvers. Whenever general routes are considered, a naive formulation would include a lot of disjunctive decisions, because vehicles can in principle conflict with all other vehicles that share a part of their route, even if it is clear that this would never happen in any sensible schedule.

During the solving procedure, the currently best solution is remembered. Therefore, MILP solving can be used as an approximation method by setting a limit on the computation time, which addresses our second overall objective. Another benefit of the MILP framework is the easy incorporation of problem-specific *cutting planes* in order to exploit additional structure in problem instances. For example, we observe that the above property on crossing without a gap in a single intersection can be used to formulate multiple types of cutting planes to improve the running time of the solver.

### 2.2.4 Constructive neural heuristic

We now explain how to model VSP as a sequential decision making process such that deep reinforcement learning can be applied to address our objective of obtaining a learning optimization algorithm. The fundamental challenge of the scheduler is to determine in which order vehicles are allowed to cross the intersections on their routes. Observe that the relative order among vehicles on the same route stays fixed, because they are not allowed to overtake. We propose to order vehicles in a step-by-step fashion. For each intersection, we keep track of a partial ordering of vehicles. Each step corresponds to choosing some combination of an intersection and a vehicle that has not yet been added to the partial ordering of this intersection and has this intersection on its route. We will refer to such intersection-vehicle pair as a *crossing*. Figure 6 illustrates an example of a partial ordering after four steps.

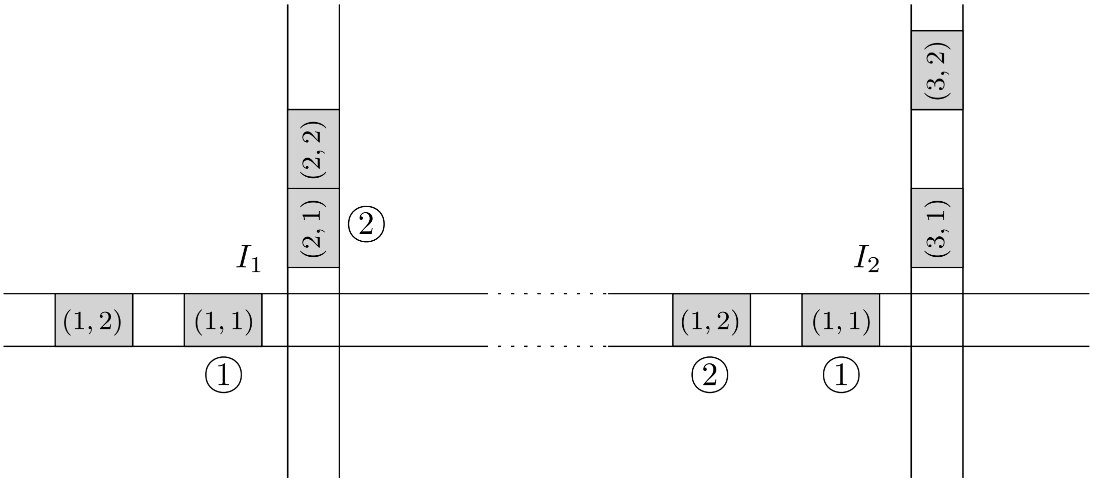

Figure 6: Illustration of some partial ordering, indicated by the encircled numbers. The order among vehicles without an encircled number is still to be decided by the scheduler, bust must respect the fact that vehicles on the same route cannot overtake. In this particular example, this means that the crossing order at $I_2$ can only be completed in one way.

This step-by-step process can be cast into the Markov Decision Process (MDP) framework. The state corresponds to the current partial ordering and each action corresponds to a valid intersection-vehicle pair to add to the partial ordering. Observe that the MDP is completely deterministic by definition. As we will show next, a complete ordering of vehicles induces a complete schedule with exact crossing time slots, so we may simply define an episodic reward in terms of the total delay of the final schedule. The disjunctive graph, see Figure 7, provides a way to encode a partial ordering by specifying the direction of a selection of disjunctive edges. Using the resulting *partial disjunctive graph* we can derive lower bounds on the crossing times. Recall that every node of the disjunctive graph corresponds to a crossing and that every arc corresponds to some constraint. We can derive lower bounds on the crossing times based on all the constraints whose arc has been added to the partial disjunctive graph by solving a linear program. However, we think that it is possible to develop a tailored update scheme that only propagates the necessary changes over the partial disjunctive graph whenever a set of disjunctive arcs is added. Finally, it is not difficult to see that the lower bounds for a complete ordering are precisely the crossing times of the corresponding schedule.

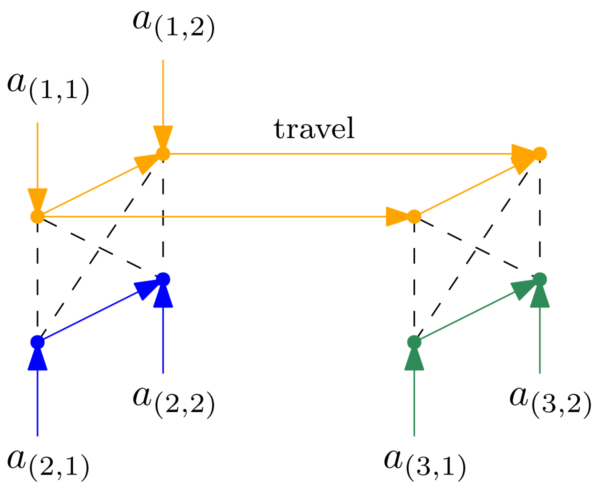

Figure 7: Illustration of a disjunctive graph corresponding to the instance in Figure 2 with respect to the network from Figure 4. For each route, conjunctive arcs are drawn at a slight angle and the horizontal arcs correspond to the travel constraints (only applicable to vehicles on route 1). The dashed lines represent disjunctive edges for the two intersections. For each node corresponding to the first intersection on the route $r$, an incoming arc labeled $a_{(r,k)}$ represents the earliest arrival time constraints. Buffer constraints are not shown, as they are not applicable to this small example.

We will now discuss how to parameterize the scheduling policy, which is a function from a partial solution (partial disjunctive graph) to an action (crossing). Previous works have been able to train good policies based on encodings of the disjunctive graph for JSSP. Following this direction, we propose build an encoding of the partial disjunctive graph augmented with the corresponding crossing time lower bounds. Graph Neural Networks (GNN) are deep neural networks that are suited to encode graph-structured data from node-level features. For each node in the (partial) disjunctive graph, we record two basic features: (i) a binary variable indicating whether this crossing has already been scheduled and (ii) the current crossing time lower bound. By iteratively combining these node-level features in a parameterized non-linear fashion, we obtain an embedding $h_O$ for every node $O$ and a global graph embedding $O_{\mathcal{G}}$. These embeddings are then mapped to an action distribution via a fully connected net and softmax function. As already noted in [5], such an architecture has the major advantage of being size-agnostic, meaning that the same policy can be applied to other instances with a different network and varying number of vehicles.

We will consider two learning paradigms for tuning the parameters in order to obtain good policies. In the *imitation learning* setting, the policy is trained to imitate expert demonstration. For example, we can use a MILP solver to obtain the optimal schedule and then backtrack the corresponding state-action pairs that our policy must take in order to arrive at the same schedule. These resulting training set of state-action pairs can then be used to tune the policy in order to mimic the expert policy as closely as possible. When expert demonstration is not available, we can directly learn from interaction with the MDP, which is what is generally known as *reinforcement learning*. Policy parameters are tuned using a policy gradient learning algorithm like the classical REINFORCE [22] or the more recent Proximal Policy Optimization [23].

### 2.3 Research plan and timeline

Time estimates are based on a total duration of 30 weeks (three quartiles of 10 weeks each) of which the last four weeks are reserved for report writing and preparing material for a final presentation.

- (done) **Direct transcription of lower-level.** The proposed bilevel decomposition depends heavily on our ability to solve the lower-level problem efficiently. Therefore, our first effort should be to develop a working direct transcription solution to obtain trajectories for a given crossing time schedule.
- (2 weeks) **Upper-level formulation.** We precisely formulate VSP for networks with delay objective (network scheduling problem). Particularly, this mainly involves formulating the travel constraints and buffer constraints such that feasibility of the lower-level problem is guaranteed. At this stage, we only aim to provide a rough proof sketch of this, because we simply verify this feasibility

- (1 week) **MILP for VSP.** Solve the network scheduling problem by formulating and solving it as a MILP. This method can be used to collect expert demonstrations for training the neural scheduler using imitation learning and it also provides a baseline for peformance evaluation of our proposed neural scheduler.
- (3 weeks) **MDP scheduling formulation.** As a first step towards implementing the actual neural scheduler, we precisely formulate the constructive combinatorial optimization in terms of a Markov Decision Process. Next, we implement a simulation of this MDP (or *environment*) that we will use to train and test our policies.
- (4 weeks) **Augmented disjunctive graph.** Next, we focus on the policy parameterization based on the disjunctive graph. First, we show how to augment the disjunctive graph with lower bounds on starting times. Each time a scheduling step is taken, the lower bounds can be efficiently recomputed using a message-passing scheme over the disjunctive graph, where the number of required messages is limited. After formulating this message-passing scheme, we prove correctness before writing the implementation.
- (4 weeks) **GNN encoding.** Based on the augmented disjunctive graph, we design a suitable embedding to parameterize the policy. We do not have much experience with the representational power of graph neural networks, so we plan to first consult the available literature to make an informed choice about the specific architecture to use. In particular, we know that not all graph neural networks are able to distinguish different graph structures, which we need for our purposes.
- (2 weeks) **Imitation learning.** Fit the policy to expert demonstration obtained from the MILP solver. We first need to extract state-action pairs from optimal solutions by some straightforward backtracking procedure. Next, we formulate and implement the corresponding supervised learning problem.
- (3 weeks) **Reinforcement learning.** Next, we decide on a suitable policy gradient method and we implement the main reinforcement learning training loop. This should be a simple coding exercise, since we already have a working environment at this point.
- (3 weeks) **Policy evaluation.** We design a series of numerical experiments to assess the performance of the learned scheduling policy. In order to make a fair comparison, we set a time limit on the MILP solver and compare schedules generated by our trained policy to those obtained from simple priority rules and those found by the MILP solver. Furthermore, since our policy is size-agnostic by design, we study how well the trained policy generalizes to instances with larger networks and larger number of vehicles.
- (4 weeks) **Explicit trajectories.** We would like to derive explicit trajectories for the lower-level trajectory optimization problem, which might be used in the future to show that the decomposition is sound, rigorously proving that valid schedules always lead to feasible lower-level problems.

### 2.4 Identified risks and their mitigation

It may turn out that our formulation of VSP does not always yield feasible lower-level problems. This can easily be verified empirically once we have a working implementation of the lower-level trajectory planning and some basic scheduling algorithm (or heuristic rule) to generate random example schedules, for which we try to compute trajectories. When some of these lower-level problems turn out to be infeasible, it is very likely that this is due to the buffer constraints, for which it is nontrivial to argue correctness. A possible solution to try in this situation is to tighten the buffer constraints, meaning that we further restrict the number of allowed vehicles between intersections.

Once we start considering the crossing time scheduling problem in networks, it is not guaranteed that even modern MILP solvers will find optimal solutions for small instances. This is a problem, because we would like to use the MILP technique as a baseline in the analysis of our neural scheduler and for obtaining expert demonstration for imitation learning. In any case, we might choose a fixed MILP optimality gap to obtain approximate schedules in limited time, or we could use simple priority rules instead.

We aim to train scheduling policies that provide reasonably good schedules in limited time for large networks and hopefully to outperform MILP-based heuristics. We expect that potential issues are mostly related to model capacity and training time. First, it might be the case that our model has not enough representational power (model capacity) to capture the complex dynamics required for good policies. In other words, it is not guaranteed that the resulting total space of possible policies does not include optimal policies or near-optimal policies. For example, it is well known that graph neural networks may differ considerably in representational power, depending on their exact formulation [24]. Therefore, we choose to first do imitation learning on expert demonstration. If this yields policies with satisfactory performance (in terms of generated schedule quality), we can have some confidence that the policy space is complex enough to yield good policies with reinforcement learning. Even if the model has enough representational power, reinforcement learning might show very slow convergence. To tackle this, we could opt to implement the $n$-step REINFORCE algorithm proposed in [6].

We mentioned in passing that we think a tailored update scheme is possible for calculating the starting time lower bounds from the disjunctive graph. Although we have a rough idea of how this would work, we did not have enough time to formulate our idea and provide arguments for it could work. However, we can obtain the lower bounds by just solving the linear program, which we expect is not too expensive to run at every transition of the reinforcement learning loop.

### 2.5 Discussion and further directions

*Time step-based modeling.* It is not difficult to imagine a time-step based model of vehicle movement through a network of intersections, which is the main principle of traffic micro-simulators like SUMO [25]. Actions may be defined in terms of increasing and decreasing the acceleration of individual vehicles and we could consider the corresponding *joint acceleration action* for all vehicles in the network. The main problem with this parameterization of trajectories is that safety is not guaranteed by design. Without additional constraints, it is possible that some sequence of joint acceleration actions eventually lead to collision. This means that the set of allowed joint actions should change with the current state. In addition to this feasibility problem, any end-to-end method that uses model-free reinforcement learning in such time step-based environment is inherently sample-inefficient, because it would implicitly be learning the vehicle dynamics, which is unnecessary. Moreover, we think that there is much more to gain by coordination on a higher level. The macroscopic phenomena that naturally occur in networks of intersection, such as the emergence of platoons, are better modeled with this higher level of abstraction.

*Full speed crossing.* For general objective functions involving some measure of energy consumption, for example in terms of acceleration, it is not always optimal to require vehicles to cross intersections at full speed. In particular, previous works that treat a similar proplem with only a single intersection [13, 14] do not make this explicit assumption. However, we propose to focus first on the higher-level decision of crossing order, because we think this has most impact on the overall quality of trajectories. Nevertheless, it remains an interesting direction for further investigation to extend our method to also allow different crossing speeds.

*Explicit trajectory expressions.* It can be observed that the resulting trajectories satisfy so-called "bang-bang" control, which means that the control input only takes both extreme values (maximum acceleration, maximum deceleration) and zero (no acceleration), in this case. Therefore, we think that it should be possible to derive expressions for the time intervals during which these three control inputs are active, as a function of the crossing times. Such explicit expressions would also help towards rigorously proving soundness of the bilevel decomposition, for which we would have to show that the lower-level problem is feasible for every valid crossing time schedule.

*Stochastic modeling.* Orthogonal to the computational challenges that the current proposal addresses is the incorporation of assumptions that make the coordination model more realistic. Most importantly, real-world systems should be robust in terms of dealing with different forms of unexpected events. For example, different kinds of failure in hardware or communication systems complicate the design of systems with guarantees on safety and efficiency. In particular, it is not always realistic to assume future arrivals to the network are known ahead of time. Instead, we assume vehicle arrive according to some (unknown) random process, which means that current trajectories may need to be reconsidered whenever a new arrival happens. Therefore, we refer to this setting as *online control*, where the focus is on finding optimal *control policies* that specify how to update trajectories over time. For the online control problem with a single intersection and delay objective, the paper [26] discusses a model based on a similar bilevel decomposition as discussed above. Whenever a new vehicle arrives to some control area, the proposed algorithm simulates the behavior of a polling policy to determine the new vehicle crossing order. It is shown that it is always possible to compute a set of collision-free trajectories from the updated schedule. Moreover, for certain classes of polling policies, explicit trajectory expressions (also refered to as *speed profile algorithms*) are available [27]. A straightforward approach to the online problem in a single intersection would be to re-optimize the crossing time schedule each time a new vehicle arrives. However, the updated schedule should always have a feasible lower-level problem, so we need to define constraints that take into account the fact that vehicles cannot stop or accelerate instantaneously. It has been shown that the feasibility of the lower-level optimization is guaranteed when the schedule is updated based on the polling policy simulation of [26], but we think that it is possible to achieve more freedom in the update step, which would allow schedules to reach closer to optimality.

# References

[1] Daniel J. Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, July 2015.

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

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

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

[5] 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.

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

[7] 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.

[8] William R. McShane and Roger P. Roess. *Traffic Engineering*. Prentice Hall Polytechnic Series in Traffic Engineering. Prentice-Hall, Englewood Cliffs, N.J, 1990.

[9] Qing He, K. Larry Head, and Jun Ding. PAMSCOD: Platoon-based arterial multi-modal signal control with online data. *Transportation Research Part C: Emerging Technologies*, 20(1):164–184, February 2012.

[10] Mohammad Noaeen, Atharva Naik, Liana Goodman, Jared Crebo, Taimoor Abrar, Zahra Shakeri Hossein Abad, Ana L.C. Bazzan, and Behrouz Far. Reinforcement learning in urban network traffic signal control: A systematic literature review. *Expert Systems with Applications*, 199:116830, August 2022.

[11] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. A Survey on Traffic Signal Control Methods, January 2020.

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

[13] 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.

[14] Weiming Zhao, Ronghui Liu, and Dong Ngoduy. A bilevel programming model for autonomous intersection control and trajectory planning. *Transportmetrica A: Transport Science*, 17(1):34–58, January 2021.

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

[16] Matthew Hausknecht, Tsz-Chiu Au, and Peter Stone. Autonomous Intersection Management: Multi-Intersection Optimization.

[17] Giorgio Sartor, Carlo Mannino, and Lukas Bach. Combinatorial Learning in Traffic Management. In Giuseppe Nicosia, Panos Pardalos, Renato Umeton, Giovanni Giuffrida, and Vincenzo Sciacca, editors, *Machine Learning, Optimization, and Data Science*, Lecture Notes in Computer Science, pages 384–395, Cham, 2019. Springer International Publishing.

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

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

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

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

[22] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, May 1992.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017.

[24] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks?, February 2019.

[25] Pablo Alvarez Lopez, Evamarie Wiessner, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flotterod, Robert Hilbrich, Leonhard Lucken, Johannes Rummel, and Peter Wagner. Microscopic Traffic Simulation using SUMO. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, Maui, HI, November 2018. IEEE.

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

[27] R. W. Timmerman and M. A. A. Boon. Platoon forming algorithms for intelligent street intersections. *Transportmetrica A: Transport Science*, 17(3):278–307, February 2021.

[28] 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.

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

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

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

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

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

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

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

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

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

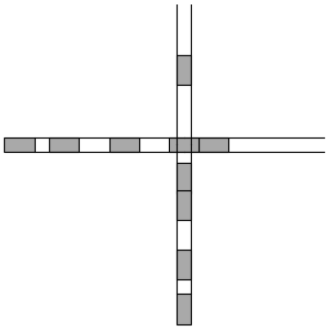

Figure 8: Illustration of a single intersection with vehicles drawn as grey rectangles. Vehicles approach the intersection from the east and from the south and cross it without turning. Note that the first two waiting vehicles on the south lane kept some distance before the intersection, such that they are able to reach full speed whenever they cross.

## Appendix A — Preliminary results for single intersection

This part presents the design of a neural scheduling heuristic for the offline trajectory optimization problem with a single intersection. We discuss how these results provide evidence that a similar methodology is feasible in a network of intersections, thereby strengthening our believe that the proposed research approach is setup to successfully answer research questions Q1 and Q2. In this preliminary stage, we used an imitation learning setting to assess whether the proposed model has enough capacity ("is complex enough") to capture policies that are close to optimal. In particular, we rely on expert demonstration consisting of labeled transitions obtained from optimal solutions computed by solving the equivalent MILP with Gurobi to train a certain scheduling policy for generating solutions to the upper-level Vehicle Scheduling Problem for a single intersection, which we show has a simple shape.

### A.1 Problem formulation

Consider a single intersection with two incoming lanes, identified by indices $\mathcal{R} = \{1, 2\}$. The corresponding two routes are crossing the intersection from south to north and crossing from west to east, see Figure 8. We identify vehicles by their route and by their relative order on this route, by defining the vehicle index set

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

where $n_r$ denotes the number of vehicles following route $r$. Smaller values of $k$ correspond to being closer to the intersection. Given vehicle index $i = (r, k) \in \mathcal{N}$, we also use the notation $r(i) = r$ and $k(i) = k$. We assume that each vehicle is represented by a rectangle of length $L$ and width $W$ and that $x_i(t)$ is measured at the front bumper. Consider the longitudinal dynamics of each vehicle $i$ along its route, for which we use the well-known *double integrator* model

$$\begin{aligned} \dot{x}_i(t) &= v_i(t), \\ \dot{v}_i(t) &= u_i(t), \\ 0 \leq v_i(t) &\leq v_{\max}, \\ |u_i(t)| &\leq a_{\max}, \end{aligned}$$

where $x_i(t)$ is the vehicle's position, $v_i(t)$ its velocity and $u_i(t)$ its acceleration, which is set by the central controller. Let $D_i(s_{i,0})$ denote the set of all trajectories $x_i(t)$ satisfying these dynamics, given some initial state $s_{i,0} = (x_i(0), v_i(0))$. In order to maintain a safe distance between consecutive vehicle on the same route, vehicle trajectories need to satisfy

$$x_i(t) - x_j(t) \geq L,$$

for all $t$ and all pairs of indices $i, j \in \mathcal{N}$ such that $r(i) = r(j), k(i) + 1 = k(j)$. Let $\mathcal{C}$ denote the set of such ordered pairs of indices, which we call *conjunctive pairs*. Note that these constraints restrict vehicle from overtaking each other, so the initial relative order is maintained on each route. For each $i \in \mathcal{N}$, let $\mathcal{E}_i = (B_i, E_i)$ denote the open interval such that vehicle $i$ occupies the intersection's conflict area if and only if $B_i < x_i(t) < E_i$. Using this notation, collision avoidance at the intersection is achieved by requiring

$$(x_i(t), x_j(t)) \notin \mathcal{E}_i \times \mathcal{E}_j,$$

for all $t$ and for all pairs of indices $i, j \in \mathcal{N}$ with $r(i) \neq r(j)$, which we collect in the set $\mathcal{D}$ and call *disjunctive pairs*. Suppose we have some performance criterion $J(x_i)$ that takes into account travel time and energy efficiency of the trajectory of vehicle $i$, then the offline trajectory optimization problem for a single intersection can be compactly written as

$$\begin{aligned} \min_{\mathbf{x}(t)} \quad & \sum_{i \in \mathcal{N}} J(x_i) \\ \text{s.t.} \quad & x_i \in D_i(s_{i,0}), && \text{for all } i \in \mathcal{N}, \\ & x_i(t) - x_j(t) \geq L, && \text{for all } (i,j) \in \mathcal{C}, \\ & (x_i(t), x_j(t)) \notin \mathcal{E}_i \times \mathcal{E}_j, && \text{for all } \{i,j\} \in \mathcal{D}, \end{aligned} \tag{1}$$

where $\mathbf{x}(t) = [\, x_i(t) : i \in \mathcal{N}\,]$ and all constraints are meant to apply to all times $t$.

### A.2 Bilevel decomposition

For the case where only a single vehicle is approaching the intersection for each route, so $n_r = 1$ for all $r \in \mathcal{R}$, it has been shown that problem (1) can be decomposed into two coupled optimization problems, see Theorem 1 in [13]. 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. When allowing multiple vehicles per route, we show (without rigorous proof) that a similar decomposition exists. Given trajectory $x_i(t)$, the *crossing time* of vehicle $i$ — when the vehicle first enters the intersection — and the corresponding *exit time* are, respectively,

$$\inf\{t : x_i(t) \in \mathcal{E}_i\} \quad \text{and} \quad \sup\{t : x_i(t) \in \mathcal{E}_i\}.$$

We will write $y(i)$ for the crossing time decision variable and write $y(i) + \sigma(i)$ for the exit time. It turns out that trajectories can be generated separately for each route, which yields a decomposition with upper-level problem

$$\begin{aligned} \min_{y,\sigma} \quad & \sum_{r \in \mathcal{R}} F(y_r, \sigma_r) \\ \text{s.t.} \quad & y(i) + \sigma(i) \leq y(j) \text{ or } y(j) + \sigma(j) \leq y(i), && \text{for all } (i,j) \in \mathcal{D}, \\ & (y_r, \sigma_r) \in \mathcal{S}_r, && \text{for all } r \in \mathcal{R}, \end{aligned} \tag{2}$$

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

$$\begin{aligned} F(y_r, \sigma_r) = \min_{x_r} \quad & \sum_{i \in \mathcal{N}(r)} J(x_i) \\ \text{s.t.} \quad & x_i \in D_i(s_{i,0}), && \text{for all } i \in \mathcal{N}_r, \\ & x_i(y(i)) = B_i, && \text{for all } i \in \mathcal{N}_r, \\ & x_i(y(i) + \sigma(i)) = E_i, && \text{for all } i \in \mathcal{N}_r, \\ & x_i(t) - x_j(t) \geq L, && \text{for all } (i,j) \in \mathcal{C} \cap \mathcal{N}_r, \end{aligned} \tag{3}$$

where we use the notation $\mathcal{N}_r = \{i \in \mathcal{N} : r(i) = r\}$ and similarly for $x_r, y_r$ and $\sigma_r$ to group variables by route. Note that the set of feasible parameters $\mathcal{S}_r$ implicitly depends on the initial states $\{s_{i,0} : i \in \mathcal{N}_r\}$ and system parameters.

### A.2.1 Bilevel decomposition for delay objective

In general, the upper- and lower-level problems are tightly coupled. However, the problem becomes much easier when we assume that the trajectory performance criterion is exactly the crossing time, so $J(x_i) = \inf\{t : x_i(t) \in \mathcal{E}_i\}$. In this case, we simply have

$$F(y_r, \sigma_r) \equiv F(y_r) = \sum_{i \in \mathcal{N}_r} y(i).$$

Furthermore, we assume that vehicles enter the network and cross the intersection at full speed, so $v_i(0) = v_i(y(i)) = v_{\max}$, such that we have

$$\sigma(i) \equiv \sigma = (L + W)/v_{\max}, \quad \text{for all } i \in \mathcal{N},$$

see Figure 3. Therefore, we ignore the part related to $\sigma$ in the set of feasible parameters $\mathcal{S}_r$, which can then be shown that to have a particularly simple structure under these assumptions. Observe that $r_i = (B_i - x_i(0))/v_{\max}$ is the earliest time at which vehicle $i$ can enter the intersection, when it keeps driving at maximum velocity. Let $\rho = L/v_{\max}$ be such that $y(i) + \rho$ is the time at which the rear bumper of a crossing vehicle reaches the start line of the intersection, see Figure 3, then it can be shown that $y_r \in \mathcal{S}_r$ if and only if

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

Therefore, under the stated assumptions, problem (1) reduces to the following *crossing time scheduling* problem

$$\min_{y} \quad \sum_{i \in \mathcal{N}} y(i) \tag{4a}$$

$$\text{s.t.} \quad r_i \leq y(i), \qquad \text{for all } i \in \mathcal{N}, \tag{4b}$$

$$y(i) + \rho \leq y(j), \qquad \text{for all } (i,j) \in \mathcal{C}, \tag{4c}$$

$$y(i) + \sigma \leq y(j) \text{ or } y(j) + \sigma \leq y(i), \qquad \text{for all } (i,j) \in \mathcal{D}, \tag{4d}$$

which can be solved using off-the-shelf mixed-integer linear program solvers, after encoding the *disjunctive constraints* (4d) using the big-M technique. Given optimal $y^*$, any set of trajectories $[x_i(t) : i \in \mathcal{N}]$ that satisfies

$$\begin{aligned} x_i &\in D_i(s_{i,0}), && \text{for all } i \in \mathcal{N}, \\ x_i(y^*(i)) &= B_i, && \text{for all } i \in \mathcal{N}, \\ x_i(y^*(i) + \sigma) &= E_i, && \text{for all } i \in \mathcal{N}, \\ x_i(t) - x_j(t) &\geq L, && \text{for all } (i,j) \in \mathcal{C}, \end{aligned}$$

forms a valid solution. These trajectories can be computed by an efficient direct transcription method. First, note that each route may be considered separately. Trajectories on each route can then be computed in a sequential fashion by repeatedly solving the optimal control problem

$$\begin{aligned}
&\texttt{MotionSynthesize}(\tau, B, s_0, x') := \\
&\qquad \arg\min_{x:[0,\tau]\to\mathbb{R}} \int_0^\tau |x(t)| dt \\
&\qquad\qquad \text{s.t. } \ddot{x}(t) = u(t), && \text{for all } t \in [0,\tau], \\
&\qquad\qquad |u(t)| \leq a_{\max}, && \text{for all } t \in [0,\tau], \\
&\qquad\qquad 0 \leq \dot{x}(t) \leq v_{\max}, && \text{for all } t \in [0,\tau], \\
&\qquad\qquad x'(t) - x(t) \geq L, && \text{for all } t \in [0,\tau], \\
&\qquad\qquad (x(0), \dot{x}(0)) = s_0, \\
&\qquad\qquad (x(\tau), \dot{x}(\tau)) = (B, v_{\max}),
\end{aligned}$$

where $\tau$ is set to the required crossing time, $B$ denotes the distance to the intersection, $s_0$ is the initial state of the vehicle and $x'$ denotes the trajectory of the vehicle preceding the current vehicle, which is required to maintain a safe following distance between the two vehicles.

### A.2.2 Extension to networks of intersections

We briefly explain why we think that the above setup can be extended to a network of intersections, thereby strengthening our believe that we can successfully answer research question Q1. First of all, the upper-level problem can be extended by considering a crossing time variable $y(i, v)$ for every intersection $v$ on the route $r(i)$ of each vehicle $i \in \mathcal{N}$. Given a valid schedule $y$ of crossing times, we can then imagine a similar lower-level trajectory optimization problem that generates collision-free trajectories for every vehicle. The main difficulty is making sure that the constraints in the upper-level problem, which we denoted by $(y_r, \sigma_r) \in \mathcal{S}_r$, are such that schedules $y$ always result in a feasible lower-level problem.

We identify at least two new types of constraints that need to be considered in a network. We need to take into account the time that vehicles need to drive towards the next intersection on their route, which requires some kind of *travel constraints*, which can be defined relatively straightforward in the current model. Furthermore, lanes between intersections can only provide room for a finite number of driving or waiting vehicles, requiring some sort of *buffer constraints*. We expect that it is nontrivial to define these and prove their correctness, since we need to take into account a combination of both position and velocity of vehicles entering a lane.

## A.3 Crossing time scheduling

Partial solutions to the crossing time scheduling problem (4) can be represented clearly by their *disjunctive graph*, which we define next. Let $(\mathcal{N}, \mathcal{C}, \mathcal{O})$ be a directed graph with nodes $\mathcal{N}$ and the following two types of arcs. The *conjunctive arcs* encode the fixed order of vehicles having the same route. For each $(i, j) \in \mathcal{C}$, an arc from $i$ to $j$ means that vehicle $i$ reaches the intersection before $j$ due to the follow constraints (4c). The *disjunctive arcs* are used to encode the decisions regarding the ordering of vehicles on distinct routes, corresponding to constraints (4d). For each pair $\{i, j\} \in \mathcal{D}$, at most one of the arcs $(i, j)$ and $(j, i)$ can be present in $\mathcal{O}$.

When $\mathcal{O} = \varnothing$, we say the disjunctive graph is *empty*. Each feasible schedule satisfies exactly one of the two constraints in (4d). When $\mathcal{O}$ contains exactly one arc from every pair of opposite disjunctive arcs, we say the disjunctive graph is *complete*. Note that such graph is acyclic and induces a unique topological ordering $\pi$ of its nodes. Conversely, every ordering $\pi$ of nodes $\mathcal{N}$ corresponds to a unique complete disjunctive graph, which we denote by $G(\pi) = (\mathcal{N}, \mathcal{C}, \mathcal{O}(\pi))$.

We define weights for every possible arc in a disjunctive graph. Every conjunctive arc $(i, j) \in \mathcal{C}$ gets weight $w(i, j) = \rho_i$ and every disjunctive arc $(i, j) \in \mathcal{O}$ gets weight $w(i, j) = \sigma_i$. Given some vehicle ordering $\pi$, for every $j \in \mathcal{N}$, we recursively define the lower bound

$$\mathrm{LB}_\pi(j) = \max\{r_j, \max_{i \in N_\pi^-(j)} \mathrm{LB}_\pi(i) + w(i, j)\}, \tag{5}$$

where $N_\pi^-(j)$ denotes the set of in-neighbors of node $j$ in $G(\pi)$. Observe that this quantity is a lower bound on the crossing time, i.e., every feasible schedule $y$ with ordering $\pi$ must satisfy $y_i \geq \mathrm{LB}_\pi(i)$ for all $i \in \mathcal{N}$.

### A.3.1 Branch-and-bound solution

Optimization problem (4) can be turned into a Mixed-Integer Linear Program (MILP) by rewriting the disjunctive constraints using the well-known big-M method. We introduce a binary decision variable $\gamma_{ij}$ 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 with choosing disjunctive arc $i \to j$ and $\gamma_{ij} = 1$ corresponds to $j \to i$. This yields the following MILP formulation

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

where $M > 0$ is some sufficiently large number, which may depend on the specific problem instance.

### A.3.2 Neural construction heuristic

Define partial ordering $\pi$ to be a *partial permutation* of $\mathcal{N}$, which is a sequence of elements from some subset $\mathcal{N}(\pi) \subset \mathcal{N}$. Let $\pi$ be a partial ordering of length $n$ and let $i \notin \mathcal{N}(\pi)$, then we use $\pi' = \pi +\!\!+\, i$ to denote the concatenation of sequence $\pi$ with $i$, so $\pi'_{1:n} = \pi_{1:n}$ and $\pi'_{n+1} = i$. Furthermore, recursively define the concatenation of two sequences by $\pi +\!\!+\, \pi' = (\pi +\!\!+\, \pi'_1) +\!\!+\, \pi'_{2:m}$, where $m$ is the length of $\pi'$. For each partial ordering $\pi$, the corresponding disjunctive graph $G(\pi)$ is incomplete, meaning that some of the disjunctive arcs have not yet been added. Nevertheless, observe that $\mathrm{LB}_\pi(i)$ is still defined for every $i \in \mathcal{N}$. Let $\mathrm{obj}(\pi)$ denote the objective of a complete ordering $\pi$.

Observe that, due to the conjunctive constraints, ordering vehicles is equivalent to ordering routes. We will define heuristics in terms of repeatedly choosing to which route the next vehicle in the ordering belongs. It may be helpful to model this as a deterministic finite-state automaton, where the set of route indices $\mathcal{R}$ acts as the input alphabet. Let $S$ denote the state space and let $\delta : S \times \mathcal{R} \to S$ denote the state-transition function.

Let $s$ denote an instance of (4). We consider $s$ to be a fixed part of the state, so it does not change with state transitions. The other part of the state is the current partial ordering $\pi$. The transitions of the automaton are very simple. Let $(s, \pi) \in S$ denote the current state and let $r \in \mathcal{R}$ denote the next symbol. Let $i \in \mathcal{N} \setminus \mathcal{N}(\pi)$ denote the next unscheduled vehicle on route $r$, then the system transitions to $(s, \pi +\!\!+\, i)$. If no such vehicle exists, the transition is undefined. Therefore, an input sequence $\eta$ of routes is called a *valid route order* whenever it is of length

$$N = \sum_{r \in \mathcal{R}} n_r$$

and contains precisely $n_r$ occurrences of route $r \in \mathcal{R}$. Given problem instance $s$, let $y_\eta(s)$ denote the schedule corresponding to route order $\eta$. We say that route order $\eta$ is optimal whenever $y_\eta(s)$ is optimal. Observe that an optimal route order must exist for every instance $s$, since we can simply derive the route order from an optimal vehicle order.

Instead of mapping an instance $s$ directly to some optimal route order, we consider a mapping $p : S \to \mathcal{R}$ such that setting $s_0 = (s, \varnothing)$ and repeatedly evaluating

$$s_t = \delta(s_{t-1}, p(s_{t-1}))$$

yields a final state $s_N(s, \pi^*)$ with optimal schedule $\pi^*$. Observe that this mapping must exist, because given some optimal route order $\eta^*$, we can set $p(s_t) = \eta^*_{t+1}$, for every $t \in \{0, \dots, N-1\}$.

We do not hope to find an explicit representation of $p$, but our aim is to find good heuristic approximations. For example, consider the following simple *threshold rule*. Let $\pi$ denote a partial schedule of length $n$, so $i = \pi(n)$ is the last scheduled vehicle with route $\texttt{last}(\pi) = r(i)$, then define

$$p_\tau(s, \pi) = \begin{cases} \texttt{last}(\pi) & \text{if } \mathrm{LB}_\pi(i) + \rho_i + \tau \geq r_j \text{ and } (i, j) \in \mathcal{C}, \\ \texttt{next}(\pi) & \text{otherwise,} \end{cases}$$

for some threshold parameter $\tau \geq 0$. The expression $\texttt{next}(\pi)$ represents some route other than $\texttt{last}(\pi)$ with unscheduled vehicles left.

### A.3.3 Imitation learning

Instead of explicitly formulating heuristics using elementary rules, we will now consider a data-driven approach. To this end, we model the conditional distribution $p_\theta(\eta_{t+1} | s_t)$ with parameters $\theta$. Consider an instance $s$ and some optimal route sequence $\eta$ with corresponding states defined as $s_{t+1} = \delta(s_t, \eta_{t+1})$ for $t \in \{0, \dots, N-1\}$. The resulting set of pairs $(s_t, \eta_{t+1})$ can be used to learn $p_\theta$ in a supervised fashion by treating it as a classification task.

Schedules are generated by employing *greedy inference* as follows. The model $p_\theta$ provides a distribution over routes. We ignore routes that have no unscheduled vehicles left and take the argmax of the remaining probabilities. We will denote the corresponding complete schedule by $\hat{y}_\theta(s)$.

Next, we discuss the parameterization of the model. We first derive, for every $r \in \mathcal{R}$, a *route embedding* $h(s_t, r)$ based on the current non-final state $s_t = (s, \pi_t)$ of the automaton. Let $k_\pi(r)$ denote the first unscheduled vehicle in route $r$ under the partial schedule $\pi_t$. Denote the smallest lower bound of unscheduled vehicles as

$$T_\pi = \min_{i \in \mathcal{N} \setminus \mathcal{N}(\pi)} \mathrm{LB}_\pi(i).$$

Let the *horizon* of route $r$ be defined as

$$h'(s_t, r) = (\mathrm{LB}_{\pi_t}(k_{\pi_t}(r)) - T_{\pi_t}, \dots, \mathrm{LB}_{\pi_t}(n_r) - T_{\pi_t}).$$

Observe that horizons can be of arbitrary dimension. Therefore, we restrict each horizon to a fixed length $\Gamma$ and use zero padding. More precisely, given a sequence $x = (x_1, \dots, x_n)$ of length $n$, define the padding operator

$$\text{pad}(x, \Gamma) = \begin{cases} (x_1, \dots, x_\Gamma), & \text{if } \Gamma \leq n, \\ (x_1, \dots, x_n) \mathbin{+\!\!+} (\Gamma - n) * (0), & \text{otherwise,} \end{cases}$$

where we use the notation $n * (0)$ to mean a sequence of $n$ zeros. The route embedding is then defined as

$$h(s_t, r) = \text{pad}(h'(s_t, r), \Gamma).$$

Next, these are then arranged into a *state embedding* $h(s_t)$ as follows. Let $\eta_t$ be the route that was chosen last, then we apply the following *route cycling* trick in order to keep the most recent route in the same position of the state embedding, by defining

$$h_r(s_t) = h(s_t,\ r - \eta_t \bmod |\mathcal{R}|),$$

for every $r \in \mathcal{R}$. This state embedding is then mapped to a probability distribution

$$p_\theta(\eta_{t+1}|s_t) = f_\theta(h(s_t)),$$

where $f_\theta$ is a fully connected neural network.

### A.3.4 Experiments and discussion

First, we define some classes of problem instances that we will use as a benchmark. Instances are generated by sampling values of $g_i \sim G$ and $\rho_i \sim R$ according to the parameters given in Table 1 and setting

$$r_i = \sum_{k=1}^{k(i)} g_i + \sum_{k=1}^{k(i)-1} \rho_i,$$

for each $i \in \mathcal{N}$. For each set specification in Table 1, we sample 1000 instances.

We solve each instance's MILP to optimality to obtain the training data set $\mathcal{X}$, consisting of all the pairs $(s_t, \eta_{t+1})$ of state and next route belonging to the optimal schedules. We fit the simple threshold heuristic to $\mathcal{X}$ by finding the value of $\tau$ with the lowest mean objective using a simple grid search. In order to fit the model parameters of the neural models, we interpret $p_\theta(s_t)$ as the probability of choosing the first route and use the binary cross entropy loss, defined as

$$-\frac{1}{|\mathcal{X}|} \sum_{(s_t, \eta_{t+1}) \in \mathcal{X}} \mathbb{1}\{\eta_{t+1} = 1\} \log(p_\theta(s_t)) + \mathbb{1}\{\eta_{t+1} = 2\} \log(1 - p_\theta(s_t)),$$

where $\mathbb{1}(\cdot)$ denotes the indicator function. We use 5 epochs with batch size 10 and we use learning rate $10^{-3}$ with the Adam optimizer.

Let $\mathcal{Y}$ denote one of the test instance sets. Let obj$(y)$ denote the objective of schedule $y$, and let $\hat{y}$ denote the schedule obtained from a heuristic, then Table 2 reports on the average approximation ratio defined as

$$\alpha_{\text{approx}} = \frac{1}{|\mathcal{Y}|} \sum_{s \in \mathcal{Y}} \text{obj}(\hat{y}(s)) \ / \ \text{obj}(y^*(s))$$

and the fraction of instances that are solved to optimality, which we define as

$$\alpha_{\text{opt}} = \frac{1}{|\mathcal{Y}|} \sum_{s \in \mathcal{Y}} \mathbb{1}\{\text{obj}(\hat{y}(s)) = \text{obj}(y^*(s))\}.$$

Observe that the policy parameterization is complex enough to model near-optimal constructive policies for this single intersection scheduling problem. However, these results should be interpreted with some care, as the very simple threshold policy also achieves very close to optimal performance, which might suggest that there is some fundamental structure hidden in the problem. The fact that the relatively simple neural policy is able to capture the apparent structure in the problem for a single intersection gives us hope to obtain similar results whenever we consider a network of intersections. This provides some motivation that research question Q2 might eventually allow a positive answer.

Table 1: Specification of test instance sets. The total number of vehicles is shown as the sum of the number of vehicles per route $\sum n_r$. Every instance has switch-over time $\sigma = 2$.

| set id | size | $\lvert\mathcal{N}\rvert$ | $G$ | $R$ |
|---|---|---|---|---|
| 1 | 100 | $10 + 10$ | $\text{Uni}(0, 4)$ | $1$ |
| 2 | 100 | $15 + 15$ | $\text{Uni}(0, 4)$ | $1$ |
| 3 | 100 | $20 + 20$ | $\text{Uni}(0, 4)$ | $1$ |
| 4 | 100 | $25 + 25$ | $\text{Uni}(0, 4)$ | $1$ |
| 5 | 100 | $10 + 10$ | $\text{Exp}(2)$ | $1$ |
| 6 | 100 | $15 + 15$ | $\text{Exp}(2)$ | $1$ |

Table 2: Mean approximation ratio $\alpha_{\text{approx}}$ and proportion of instances solved to optimality $\alpha_{\text{opt}}$. Evaluated for each set of test instances for the fitted *threshold* heuristic and the fitted neural heuristic with *padded* embedding. Each set contains 100 instances.

|  | $\alpha_{\text{approx}}$ |  | $\alpha_{\text{opt}}$ |  |
|---|---|---|---|---|
| set id | threshold | padded | threshold | padded |
| 1 | 1.0198 | 1.0085 | 0.12 | 0.33 |
| 2 | 1.0132 | 1.0102 | 0.12 | 0.20 |
| 3 | 1.0102 | 1.0079 | 0.11 | 0.17 |
| 4 | 1.0088 | 1.0054 | 0.05 | 0.13 |
| 5 | 1.0359 | 1.0120 | 0.19 | 0.35 |
| 6 | 1.0258 | 1.0110 | 0.12 | 0.23 |

## Appendix B — In-depth discussion of related literature

### B.1 Traffic coordination

A good example of an early centralized approach is the "Autonomous Intersection Management" (AIM) paper [12], which is based on a reservation scheme. The conflict zone is modeled by 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.

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 [28] or Gurobi [29]). 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 really useful for toy problems. Therefore, approximation schemes have been studied in previous works [13, 14, 15].

The approximation method in [13] is based on this bilevel decomposition and considers an 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 that satisfies the crossing time schedule by solving a quadratic program. This approach has been shown to reduce running times significantly. Unfortunately, this study is limited to a single intersection and it is assumed that each lane approaching the intersection contains exactly one vehicle. The paper [14] 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 levels are solved in an alternating fashion, each time updating the constraints of the other problem based on the current solution. The optimization scheme in [15] 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 [30] is used that guarantees safety to reach the assigned crossing times.

### B.2 Neural combinatorial optimization

This section introduces the idea of applying a Machine Learning (ML) perspective on Combinatorial Optimization (CO) problems, which has recently gained a lot of attention. 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 [18]. It is very natural to see the sequential decision-making process of any optimization algorithm in terms of the Markov Decision Process (MDP) framework, where the environment corresponds to the internal state of the algorithm. From this perspective, two main learning regimes can be distinguished. Methods like those based on the branch-and-bound framework are often computationally too expensive for practical purposes, so *learning to imitate* the decisions taken in these exact algorithms might provide us with fast approximations. In this approach, the ML model's performance is measured in terms of how similar the produced decisions are to the demonstrations provided by the expert. On the other hand, some problems do not even enable exact methods, so it is interesting to study solution methods that *learn from experience*. An interesting feature of this kind of approach is that it enables us to implicitly 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 [19]. 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 [31] 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 [32]. 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.

A major challenge in NCO is constraint satisfaction. For example, solutions produced by constructive neural policies need to satisfy the constraints of the original combinatorial problem. Therefore, an important question is how to enforce these constraints. To this end, neural network architectures have been designed whose outputs satisfy some kind of constraint, for example being a permutation of the input [33]. 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 be only visited once can be easily enforced by ignoring the visited nodes and taking the argmax among the model's probabilities for unvisited nodes.

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 [7] provides an overview based on this distinguishing feature.

A very natural approach to model JSSP in terms of an MDP is taken in [34], 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 [35] and Demirkol [36] 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 [20], 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 [5] 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 $O_{11} \rightarrow O_{22} \rightarrow O_{32}$. 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.

Recently, the authors of L2D investigated an improvement heuristic for JSSP [6] 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.

An example of a joint method is given in [37], where the environment is stated in terms of a Constraint Programming (CP) formulation. This allows the method to be trained using demonstration from an off-the-shelf CP solver.