Autonomous vehicle scheduling in networks

Jeroen van Riel

August 2025

1 Platoons in tandem of two intersections

When considering multiple vehicles driving between intersection, we must take into account the capacity of the lane segment between them, because the fact that only a limited number of vehicles can drive or wait at the same time on a lane between intersections may cause network-wide effects. The capacity of lanes between intersections is intimately related to the trajectories of vehicles, which we first want to understand better. We will be using an optimal control formulation with the objective that keeps the vehicles as close as possible to the next intersection at all times, which is similar to the MotionSynthesize problem considered in [1]. This problem can be solved using direct transcription, which works well enough if we just want to simulate the behavior of the system. However, we show that it is possible to explicitly formulate the optimal controller and explain how to compute trajectories without using time discretization.

1.1 Problem formulation

Before we turn to the general case of networks of intersection, we will first investigate the trajectories of vehicles in a tandem of two intersections as depicted in Figure 1. Let v denote the left intersection and w the right intersection and assume that vehicles drive from left to right. Furthermore, we will call the road segment strictly between both intersection areas the *lane*. In the following discussion, let $p(t)$ and $v(t)$ denote the position and velocity of the vehicle, respectively, and we call $x(t) = (p(t), v(t))$ its *state*. Let the length and width of a vehicle i be denoted by L_i and W_i, respectively. We measure the position of a vehicle at the front bumper. We fix position $p = 0$ at the stop line of intersection w. We make the following additional assumptions about our lane model.

Assumption 1. *Vehicles are represented as rectangles, all having the same length $L_i = L$ and width $W_i = W$. Lanes are axis-aligned and have width W, such that when lanes intersect, the intersection area is a square. Vehicles are not able to overtake other vehicle.*

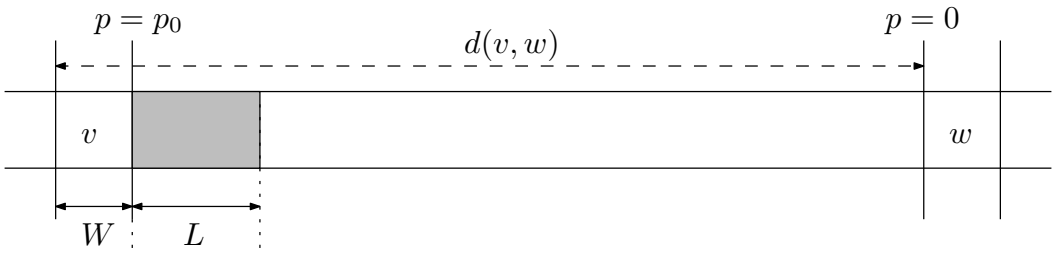

Figure 1: Tandem of two intersections v and w with lane of length $d(v, w)$. The grey rectangle represents some vehicle that just left intersection v. We assume that vehicles must drive at maximum speed as long as they occupy any intersection, so deceleration is only allowed from the shown position onwards.

Assumption 2. *All vehicles satisfy the same double integrator dynamics $\dot{p} = v$, $\ddot{p} = u$ with velocity bounds $0 \le v \le v_{\max}$ and symmetric control bounds $-\omega \le u \le \omega$. We assume that the maximum speed is $v_{\max} = 1$, which is without loss of generality, because we can always achieve this by appropriate scaling of positions and the acceleration bound ω.*

Assumption 3. *Vehicles drive at full speed when entering an intersection and keep driving at full speed as long as they occupy an intersection.*

Now assume that some vehicle is scheduled to start entering the lane at time $t_0 < 0$ and has to start entering w at time $t = 0$. Let $p_0 = -d(v, w) + W$ denote the position from where the vehicle starts to enter the lane. We will refer to the vehicle driving in front of the current vehicle as the *lead vehicle*. Let $\bar{p}(t)$ denote the position of rear bumper of the lead vehicle, assuming there is one. To avoid collision with this vehicle, the current vehicle must satisfy the *lead* constraint $p(t) \le \bar{p}(t)$ at all times. We try to keep the vehicle as close to w as possible at all times. This yields the optimal control problem

$$\begin{aligned} \max_{u} \quad & \int_{t_0}^{0} p(t)dt \\ \text{s.t.} \quad & \dot{p} = v,\ \ddot{p} = u, \\ & p(t_0) = p_0, \quad p(0) = 0, \\ & v(t_0) = 1, \quad\ \ v(0) = 1, \\ & -\omega \le u(t) \le \omega, \\ & 0 \le v(t) \le 1, \\ & p(t) \le\ \bar{p}(t). \end{aligned} \tag{1}$$

1.2 Optimal control

The aim of this section is to provide an explicit parameterization of optimal solutions of problem (1). Before we proceed, we first rewrite it to the following standard form of optimal control problems

$$\begin{aligned} \max \quad & \int_{t=t_0}^{t_f} F(x(t), u(t), t)dt \\ \text{s.t.} \quad & \dot{x}(t) = f(x(t), u(t), t), \quad x(t_0) = x_0, \\ & a(x(t_f), t_f) \ge 0, \\ & b(x(t_f), t_f) = 0, \\ & g(x(t), u(t), t) \ge 0, \\ & h(x(t), t) \ge 0, \end{aligned} \tag{2}$$

where x is the vector of state variables and u is the control input. Here, the constraints g are called mixed state constraints, because they involve both state and control variables, while constraints h are called pure state constraints. In this subsection, we will write the state as $x = (x_1, x_2)$ with position x_1 and velocity x_2. The control function u again corresponds to acceleration. The initial state is (p_0, v_0) and the target state is $(0, 1)$. We write v_0, because we will be considering optimal trajectories with $v_0 < 1$ as an intermediate step of the analysis. Let $\bar{x}$ denote the state trajectory of the rear bumper of the lead vehicle. With this new

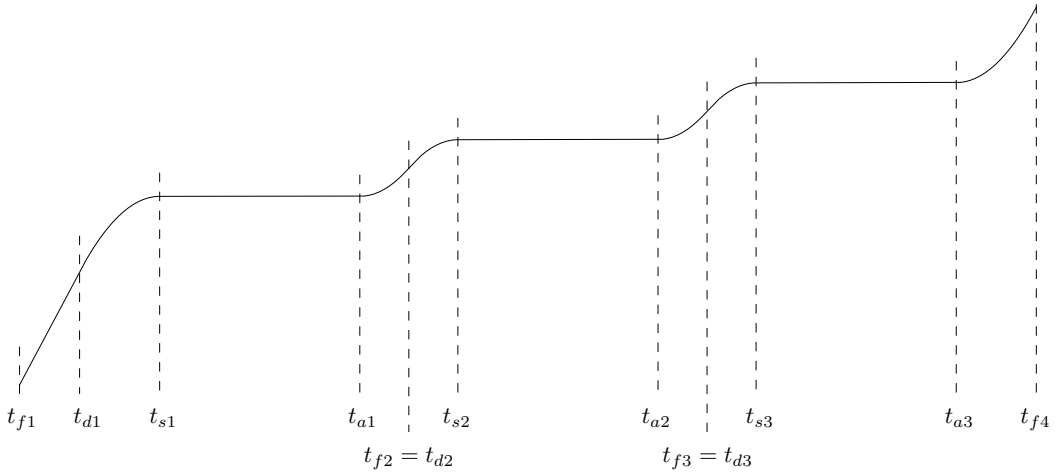

Figure 2: Some example of an alternating vehicle trajectory x. The particular shape of this trajectory is due to two preceding vehicles, which causes the two "bumps" at the times where these vehicles exit the lane. The trajectory of the current lead vehicle and previous lead vehicles that gave rise to this particular shape are not shown for clarity.

notation, optimal control problem (1) is equivalent to (2) for $v_0 = 1$ and setting[1]

$$\text{(2a)} \quad \left\{ \begin{aligned} t_f &= 0, \\ F(x,u,t) &= x_1, \\ f(x,u,t) &= (x_2, u), \\ x_0 &= (p_0, v_0), \\ b(x,t) &= (x_1, x_2 - 1), \\ g(x,u,t) &= \omega^2 - u^2, \\ h_1(x,t) &= x_2 - x_2^2, \\ h_2(x,t) &= \bar{x}_1(t) - x_1. \end{aligned} \right\} \quad \text{(2b)}$$

We refer to problem (2a) as the *single vehicle variant*, because ignoring the lead constraint is equivalent to considering a single vehicle in the system. For both problem variants, we will denote an instance using the tuple $z = (t_0, p_0, v_0)$.

In the upcoming analysis, we will encounter control functions that switch between no acceleration, full deceleration and full acceleration, to which we might refer as *bang-off-bang* control. Furthermore, we will see that optimal solutions have a particularly simple structure, illustrated in Figure 2, to which we refer as *alternating control*. The following definition makes this notion precise and introduces the notation that we use to completely characterize optimal trajectories.

1.3 Feasibility of crossing times

We now investigate the limit on the number of vehicles that can occupy the lane when waiting. Imagine that vehicles enter the lane until it is full and then only after all vehicles have come to a full stop, they start to leave the lane by crossing w. We refer to the maximum number of vehicles that can wait in the lane as the *stationary lane capacity*. Suppose that we want to design the tandem network that has a stationary lane capacity of at least $c(v, w)$. Vehicles are

[1] Instead of using quadratic expressions, constraints g and h_1 could have each been written as two linear constraints, e.g., we could have chosen $g(x,u,t) = (\omega - u, \omega + u)$. However, this would violate the constraint qualification conditions of the theorem that we use in Section **??**.

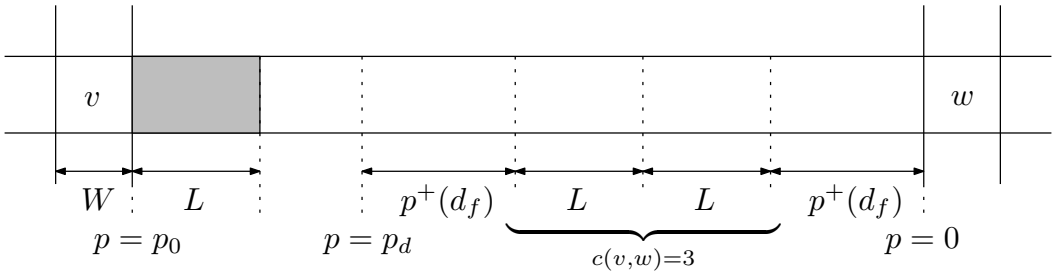

Figure 3: Tandem of intersections with indicated distances used in the derivation of the stationary lane capacity, which is the maximum number of vehicles that can stop and wait in the lane, before they leave.

required to drive at full speed as long as they occupy any intersection. Therefore, a vehicle crossing v can only start decelerating after $p(t) \geq p_0 + L$, so the earliest position where a vehicle can come to a stop is $p_0 + L + p^+(d_f)$. Because vehicles need to gain maximum speed before reaching w, the position closest to w where a vehicle can wait is $-p^+(d_f)$. Hence, in order to accomodate for $c(v, w)$ waiting vehicles, the length of the lane must satisfy

$$d(v, w) \geq W + L + 2p^+(d_f) + (c(v, w) - 1)L,$$

as illustrated in Figure 3. Conversely, given the lane length $d(v, w)$, the corresponding stationary lane capacity is given by[2]

$$c(v, w) = \texttt{floor}\left(\frac{d(v, w) - W - 2p^+(d_f)}{L}\right),$$

where $\texttt{floor}(x)$ denotes the largest integer smaller than or equal to x.

It turns out that the fixed locations where vehicles wait in the above scenario are helpful in describing the optimal trajectories, even when vehicles never fully stop. We will denote these fixed *waiting positions* as

$$p_k = -p^*(d_f) - (c(v, w) - k)L,$$

for $k = 1, \ldots, c(v, w)$. Furthermore, let $p_d = p_1 - p^*(d_f)$ denote the position from which vehicles must decelerate in order to stop at the first waiting position p_1. Now consider a vehicle that moves from p_k to the next waiting position p_{k+1}, so it moves exactly distance L. We consider such a start-stop movement, without considering any safe following constraints. By symmetry of the control constraints, the vehicle moves the same distance during both acceleration and deceleration. Furthermore, the vehicle needs to be at rest at the start and end of such trajectory. Hence, it is clear that it takes the same amount of time d_s to accelerate and decelerate. We assume that $d_s < d_f$, which ensures that maximum velocity is never reached during the start-stop movement, which is illustrated in Figure **??**. In this case, it is clear that we must have $L = 2p^*(d_s)$, from which we derive that $d_s = \sqrt{L/a_{\max}}$.

[2]Without Assumption 1, we cannot derive such a simple formula, because it would depend on the specific lengths of those vehicles currently in the system.

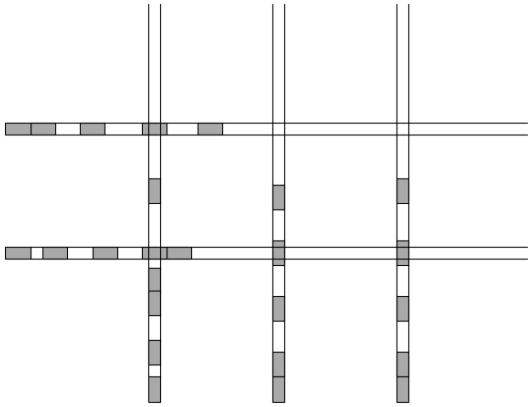

Figure 4: Illustration of some grid-like network of intersections with vehicles drawn as grey rectangles. There are five vehicle routes: two from east to west and three from south to north. Turning at intersections is not allowed.

2 Vehicle scheduling in networks

We now extend the single intersection model to a network of intersections without turning routes, illustrated in Figure 4. We define a directed graph $(\bar{V}, E)$ with nodes $\bar{V}$ and arcs E, representing the possible paths that vehicles can follow. Nodes with only outgoing arcs are *entrypoints* and nodes with only incoming arcs are *exitpoints*. Let V be the set of *intersections*, which are all the nodes with in-degree at least two. Let $d(v, w)$ denote the distance between nodes v and w. For each route index $r \in \mathcal{R}$, we let

$$\bar{V}_r = (v_r(0), v_r(1), \ldots, v_r(m_r), v_r(m_r + 1))$$

be the path that vehicles $i \in \mathcal{N}_r$ follow through the network. We require that the first node $v_r(0)$ is an entrypoint and that the last node $v_r(m_r + 1)$ is an exitpoint and we write

$$V_r = \bar{V}_r \setminus \{v_r(0),\, v_r(m_r + 1)\}$$

to denote the path restricted to intersections. We say that some $(v, w) \in E$ is on path V_r whenever v and w are two consecutive nodes on the path and we write E_r to denote the set of all these edges. We require that routes can only overlap at nodes by making the following assumption.

Assumption 4. *Every arc $(v, w) \in E$ is part of at most one route V_r, such that routes do not share lanes. This ensures that the order of vehicles on each lane is completely determined by the order of vehicles on the corresponding lane.*

We start by considering networks in which all roads are axis-aligned such that intersections always involve perpendicular lanes and where routes are such that no turning is required. For each $v \in V_r$ define the conflict zone $\mathcal{E}_r(v) = (b_r(v), e_r(v))$ and consider the union

$$\mathcal{E}_r = \bigcup_{v \in V_r} \mathcal{E}_r(v)$$

corresponding to the positions of vehicles $i \in \mathcal{N}_r$ for which it occupies an intersection on its path V_r. By reading $\mathcal{E}_i \equiv \mathcal{E}_r$ for $r(i) = r$, the single intersection problem naturally extends to the network case. Like before, the resulting problem can be numerically solved by a direct transcription method.

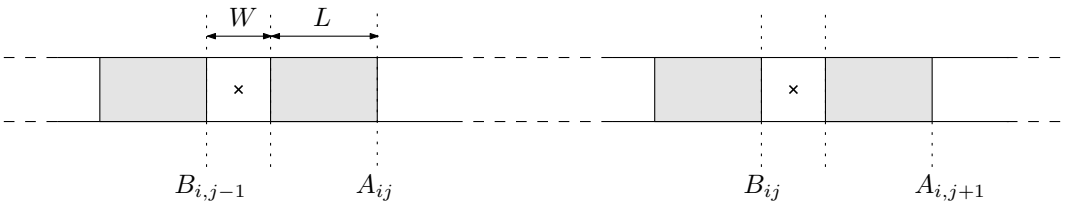

Figure 5: Illustration of how the individual lane models are connected to form a route and how to model an intersection. The intersection region is marked with a little cross. The width of the vehicle W is used as the length of the intersection. The longitudinal positions A_{ij} and B_{ij} denote the start and end, respectively, of the jth lane on route i.

2.1 General decomposition

The general two-stage decomposition for the single intersection extends rather naturally to the present model. Let for each pair (i, v) of some vehicle $i \in \mathcal{N}$ and an intersection $v \in V_{r(i)}$ along its route, let

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

be the crossing time and exit time, which we denote by $y(i, v)$ and $y(i, v)+\sigma(i, v)$, respectively. Instead of a single set of conflicts, we now define for each intersection $v \in V$ in the network the set of conflict pairs

$$\mathcal{D}^v = \{\{i, j\} \subset \mathcal{N} : r(i) \neq r(j), v \in V_{r(i)} \cap V_{r(j)}\}.$$

Now the two-stage approach is to solve

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

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

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

where we again use subscript r to group variables according to their associated route.

2.2 Decomposition for delay objective

Suppose we use use the crossing at the last intersection as performance measure, by defining the objective function as

$$J(x_i) = \inf\{t : x_i(t) \in \mathcal{E}_r(v_r(m_r))\}.$$

We show how to reduce the resulting problem to a scheduling problem, like we did in the single intersection case. We will again assume Assumption 1 and Assumption 3, so vehicles

will always cross intersections at full speed, and all vehicles share the same geometry. Hence, the occupation time $\sigma \equiv \sigma(i,v)$ is the same for all vehicles and intersections. For this reason, we will write the shorthand $y_r \in \mathcal{S}_r$, because σ_r is no longer a free variable.

As a consequence of Assumption 1 and Assumption 3, each lower-level trajectory optimization problem for a given route $r \in \mathcal{R}$ decomposes into a sequence of problems, each corresponding to two consecutive intersection along V_r. This means that $y_r \in \mathcal{S}_r$ is equivalent to $y_{(v,w)} \in \mathcal{S}_{(v,w)}$ for each $(v,w) \in E_r$, where $y_{(v,w)}$ denotes the vector of all variables $y(i,v)$ and $y(i,w)$ for all $i \in \mathcal{N}_r$ and $\mathcal{S}_{(v,w)}$ denotes the set of values of $y_{(v,w)}$ for which a feasible trajectory part can be found. Hence, we will now focus on a tandem of two intersections and investigate the trajectories of vehicles in this with the goal of stating sufficient conditions for $y_{(v,w)} \in \mathcal{S}_{(v,w)}$.

2.3 Crossing time scheduling

Analogously to the single intersection case, we let the earliest crossing time $\beta_t(i,v)$ for vehicle $i = (r,k) \in \mathcal{N}$ at network nodes $v \in \bar{V}_r$ in current disjunctive graph G_t be recursively defined through

$$\beta_t(i,v) = \begin{cases} a(i,v) & \text{if } v \text{ is an entrypoint,} \\ \max_{i \in \mathcal{N}_t^-(j)} \beta_t(i,v) + w(i,j) & \text{otherwise,} \end{cases}$$

where $\mathcal{N}_t^-(j)$ is again the set of in-neighbors of node j in G_t. For empty schedules, it is easily seen that we have $\beta_0(i, v_r(0)) = a(i, v_r(0))$ for entrypoints and we have $\beta_0(i, v_r(l+1)) = \beta_0(i, v_r(l)) + d(v_r(l), v_r(l+1))/v_{\max}$ for $l = 1, \ldots, m_r$, so between consecutive intersections on the same route.

There are two natural choices for the objective to optimize in this scheduling setting. We minimize the time each vehicle is in the system, which is equivalent to minimizing the delay at the last intersection of each vehicle's route, written as

$$\text{obj}_1(y) = \sum_{i \in \mathcal{N}} y(i, v_r(m_r)) - \beta_0(i, v_r(m_r)).$$

Alternatively, it also makes sense to minimize the delay at every intersection along each vehicle's route, so we can also minimize

$$\text{obj}_2(y) = \sum_{i \in \mathcal{N}} \sum_{v \in V_{r(i)}} y(i,v) - \beta_0(i,v).$$

3 Learning to schedule

Like in the single intersection case, we try to model optimal solutions using autoregressive models. In the single intersection case, we argued that each schedule is uniquely defined by its route order η. Generalizing this to the case of multiple intersections, we see that a schedule is uniquely defined by the set of route orders η^v at each intersection $v \in V$. Instead of working with such a set of sequences, we will intertwine these sequences to obtain a single *crossing sequence* η, which consists of pairs (r, v) of a route $r \in \mathcal{R}$ at some intersection $v \in V_r$ and we will refer to such a pair as a *crossing*. Of course, this crossing sequence can be constructed in many different ways, because it does not matter in which order the intersections are considered, which is illustrated in Figure 6. Given some problem instance s, we will consider autoregressive models of the form

$$p(\eta|s) = \prod_{t=1}^{N} p(\eta_t|s, \eta_{1:t-1}). \tag{3}$$

These models can also be understood in terms of a step-by-step schedule construction process that transitions from a partial schedule state $s_t = (s, \eta_{1:t})$ to the next state s_{t+1} by selecting some *action* $\eta_{t+1} = (r_{t+1}, v_{t+1})$. We say a crossing is pending when it still has unscheduled vehicles. Similarly, we say an intersection is pending when some of its crossings are still pending. With these definitions, the set of *valid actions* $\mathcal{A}(s_t)$ at some intermediate state s_t is exactly the set of pending crossings. We again emphasize that multiple sequences of actions lead to the same schedule, because the order in which intersections are considered does not matter for the final schedule, which is illustrated in Figure 6. The models that we study can be understood as being parameterized as some function of the disjunctive graph G_t of partial schedule s_t, so an alternative way of writing (3) that emphasizes this is

$$p(\eta = ((r_1, v_1), \ldots, (r_N, v_N)) \mid s) = \prod_{t=1}^{N} p(r_t, v_t|G_{t-1}). \tag{4}$$

Instead of modeling the joint probability distribution $p(r_t, v_t|G_{t-1})$ over crossings, we can also apply the chain rule to factorize it as

$$p(r_t, v_t|G_{t-1}) = p(r_t|v_t, G_{t-1})p(v_t|G_{t-1}). \tag{5}$$

In this case, the model $p(r_t|v_t, G_{t-1})$ can be thought of as predicting a set of actions, one for each intersection.

Intersection visit order. In the general class of autoregressive models for inputs and outputs with sets, of which our model above is a special case (we have a set of sequences as output), it has been noted before that the order in which inputs or outputs are presented to the model during training has a considerable impact on the final model fit [4]. For our model, we also expect to find this effect, so we will investigate the impact of the order in which intersections are visited, determined by $p(v_t|G_{t-1})$. We now propose some heuristic ways to define $p(v_t|G_{t-1})$. Later, we will consider a neural network parameterization. First of all, the simple *random* strategy would be to sample some intersection with pending crossings at each step. In the *boundary* strategy, we keep visiting the same intersection until it is done (when it has no pending crossings anymore), then move to some next intersection. When the network of intersections is free of cycles, we could for example follow some topological order. We use the term "boundary" because this strategy produces trajectories along the boundary of the grid in Figure 6. In the *alternating* strategy, we keep alternating between intersection to keep the number of scheduled vehicles balanced among them. This produces trajectories that can be understood as being close to the "diagonal" of the grid in Figure 6. Again, the order in which we alternate between intersections may again be based on some topological order.

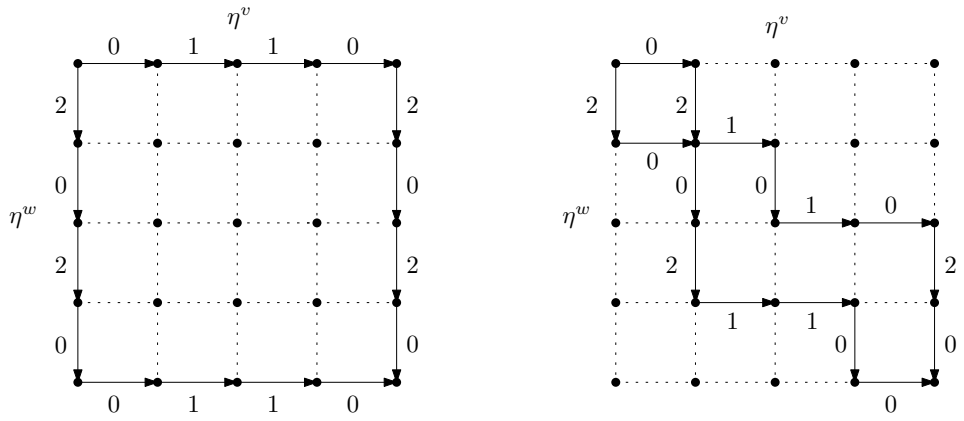

Figure 6: When each route order is locally fixed at every intersection, the global crossing sequence is not uniquely determined, because these local sequences may be merged in any order. Suppose we have a tandem of two intersections and a horizontal arrows correspond to taking the next local action at intersection v, a vertical arrows correspond to taking next local action at intersection w. Each valid crossing sequence corresponds to some path from the top-left to the bottom-right corner. Although any such crossing sequence produces the same schedule, it might be that our autoregressive model fits better on some sequences than others. For example, we might expect that sequences on the boundary of the grid, shown in the left grid, are harder to learn from data than sequences that stay closer to the diagonal, like in the right grid. The intuition is that we need to "look into the future" more to learn the former, while in the latter trajectories, progress in the two intersections is more balanced.

3.1 Model parameterization

3.1.1 Threshold heuristics

It is straightforward to extend the threshold rule to networks of intersections, when assuming a fixed intersection order. Each time some next intersection is visited, we apply the single intersection threshold rule to pick the next route. This is straightforward to do, because we can just consider the disjunctive subgraph induced by the nodes belonging to that intersection to arrive at the single intersection case. Furthermore, the definition of the threshold rule itself does not depend on the network of intersections. This is a desirable property, because it allows us to tune the threshold on small networks and then apply it on larger ones.

3.1.2 Neural constructive heuristic

We will now propose a neural network parameterization of $p(r_t|v_t, G_{t-1})$ and train it based on optimal schedules in a supervised learning setting. The model can be best understood as solving a so-called multi-label classification problem, because it needs to provide a distribution over routes at every intersection. The training data set $\mathcal{X}$ consists of pairs $(G_{t-1}, (r_t, v_t))$, to which we refer to as *state-action* pairs to draw the parallel with the terminology used in the reinforcement learning literature. To obtain these pairs, we sample a collection of problem instances, which are solved to optimality using a MILP solver. For each optimal schedule, we compute the corresponding optimal route order η^v for each intersection. From these, we can construct $\mathcal{X}$ in different ways. For example, for each solved instance, we can randomly select some intersection order μ, which fixes the crossing order η. We can then replay this sequence of actions step-by-step to obtain the corresponding sequence of state-action pairs. The model might become more robust when training on multiple samples of intersections orders per instance. Alternatively, we can consider one of the fixed intersection orders described at the start of this section (random, boundary, alternating). Furthermore, combined with one of the

Table 1: Average scaled optimal objective value computed using MILP and using the threshold heuristic with threshold $\tau = 0$. Each class of problem instances is identified by the number n of vehicle arrivals per route and the grid network size as cols x rows.

n	size	MILP	time	$\tau = 0$ (gap)	random (gap)	boundary (gap)	alternate (gap)
5	2x1	57.27	0.06	65.27 (11.45%)	58.96 (2.95%)	58.72 (2.52%)	58.23 (1.66%)
5	3x1	57.67	0.12	68.34 (15.44%)	59.77 (3.65%)	59.82 (3.74%)	58.72 (1.83%)
5	3x2	57.35	1.38	69.17 (18.32%)	60.88 (6.16%)	60.36 (5.25%)	58.82 (2.56%)

above strategies, we can also employ some kind of fixed lookahead procedure, as illustrated in Figure **??**. At inference time, we use the same intersection order as during training and at each step we greedily select r_t.

We will now describe how the model is parameterized based on recurrent embeddings of the sequences of crossing time lower bounds at each crossing, in a somewhat similarly to how we used the horizons in the single intersection model. Let $k(r, v)$ denote the number of scheduled vehicles at crossing (r, v) and let n_r denote the total number of vehicles on route r. For each crossing (r, v), consider the earliest crossing time of the next unscheduled vehicle, which we denote as

$$T(r, v) = \beta(r, k(r, v) + 1, v).$$

We define the *horizon* of crossing (r, v) to be the sequence of relative lower bounds

$$h(r, v) = (\beta(r, k(r, v) + 2, v) - T, \ldots, \beta(r, n_r, v) - T).$$

Each such horizon is embedded using an Elman RNN. To produce a logit for each crossing, these embeddings are fed through a feedforward network, consisting of two hidden layers of size 128 and 64 neurons with relu activation and batch normalization layers in between. See Figure **??** for a schematic overview of the architecture. Next: instead of $h(r, v)$, feed $(T(r, v), h(r, v))$ to the FF.

To train the model, we use the Adam optimizer with learning rate 10^{-4} and a batch size of 40 state-action pairs. We experienced a case of the exploding gradients problem when we did not use the batch normalization layers. To allow a fair comparison of methods accross instances of different sizes, both in terms of the number of vehicles and the size of the network, we adapt both objective as follows. For the first variant, we divide by the total number of vehicles, so we report $\text{obj}_1(y)/|\mathcal{N}|$. Given some problem instance, let N denote the length of a crossing sequence, so it is also the total number of vehicle-intersection pairs (i, v) that need to be scheduled. For the second objective variant, we consider the average delay per vehicle-intersection pair, so we report $\text{obj}_2(y)/N$, which we report in Table 1.

3.2 Reinforcement learning

Instead of using a fixed training set $\mathcal{X}$ of state-action pairs during training, we can fit our model from the perspective of a reinforcement learning problem, which we already alluded to in Section 3.1.2. More precisely, given some scheduling problem instance s, we are dealing with a Deterministic Markov Decision Process (DMDP), where partial disjunctive graphs serve as states and crossings correspond to actions. The potential benefit of using reinforcement learning is that we do not have to fix an intersection order: our hope is that the training procedure will automatically converge to some good intersection order.

The threshold heuristic ($\tau = 0$) can provide a *baseline* for reinforcement learning, reducing the variance of the REINFORCE estimator.

N.B. The step-numbering is different in $G_0, \eta_1, G_1, \eta_2, \ldots$ from the common RL notation $S_0, A_0, R_1, S_1, A_1, \ldots$, so generally $A_t = \eta_{t+1}$.

References

[1] D. Miculescu and S. Karaman, "Polling-systems-based Autonomous Vehicle Coordination in Traffic Intersections with No Traffic Signals," July 2016.

[2] R. F. Hartl, S. P. Sethi, and R. G. Vickson, "A Survey of the Maximum Principles for Optimal Control Problems with State Constraints," *SIAM Review*, vol. 37, no. 2, pp. 181–218, 1995.

[3] S. P. Sethi, *Optimal Control Theory: Applications to Management Science and Economics*. Cham: Springer International Publishing, 2019.

[4] O. Vinyals, S. Bengio, and M. Kudlur, "Order Matters: Sequence to sequence for sets," Feb. 2016.