Learning to Coordinate Autonomous Vehicles through Networks of Intersections using Neural Combinatorial Optimization

Jeroen van Riel

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.

Overall aim and goals

Motivation and challenges

Given the ongoing development of self-driving vehicle technology, it is not hard 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 spend 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 about 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].

We can identify different levels at which coordination can take place [2]. At a local level, platooning of vehicles may reduce energy consumption due to the reduced aerodynamic resistance and may result in more efficient use of intersections. On the other end of the spectrum, network-wide dynamic route optimization can be employed to reduce the travel delay for all users of the system. We choose to study a global model of coordination in which every vehicle in the network is controlled by a single

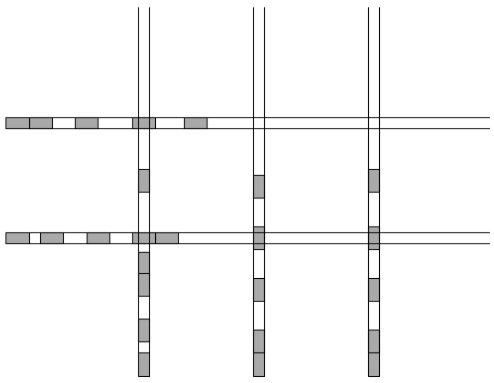

Figure 1: Illustration of some grid-like network of intersections with vehicles drawn as grey rectangles. There are five vehicle routes: two from east to west and three from south to north. Turning at intersections is not allowed.

entity, to which we will simply refer as the *controller*. We ignore the choice of routes by assuming that each vehicle follows a fixed predetermined route.

We identify the following two main challenges that are not fully addressed by existing coordination schemes. 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. Even when safety is guaranteed, optimal decision making is still a very computationally challenging task due to the complex interactions and constraints between vehicle trajectories that emerge in large traffic networks. Therefore, an important research direction is the design of computationally efficient coordination schemes with safety guarantees that scale to large networks with large numbers of vehicles. Orthogonal to the above issues 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.

Motivated by successful applications of reinforcement learning to highly complex control tasks, we aim to apply a similar perspective to traffic coordination in order to automatically learn how to deal with the complex decision-making problem. However, 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 [3]. We observe that traffic coordination shows some resemblance with classical job-shop scheduling problems, for which successful reinforcement learning methods have recently been proposed [4, 5, 6], motivating the investigation of a similar approach to the current traffic coordination problem.

Broad literature analysis

Traffic lights are currently the most effective way of steering traffic, so current coordination methods often rely on some sort of synchronization of traffic light settings at neighboring intersections [7, 8], 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 [9, 10]. 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 [11], current coordination schemes are studied under very different model assumptions regarding vehicle dynamics and modes of communication and control [12]. An important distinction can be made based on the used communication models, which can be roughly categorized in 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]. 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].

Problem formulation and objectives

We are interested in centralized methods for network-wide coordination of autonomous vehicles without relying on traffic lights. More precisely, we model coordination as an optimal control problem, with assumptions of perfect communication and control. Imagine that some central coordinator controls the acceleration of all individual vehicles, each modeled as a double integrator. After a vehicle arrives to the network, it follows a predetermined route and leaves again. The goal is to control the trajectory of each vehicle in the network while avoiding collisions and optimizing some global measure of efficiency. A natural measure of efficiency that is commonly used in literature is total delay experienced by all vehicles. However, it might be desirable to also penalize acceleration as a proxy for energy consumption. When all vehicle arrival times are known in advance, we may assume that trajectories can be computed without any prior interaction with the system, so we refer to this setting as *offline trajectory optimization.*

Our overall goal is to investigate the limits of centralized coordination algorithms in terms of performance and computational complexity. Therefore, we state the following two research questions:

- What are the fundamental limits on performance gains that centralized network-wide coordination of autonomous vehicles can bring?
- Is there any structure in the trajectory optimization problem that can be exploited by coordination algorithms?

To answer these questions, we propose to develop computationally efficient coordination algorithms for the model skeched above. Results obtained in such stylized models provide rough upper bounds on the performance gains we can expect in a more realistic setting, where vehicle hardware and communication systems are assumed to be less perfect. More concretely, we aim to investigate how deep reinforcement learning can be applied to the offline trajectory optimization problem, because we are convinced that a data-driven perspective is beneficial in complex decision-making problems (either deterministic or with uncertainty) where good policies are unlikely to display simple structure. To this end, we propose to develop a coordination algorithm that satisfies the following three key requirements:

- *Safety.* Because vehicle dynamics are assumed to be known precisely, hard collisions-avoidance constraints can be formulated, which the controller needs to respect at all times. Because safety is so crucial in real-world application, we feel it does not make sense to study algorithms without this guarantee.
- *Scalability.* We want our coordination algorithm to naturally scale to complex networks with large numbers of vehicles in such a way that the required level of optimality can be achieved by providing sufficient computational resources.
- *Learning.* The previous objective calls for the use of approximation schemes, because the computational complexity of the general coordination problem does not allow scalable solutions. Our hypothesis is that there is a lot of structure in the problem which can be algorithmically exploited, but might be hard to formulate precisely. Therefore, the algorithm must be able to automatically learn how to exploit the structure in some class of problem instances.

Research approach

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 coordinator 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 along their route — and a *lower-level* optimal control problem to find trajectories that match these crossing times, see Figure 2 for an illustration.

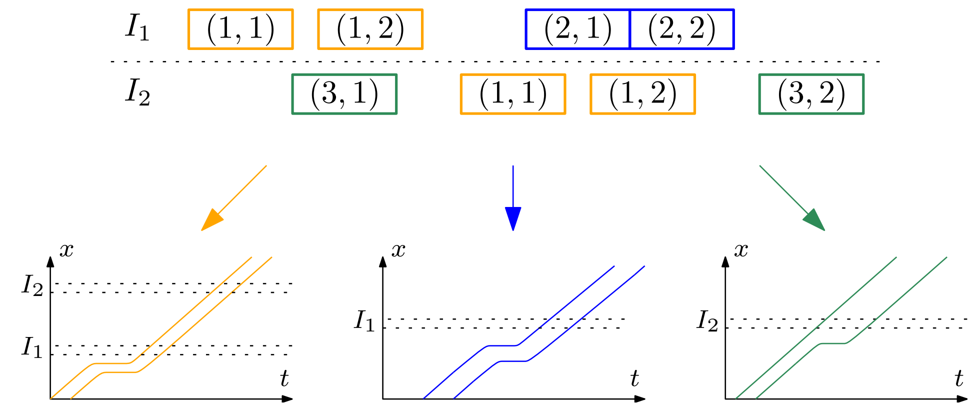

Figure 2: Illustration of the proposed bilevel decomposition. The upper-level problem is assign each vehicle a time slot for each of the intersections (here denoted I_1 and I_2) on its route. Based on this schedule, the lower-level problem is to generate trajectories satisfying these time slots.

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 along 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.

Models and methods

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 $y + \sigma$.

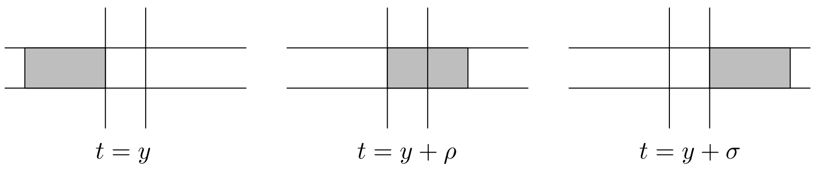

Figure 3: Three different snapshots of a vehicle crossing an intersection.

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.

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.

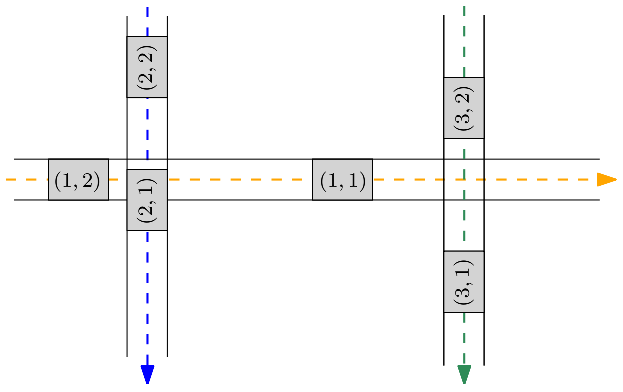

Figure 4: Example network with three different routes satisfying the edge-disjoint assumption (such that vehicle flows never merge) and vehicle indices (r, k).

We now explain how JSSP can be extended to our Vehicle Scheduling Problem (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 along this route is modeled as an operation. The processing time of a crossing corresponds to ρ 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 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 σ time (see Figure 3) between crossings of vehicles belonging to different routes.
- *Travel constraints.* Vehicles need to physically drive towards the next intersection along 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.

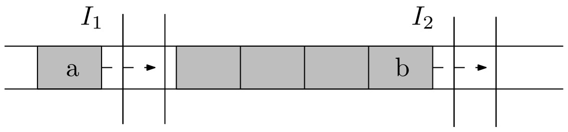

Figure 5: Illustration of buffer constraint for a pair of vehicles on the same route. Suppose that the vehicles on the lane between the two intersections are currently not moving and waiting to cross I_2. It is clear that the crossing time of vehicle a at I_1 must be happen at least after vehicle b starts 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 $k_2 = k_1 + 3$.

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 very 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.

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 along 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.

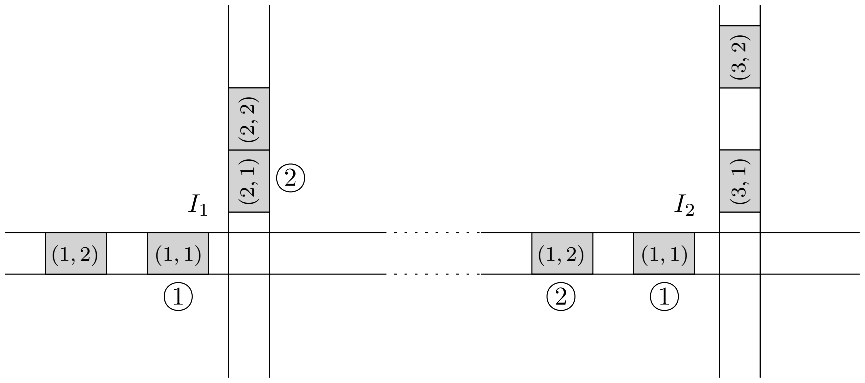

Figure 6: Illustration of some partial ordering, indicated by the encircled numbers. The order among vehicles without an encircled number is still to be decided by the scheduler, bust must respect the fact that vehicles on the same route cannot overtake. In this particular example, this means that the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

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.

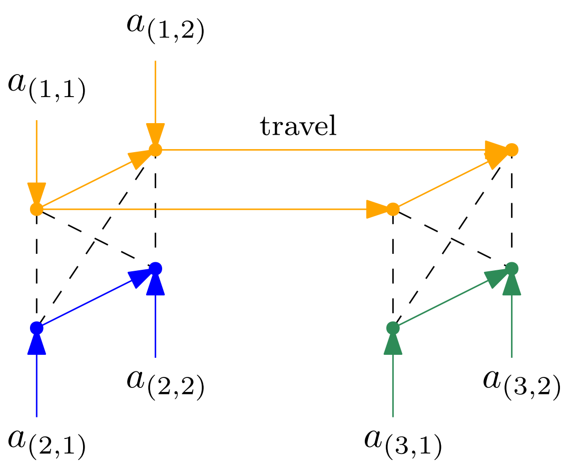

Figure 7: Illustration of a disjunctive graph corresponding to the instance in Figure 2 with respect to the network from Figure 4. For each route, conjunctive arcs are drawn at a slight angle and the horizontal arcs correspond to the travel constraints (only applicable to vehicles on route 1). The dashed 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.

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 [4], 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].

Research plan and timeline

Time estimates are based on a total duration of 20 weeks of which the last two 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

- (done) **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.

- (2 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.

- (3 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.

- (2 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.

- (2 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.

- (2 weeks, not essential) **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.

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 intersection.

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 [5].

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.

Discussion and further directions

Why not model the coordination problem in terms of time steps? It is not difficult to image 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. 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.

Bibliography

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

[2] S. Mariani, G. Cabri, and F. Zambonelli, “Coordination of Autonomous Vehicles: Taxonomy and Survey,” *ACM Computing Surveys*, vol. 54, pp. 1–33, Jan. 2022.

[3] Y. Min, A. Sonar, and N. Azizan, “Hard-Constrained Neural Networks with Universal Approximation Guarantees,” Oct. 2024.

[4] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and C. Xu, “Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning,” Oct. 2020.

[5] C. Zhang, Z. Cao, W. Song, Y. Wu, and J. Zhang, “Deep Reinforcement Learning Guided Improvement Heuristic for Job Shop Scheduling,” Feb. 2024.

[6] I. G. Smit, J. Zhou, R. Reijnen, Y. Wu, J. Chen, C. Zhang, Z. Bukhsh, W. Nuijten, and Y. Zhang, “Graph Neural Networks for Job Shop Scheduling Problems: A Survey,” 2024.

[7] W. R. McShane and R. P. Roess, *Traffic Engineering.* Prentice Hall Polytechnic Series in Traffic Engineering, Englewood Cliffs, N.J: Prentice-Hall, 1990.

[8] Q. He, K. L. Head, and J. Ding, “PAMSCOD: Platoon-based arterial multi-modal signal control with online data,” *Transportation Research Part C: Emerging Technologies*, vol. 20, pp. 164–184, Feb. 2012.

[9] M. Noaeen, A. Naik, L. Goodman, J. Crebo, T. Abrar, Z. S. H. Abad, A. L. Bazzan, and B. Far, “Reinforcement learning in urban network traffic signal control: A systematic literature review,” *Expert Systems with Applications*, vol. 199, p. 116830, Aug. 2022.

[10] H. Wei, G. Zheng, V. Gayah, and Z. Li, “A Survey on Traffic Signal Control Methods,” Jan. 2020.

[11] K. Dresner and P. Stone, “A Multiagent Approach to Autonomous Intersection Management,” *Journal of Artificial Intelligence Research*, vol. 31, pp. 591–656, Mar. 2008.

[12] M. Khayatian, M. Mehrabian, E. Andert, R. Dedinsky, S. Choudhary, Y. Lou, and A. Shirvastava, “A Survey on Intersection Management of Connected Autonomous Vehicles,” *ACM Transactions on Cyber-Physical Systems*, vol. 4, pp. 1–27, Oct. 2020.

[13] R. Hult, G. R. Campos, P. Falcone, and H. Wymeersch, "An approximate solution to the optimal coordination problem for autonomous vehicles at intersections," in *2015 American Control Conference (ACC)*, (Chicago, IL, USA), pp. 763–768, IEEE, July 2015.

[14] W. Zhao, R. Liu, and D. Ngoduy, "A bilevel programming model for autonomous intersection control and trajectory planning," *Transportmetrica A: Transport Science*, vol. 17, pp. 34–58, Jan. 2021.

[15] P. Tallapragada and J. Cortés, "Hierarchical-distributed optimized coordination of intersection traffic," Jan. 2017.

[16] M. Hausknecht, T.-C. Au, and P. Stone, "Autonomous Intersection Management: Multi-Intersection Optimization,"

[17] G. Sartor, C. Mannino, and L. Bach, "Combinatorial Learning in Traffic Management," in *Machine Learning, Optimization, and Data Science* (G. Nicosia, P. Pardalos, R. Umeton, G. Giuffrida, and V. Sciacca, eds.), Lecture Notes in Computer Science, (Cham), pp. 384–395, Springer International Publishing, 2019.

[18] Y. Bengio, A. Lodi, and A. Prouvost, "Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon," Mar. 2020.

[19] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement Learning for Combinatorial Optimization: A Survey," Dec. 2020.

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

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

[22] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017.

[24] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?," Feb. 2019.

[25] P. A. Lopez, E. Wiessner, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flotterod, R. Hilbrich, L. Lucken, J. Rummel, and P. Wagner, "Microscopic Traffic Simulation using SUMO," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, (Maui, HI), pp. 2575–2582, IEEE, Nov. 2018.

[26] D. Miculescu and S. 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*, vol. 17, pp. 278–307, Feb. 2021.