# Traffic Scheduling - Project Outline

Jeroen van Riel

January 2024

## 1 Introduction

Given the growing number of autonomous vehicles, an interesting question becomes how efficient traffic can be handled in a situation where every individual vehicle can be fully controlled by a single traffic manager with full information and perfect communication. Studying this theoretical question is important, because it gives us an idea of the best we can do in an idealized situation, without considering issues such as decentralized control, communication latency and partial information.

## 2 Traffic Problem

We will start by introducing a traffic scheduling problem that forms the basis of the entire project. Consider a very abstract problem in which vehicles represented as little dots are moving over a graph in a continuous fashion. Vehicles arrive to the graph over time and need to follow a fixed route towards a final node, where the vehicle leaves the system. The goal of the traffic controller is to determine how vehicles move exactly along their route, while satisfying some constraints on the interaction between vehicles that are based on safety considerations.

Let the traffic network be modeled as a weighted directed graph $G = (V, E)$, with nodes and arcs representing intersections and roads, respectively. Vehicles can travel along a series of arcs that are connected. Each node of degree one is called an *external node* and models the location where vehicles enter (*entrypoint*) or exit (*exitpoint*) the network. A node of degree at least three is called an *intersection*.

For each vehicle $j$, let its route be denoted by $R_j$, which is encoded as a sequence of nodes. Vehicle $j$ enters the network at some external node $R_j(0)$ at time $t_j^0$ and then visits the $n_j$ intersections $R_j(1), \ldots, R_j(n_j)$ until it reaches the exitpoint $R_j(n_j + 1)$, where it leaves the network. Given two routes $R_j$ and $R_l$, we define a *common path* $P = (i_1, \ldots, i_L)$ as a substring of some length $L$ of both routes (including external nodes). We refer to the first node $i_1$ of a common path as a *merge point*. The set of all common paths for vehicles $j$ and $l$ is denoted $P_{jl}$. See Figure 2 for an illustration of the above concepts.

Let $x_j(t)$ denote the position of vehicle $j$ on its route $R_j$ at time $t \geq t_j^0$. We also use the notation $x_{ij}(t)$ to denote the position on the lane on the route of $j$ leading to the $i$th intersection on its route. Furthermore, let $v_j(t)$ and $a_j(t)$ denote the speed and acceleration, respectively, of vehicle $j$ at time $t$.

The task of the traffic controller is to determine trajectories $x_j$ (or equivalently either $v_j$ or $a_j$) satisfying some kind of smoothness and safety constraints. For example, we may impose maximum/minimum speed and acceleration constraints

$$v_{\min} \leq v_j(t) \leq v_{\max}, \tag{1}$$

$$a_{\min} \leq a_j(t) \leq a_{\max}. \tag{2}$$

Given a node $i$, we will use $t_{ij} = t_{ij}(x_j)$ to denote the time when vehicle $j$ crosses $i$. Between crossing of vehicles from different lanes, we enforce a *safe lane-switching* time $s_{\text{lane}}$.

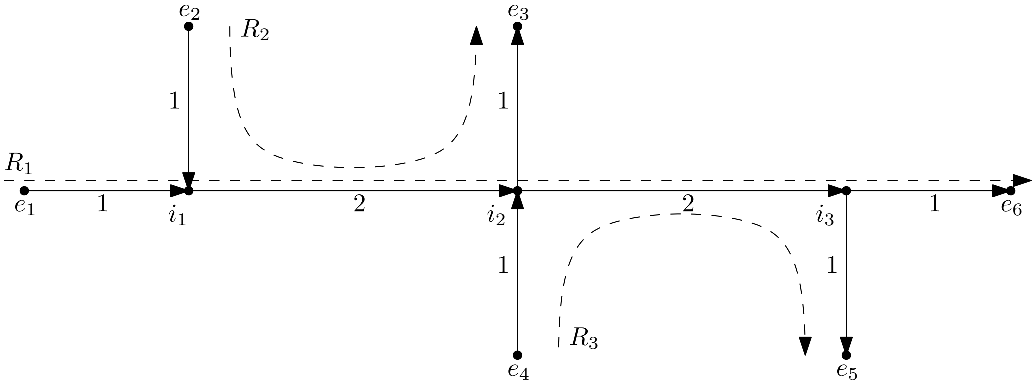

Figure 1: Example graph with three intersection $i_1, i_2, i_3$ and three vehicles, for which the routes are indicated by $R_1, R_2$ and $R_3$. The external nodes are labeled as $e_1, \dots, e_6$. The weight (travel time) of each arc is indicated next to it. The common paths are $P_{12} = \{(i_1, i_2)\}$, $P_{13} = \{(i_2, i_3)\}$ $P_{23} = \{(i_2)\}$ and the corresponding merge points are $M_{12} = \{i_1\}, M_{13} = \{i_2\}, M_{23} = \{i_2\}$.

At each merge point $i_1$ of each path $P = (i_1, \dots, i_L) \in P_{jl}$ for all pairs of distinct vehicles $\{j, l\}$, we require that either

$$t_{ij} + s_{\text{lane}} \leq t_{il} \quad \text{or} \quad t_{il} + s_{\text{lane}} \leq t_{ij}. \tag{3}$$

Furthermore, when two vehicles $j$ and $l$ are driving on the same lane we require some minimum *safe following distance* $d_{\text{follow}}$. For all pairs of distinct vehicles $\{j, l\}$, for each common path $P = (i_1, \dots, i_L) \in P_{jl}$, we require for each intersection $i \in P \setminus \{i_1\}$ that

$$x_{ij}(t) + d_{\text{follow}} \leq x_{il}(t) \quad \text{or} \quad x_{il}(t) + d_{\text{follow}} \leq x_{ij}(t), \tag{4}$$

for all $t$ in the time interval when both vehicles are driving on the common path. When we do not allow vehicles to overtake each other, exactly one of the inequalities must hold for the entire common path. For simplicity, this second type of constraints was stated in terms of distance, but note that it could be equivalently stated in terms of time as well.

## 2.1 Optimization setting

Let us now consider more precisely how the controller interacts with the system defined above. In general, at some time $t$, we assume that the controller knows the arrival times of

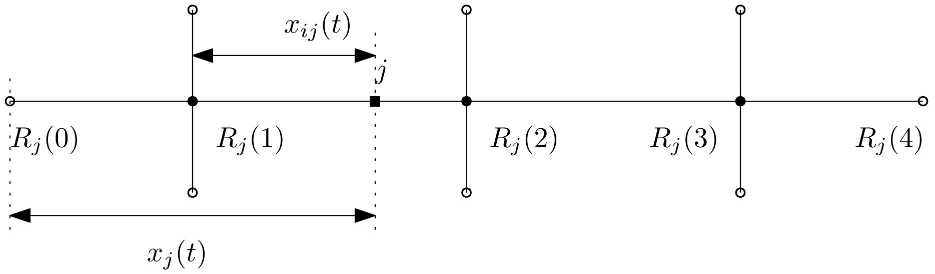

Figure 2: Illustration of a network of intersections with the position of some vehicle $j$ given in two ways.

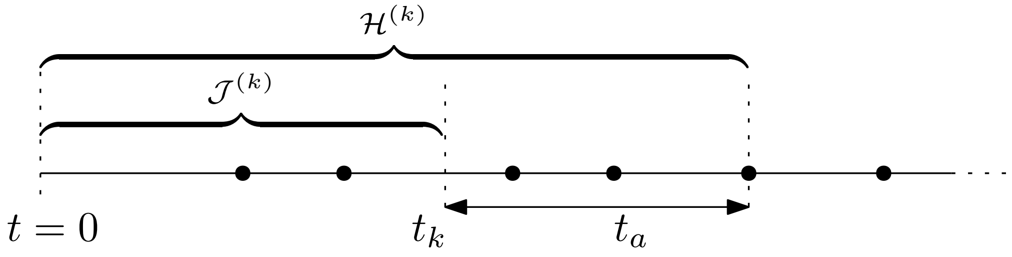

Figure 3: Illustration of some decision epoch $t_k$ and the corresponding set of vehicles under control and the visible horizon. The dots represent vehicle arrival times $t_j^0$.

vehicles

$$\mathcal{H}_t = \{j : t_j^0 \leq t + t_\mathrm{a}\}, \tag{5}$$

for some non-negative look-ahead time $t_\mathrm{a}$. We call this set of vehicles (with their corresponding arrival times) the *visible horizon* of the controller. Similarly, let the set of vehicles in the system at time $t$ be denoted by

$$\mathcal{J}_t = \{j : t_j^0 \leq t\}. \tag{6}$$

Ignoring for now that vehicles eventually leave the system, we say that $\mathcal{J}_t$ are precisely the vehicles *under control* of the traffic controller.

Note that the only uncertainty in the system stems from the random arrival times of vehicles. Once trajectories are determined, the dynamics of the system are completely deterministic as long as $\mathcal{H}_t$ does not change. Therefore, the only sensible decision epochs to consider are the points in time $t_k$ whenever $\mathcal{H}_t$ changes, i.e., when new information becomes available, see Figure 3 for an illustration. Whenever this happens, the controller needs to compute a trajectory for the newly arrived vehicles, and is free to revise the trajectories of the vehicles that are already in the system. However, these updated trajectories need to respect the trajectories that have been followed up till that point.

Let us make the sequential decision process sketched above a little bit more precise. To simplify notation, we define $\mathcal{J}^{(k)} = \mathcal{J}_{t_k}$ and $\mathcal{H}^{(k)} = \mathcal{H}_{t_k}$. Let the *state* of the system at time $t_k$ be represented by the sets

$$s_k = (\{(t_j^0, R_j)\}_{j \in \mathcal{H}^{(k)}},\ \{(x_j^{(k-1)}(t_k), v_j^{(k-1)}(t_k), a_j^{(k-1)}(t_k))\}_{j \in \mathcal{J}^{(k)}}). \tag{7}$$

The controller needs to compute a set of trajectories

$$a_t = \{x_j^{(k)}\}_{j \in \mathcal{H}^{(k)}}, \tag{8}$$

which are functions of $t \geq t_j^0$, satisfying the constraints from the previous section and the additional constraints

$$x_j^{(k)}(t_k) = x_j^{(k-1)}(t_k), \tag{9a}$$

$$v_j^{(k)}(t_k) = v_j^{(k-1)}(t_k), \tag{9b}$$

$$a_j^{(k)}(t_k) = a_j^{(k-1)}(t_k), \tag{9c}$$

for the vehicles $j \in \mathcal{J}^{(k)}$ that were already under control, guaranteeing in some sense the *continuation* of the updated trajectories.

From the above discussion, it should be clear that the transition to the next state $s_{k+1}$ is particularly simple. The arrivals that happen at $t_{k+1} + t_a$ are simply added to the visible horizon and the positions, speeds and accelerations follow directly from the determined trajectories $a_t$.

We still have to define an objective to compare the performance of different controllers. Measures of how well the controller performs on some particular sequence of arriving vehicles may be based on properties of the computed trajectories. For example, we could relate some measure of energy consumption to acceleration, or we can measure delay based on the crossing times $t_{ij}$. In general, the optimization objective is the expected value of this measure, where the expectation is taken over the distribution of the vehicle arrival process.

Finally, we may distinguish between a finite-horizon problem with a finite number of vehicles, or an infinite-horizon case, in which particular attention needs to be paid to the definition of the optimizaton objective, e.g., we might need to use discounting to avoid infinite sums.

# 3 Analysis

Before discussing concrete models for solving the general traffic scheduling problem, let us discuss two important aspects that will be relevant throughout the analysis of all the models and the corresponding reinforcement learning formulations.

First of all, we need to make some assumptions on how vehicles arrive to the network. We assume that each vehicle $j$ has an earliest arrival time $r_j$. The controller decides the exact arrival time $t_j^0 \geq r_j$ when the vehicle actually enters the network. By having this assumption, we make sure that the problem is always feasible, because vehicles can be kept waiting at the entrypoints for as long as necessary. Would we assumed that vehicle are forced to enter the network at their arrival time $r_j$, it is not difficult to see there are instances in which no trajectories satisfy the safety constraints.

An interesting aspect of any learned policy is its generalization to similar problems. It would be interesting to study generalization across different network topologies, starting with trying to formulate a policy that is agnostic to the network topology. More straightforward would be the study of generalization across different arrival distributions. It is well known that road traffic demand fluctuates highly throughout the day. Therefore, a policy that keeps good performance under increased arrival rates is desirable. An interesting next step would be to see if an explicit memory mechanism could improve the ability to adapt to changing arrival rates.

# 4 Research Plan

In order to compute trajectories for the general traffic control problem sketched above, we propose some models that are all based on some kind of discretization. This allows us to formulate a policy space for which we can use reinforcement learning techniques for finding good policies. The following subsections will shortly introduce the three main kinds of models that we want to consider, ordered by increasing complexity.

## 4.1 Single Intersection

A trajectory generating approach for a single intersection has been proposed in [1]. The main idea is to first schedule the times at which vehicles cross the intersection and then generate feasible trajectories accordingly. When the objective only measures delay at the intersection, this shows that the single intersection problem reduces to scheduling crossing times at the intersection. More precisely, in machine scheduling terminology, we are dealing with a single machine scheduling problem with release dates, job families with chain precedence constraints (corresponding to lanes), family-dependent setup times and total completion time objective. Furthermore, the platoon preservation theorem from [2] lowers the complexity of the problem even further. Under the assumption that all future arrivals are known, the problem can be solved to optimality by formulating it as a Mixed-Integer Linear Program (MILP). Nevertheless, it makes sense to study heuristic method that would allow us to tackle larger instances.

When we drop the assumption of complete knowledge of the future, we are in the setting of online scheduling [3], in which vehicle arrival times become gradually known to the controller over time. There are multiple ways to assess the performance of an online scheduler, but the most straightforward way is to study the competitive ratio compared with a so-called adversary. In case of a deterministic algorithm, the adversary constructs a problem instance based on the description of the algorithm and then solves it optimally. The worst-case ratio between the objective value of the algorithm and the objective of the optimal solution is called the competitive ratio. Following a similar approach as in the proof of Theorem 2.1 in [3], we might be able to show a lower bound on the competitive ratio. After some quick calculations, we found that (for some particular problem parameters) this ratio is strictly larger than 1, which roughly means that there is some value in knowing the future. However, the expressions that we obtained do not allow a clean analysis, so obtaining the lower bound might require some numerical approximation.

Our goal is to study heuristics for the offline and online single intersection scheduling problem. A common approach in the scheduling literature is to manually formulate heuristics based on our understanding of the problem structure. Instead, we will be using reinforcement learning techniques to automatically learn heuristics. In order to apply reinforcement lenaring techniques, we first need to precisely specify the interaction between the controller and the environment, which is done by formulating a Markov Decision Process. We will study an MDP that is based on the idea of *dispatching*, which is an idea from scheduling theory in which a schedule is constructed by iteratively choosing the next vehicle that is allowed to use a shared resource. In this way, the learned policy is applicable in both the offline and online problem setting.

We have a precise definition of the single intersection scheduling problem. Furthermore, we formulated a MILP and have some code to solve arbitrary instances exactly using the commerical Gurobi solver [10]. We have implemented the dispatching MDP as a Gymnasium [4] environment and we have setup a basic DQN [5] agent based on the off-the-shelf implementation of CleanRL [6]. What remains to be done is a thorough analysis of the learned policies for different distributions of the vehicle arrival process. Part of this analysis will be a comparison to the exact solutions found via the MILP solver.

## 4.2 Network

Next, we consider multiple intersections in a network. When we neglect the dynamics of vehicles on lanes between intersections, a very natural formulation of the scheduling problem is given by a job shop [7] scheduling problem with some additional constraints to encode the time that a vehicle needs to travel to the next intersection on its route. In theory, it is still possible to solve the offline variant exactly using a MILP solver. However, we expect that this approach does not scale well in terms of the network size and number of vehicles. Therefore, in order to solve real-world problems, we want to focus on finding good heuristics.

A natural extension of the dispatching approach that we used in the single intersection case would be based on dispatching at every intersection. Like we did for the single intersection case, this method could be compared to exact solutions, but we expect that the scale of the problem will be prohibitive here.

Some recent works [8, 9] have investigated reinforcement learning method for solving job shop problems and showed some promising results. Therefore, we would like to investigate how to use these methods to construct an efficient reinforcement learning formulation for the offline variant. The method from [8] uses the idea of a disjunctive graph corresponding to the jobs shop problem. Determining the order of vehicles in the schedule is equivalent to determining the directions of a certain set of arcs in this disjunctive graph. A reinforcement learning controller can be learned to iteratively direct these arcs based on an graph neural network embedding of the current disjunctive graph.

The model from [1] together with the platoon preservation theorem [2] imply that it is never beneficial to split platoons when there is only a single intersection. However, for more than one intersections, we have found examples where platoon splitting is necessary in the optimal schedule. It would be interesting to understand better why this happens and characterize somehow these kind of situations.

## 4.3 Finite buffers

In the models discussed above, we assumed that vehicles do not interact on lanes, hence it is possible for an unbounded number of vehicles to wait on a lane at any given time. However, in real-world traffic networks, the allowed number of vehicles is limited by space constraints and even depends non-trivally on the speeds of individual vehicles. Therefore, we propose a model that incorporates this aspect by considering the position of vehicles on lanes. Instead of encoding the precise location in a continuous sense, we divide each lane in a finite number of locations. For a given vehicle, the controller has to decide how long it should wait on each location. For large number of locations per lane, this method allows us to approximate continuous trajectories.

In order to illustrate this kind of model, we will make the case with one intersection a little bit more formal. Assume that it takes constant time $\Delta t$ for a vehicle to travel to the next location. For each lane $k$, let $m_k$ denote the number of locations, excluding the entrypoint. Let the locations of lane $k$ be identified by increasing numbers $\mathcal{L}(k) = (1, \dots, m_k)$, where the last one corresponds to the final intersection. In the following variable definitions, we will not explicitly mention the dependency on the lane to keep notation simple. Let $y_{ij}$ denote the time at which vehicle $j$ departs from location $i \in \{0, \dots, m_k\}$, then $y_{ij} + \Delta t$ is the arrival time of vehicle $j$ at location $i + 1$. To simplify notation, we define $\bar{y}_{ij} = y_{i-1,j} + \Delta t$ to be the arrival time of vehicle $j$ at location $i$. For every location $i$ and vehicle $j$, we require

$$\bar{y}_{ij} \leq y_{ij}. \tag{10}$$

For each pair of consecutive vehicles on the same lane $k$ with precedence constraint $j \to l \in \mathcal{C}_k$, we have the inequalities

$$y_{ij} + p \leq \bar{y}_{il}, \tag{11}$$

for every location $i$ along their route. Furthermore, we assume that for every vehicle $j$, the initial departure time from the entrypoint of vehicle $j$ satisfies

$$t_j^0 \leq y_{0j} \tag{12}$$

in order to model the *arrival time* of the vehicle.

Finally, we need to model the safety constraints involving vehicles from different lanes that cross the intersection. Let $j$ be some vehicle on lane $k(j)$, then let let $y_j$ denote the departure time from the intersection, so we have $y_j = y_{ij}$ with $i = m_{k(j)}$. Similarly, let $\bar{y}_j$ denote the arrival time of $j$ at the intersection, so we have $\bar{y}_j = y_{i-1,j} + \Delta t$ with $i = m_{k(j)}$. From constraints (11), we see that it makes sense to say that vehicle $j$ occupies the intersection during the interval

$$[\bar{y}_j, y_j + p]. \tag{13}$$

We require an additional *switch-over time* $s$ whenever the next vehicle to occupy the intersection comes from a different lane. Let

$$\mathcal{D} = \{\{j, l\} : j \in F_{k_1}, l \in F_{k_2}, k_1 \neq k_2\}, \tag{14}$$

denote the set of *conflict* pairs. The additional constraints are

$$y_j + p + s \leq \bar{y}_l \quad \text{or} \quad y_l + p + s \leq \bar{y}_j, \tag{15}$$

for all $j, l \in \mathcal{D}$.

When the objective only measures delay at the intersection, the case of a single intersection reduces to the single intersection scheduling problem with infinite lane capacity, because the entrypoints still have infinite capacity. Therefore, a vehicle can wait as long as necessary at the entrypoint, before entering the lane.

Once we consider more than one intersection, the controller needs to start taking into account the effects of finite lane capacity. Because of the discrete nature of the model, it is straightforward to formulate it as a MILP, which allows us to solve the problem exactly. However, as the number of variables grows rapidly with the values of $m_k$, we do not expect this approach to scale well.

We have started with proving that the single intersection case reduces to the problem with infinite lane capacity. Furthermore, we have a MILP formulation that we can solve with Gurobi, which we could use to verify this reduction claim. We still need to make precise in what way our model approximates the generation of continuous trajectories. Based on a suitable definition, we should then prove that a certain error measure can be made arbitrily small by choosing values for $m_k$ large enough.

Once we are done with the theoretical investigation sketched above, we should be ready to start experimenting with reinforcement learning controllers. A major question is how to structure the policy for determining location delays for vehicles in a online setting. It might seem straightforward to have the controller determine the location delays for a fixed number of upstream locations, but it is not immediately clear how to define this for a variable number of vehicles. One approach would be to have the controller set a location delay for each location. Each time a vehicle enters a location, it will stay there for the set location delay. This way, we have shifted the focus to the infrastructure, similarly like we did in the dispatching environment for the single intersection scheduling problem.

# References

[1] 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.

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

[3] A. Vestjens, *On-Line Machine Scheduling.* Phd Thesis 1 (Research TU/e / Graduation TU/e), Technische Universiteit Eindhoven, Eindhoven, 1997.

[4] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. de Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, “Gymnasium,” Mar. 2023.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[6] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, “CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms,” *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022.

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

[8] 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.

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

[10] Gurobi Optimization, LLC, “Gurobi optimizer reference manual,” 2024.