Introduction

Various models and systems have been proposed in the literature for the problem of coordinating autonomous vehicles through urban road networks. Previous works vary considerably in the types of communication model, 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. We are mainly interested in the latter type, so we will discuss some examples from the literature in the next section. For a more complete overview of previous works, we refer to the survey [1].

In its most pure form, the coordination of autonomous vehicles through a road network with intersections may be regarded as a high-dimensional optimal control problem. Most previous works with this perspective use the a simple one-dimensional vehicle model known as the double integrator [2] in optimal control literature. Imagine that some central coordinator controls the acceleration of each individual vehicle. 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*.

A key issue for the coordinator is to decide the *order of crossing* at intersections, which is precisely what makes the optimization problem non-convex and thus hard to solve with standard methods. Under some assumptions, the problem can be shown to decompose in a combinatorial problem to determine a *crossing time schedule* — indicating when vehicles cross the intersections along their route — and an optimal control problem to find trajectories that match these crossing times. We propose to address the scheduling part by applying recent machine learning methods for combinatorial optimization.

Autonomous intersection control

A good example of an early centralized approach is the "Autonomous Intersection Management" (AIM) paper [3], 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 [4] or Gurobi [5]). 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. A common observation is that the optimization problem may be thought of as two coupled optimization problems, 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. The approximation method in [6] 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 [7] 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 [8] 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 [9] is used that guarantees safety to reach the assigned crossing times.

Job-shop scheduling

Optimizing the crossing time schedule in a network of intersections is very similar to the classical Job-Shop Scheduling Problem (JSSP). Therefore, we briefly discuss this class of models such that we can use the related terminology and notation in the following. 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 $O_{i1}, \ldots, O_{in_i}$ 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 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 direct acyclic graph.

Neural combinatorial optimization

This section will introduce 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 [10]. 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 [11]. 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. 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 [12]. 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 [13]. 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 [14] provides an overview based on this distinguishing feature.

A very natural approach to model JSSP in terms of an MDP is taken in [15], 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 [16] and Demirkol [17] 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 [18], 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 [19] proposes a construction heuristic for JSSP with makespan objective. Their method is based on a dispatching policy that is parameterized in terms of a graph neural network encoding of the disjunctive graph belonging to a partial solution. Again, each action corresponds to choosing for which job the next operation is dispatched. The rewards are based on how much the lower bound on the makespan changes between consecutive states. They use a Graph Isomorphism Network (GIN) architecture to parameterize both an actor and critic, which are trained using the Proximal Policy Optimization (PPO) algorithm. Using the Taillard and Demirkol benchmarks, they show that their model is able to generalize well to larger instances. As we already alluded to above, this way of modeling the environment is better suited to JSSP with regular objectives, because it does not explicitly determine starting times. They use a dispatching mechanism based on finding the earliest starting time of a job, even before already scheduled jobs, see their Figure 2. By doing this, they introduce symmetry in the environment: after operations O_{11}, O_{21}, O_{31} have been scheduled, both action sequences O_{22}, O_{32} and O_{32}, O_{22} lead to exactly the same state S_5 shown in their Figure 2. In this particular example, this means that it is impossible to have $O_{11} \to O_{22} \to O_{32}$. In general, it is not clear whether the resulting restricted policy is still sufficiently powerful, in the sense that an optimal operation order can always be constructed.

Recently, the authors of L2D investigated an improvement heuristic for JSSP [20] 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 the 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 (CPM). 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 [21], 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.

Proposal

When the trajectory performance criterion only involves delay, there are closed-form expressions for the optimal vehicle trajectories, given their crossing times [22]. For a single intersection, this means that we can exploit the bilevel nature of the problem by formulating and solving the upper-level *crossing time scheduling problem* as a Mixed-Integer Linear Program (MILP). 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 [23] (with respect to the total delay objective). We observe that this property can be used to formulate multiple types of cutting planes to improve the running time of the solver.

When considering a network of intersections, two additional types of constraints are necessary to guarantee feasibility of the lower-level trajectory optimization, for which it can be shown that closed-form expressions are still available. First, we need to take into account the time it takes for vehicles to drive towards the next intersection on its route, giving rise to *travel constraints*. Second, the limited physical space on lanes between intersections requires us to define *buffer constraints*, which are similar in nature to the constraints used in JSSP with limited buffer capacities, see [24] for an introduction and heuristic solutions for this variant. 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, the resulting *network crossing time scheduling problem* is still tractable for reasonably sized instances. 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.

Our overall goal is to develop scalable solutions for the trajectory optimization problem in road networks sketched above. We believe that neural combinatorial optimization provides a flexible framework to design algorithms that trade optimality for speed. Therefore, we want to study a reinforcement learning construction heuristic for the (network) crossing time scheduling problem. Similar to previous approaches, we choose to define actions in terms of dispatching the next operation of an available job (vehicle), such that a complete sequence of actions corresponds to a complete ordering of jobs (vehicles) at every machine (intersection). Previous works have successfully trained partial solution encodings based on the corresponding partial disjunctive graph, which is defined by a selection of directed disjunctive arcs corresponding to the current partial ordering of operations (vehicles). Based on this partial disjunctive graph, lower bounds on the crossing times can be computed by solving a linear program with constraints corresponding to the directed disjunctive arcs, the conjunctive arcs and the arcs corresponding to buffer and travel constraints. Instead of solving the linear program each time, we propose to develop a tailored update that only propagates the necessary changes over the disjunctive graph upon adding a vehicle to the partial solution. Like previous DRL methods for JSSP, we propose to parameterize the scheduling policy by feeding the augmented disjunctive graph through a graph neural network, whose parameters can be tuned using a policy gradient learning algorithm like classical REINFORCE or the more recent Proximal Policy Optimization (PPO).

Why not model the full trajectory optimization problem as an MDP? 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.

Identified risks and their mitigation

It is not guaranteed that our ML model is not able to capture the desired policies. Two main possible issues are model capacity and training time. First, it might be the case that our model has not enough capacity (understood in statistical learning theory terms) to capture the complex dynamics required for near-optimal policies. Even if the model has enough representational power, the training procedure might not be efficient enough to obtain good policies within a reasonable amount of time. A possible reason could be it takes to long to compute the state encoding, which is based on lower bounds that can be computed from the current partial ordering of vehicles via a longest-path computation in the disjunctive graph, whose edges correspond to constraints. It is crucial that this encoding can be computed fast, because it must happen at every reinforcement learning step (assuming we do not rely on *offline reinforcement learning*). In any of these cases, it is straightforward to fallback to a simpler encoding (less capacity) that relies on simpler values, like current waiting time for every vehicle that has not yet been scheduled.

Further directions

Because this method relies on the bilevel decomposition, a correctness proof for the buffer constraints would be desirable. A sensible first step in this direction would be to derive explicit expressions for the lower-level procedure without buffer constraints. Using Pontryagin's maximum principle, it can be shown that the generated trajectories are instances of "bang-bang control", which basically means that the optimal controller switches between the two extreme control inputs, corresponding to full acceleration and deceleration in our current context.

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 [22].

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] 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.

[2] V. Rao and D. Bernstein, "Naive control of the double integrator," *IEEE Control Systems Magazine*, vol. 21, pp. 86–97, Oct. 2001.

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

[4] S. Bolusani, M. Besançon, K. Bestuzheva, A. Chmiela, J. Dionísio, T. Donkiewicz, J. van Doornmalen, L. Eifler, M. Ghannam, A. Gleixner, C. Graczyk, K. Halbig, I. Hedtke, A. Hoen, C. Hojny, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, J. Manns, G. Mexi, Erik Mühmer, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, D. Weninger, and L. Xu, "The SCIP optimization suite 9.0," technical Report, Optimization Online, Feb. 2024.

[5] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2024.

[6] 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.

[7] 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.

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

[9] P. Tallapragada and J. Cortes, "Distributed control of vehicle strings under finite-time and safety specifications," July 2017.

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

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

[12] Y. Tang, S. Agrawal, and Y. Faenza, "Reinforcement Learning for Integer Programming: Learning to Cut," July 2020.

[13] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer Networks," Jan. 2017.

[14] 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.

[15] P. Tassel, M. Gebser, and K. Schekotihin, "A Reinforcement Learning Environment For Job-Shop Scheduling," Apr. 2021.

[16] É. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, pp. 278–285, 1993.

[17] E. Demirkol, S. Mehta, and R. Uzsoy, "Benchmarks for shop scheduling problems," *European Journal of Operational Research*, vol. 109, no. 1, pp. 137–141, 1998.

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

[19] 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.

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

[21] P. Tassel, M. Gebser, and K. Schekotihin, "An End-to-End Reinforcement Learning Approach for Job-Shop Scheduling Problems Based on Constraint Programming," June 2023.

[22] 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.

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

[24] S. Heitmann, *Job-Shop Scheduling with Limited Buffer Capacities*. PhD thesis, Osnabrück University, Mar. 2007.

[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.