Offline Trajectory Optimization of Autonomous Vehicles in a Single Intersection

Jeroen van Riel

March 2025

Contents

1 Introduction

- general intro
  - societal relevance
  - outlook
  - other (non)technical challenges
- literature review
  - Autonomous Intersection Management
  - Hult et al.
  - Miculescu & Karaman
- overview of problem aspects
  - information

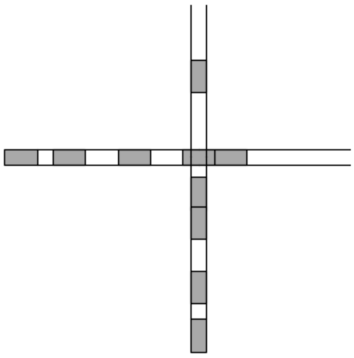

Figure 1: Illustration of a single intersection with vehicles drawn as grey rectangles. Vehicles approach the intersection from the east and from the south and cross it without turning. Note that the first two waiting vehicles on the south lane kept some distance before the intersection, such that they are able to reach full speed whenever they cross.

    * offline vs online
    * stochastic elements
- crossing order
- trajectory generation
- robustness of control

2 Model definition and analysis

This document considers the offline trajectory optimization problem for a single intersection. Recall that *offline* meant that all future arrivals to the system are known beforehand and that we assume that routes are fixed to avoid having to address some kind of dynamic routing problem. In this case, we can consider the longitudinal position $x_i(t)$ of each vehicle i along its route, for which we use the well-known *double integrator* model

$$\begin{aligned} \dot{x}_i(t) &= v_i(t), \\ \dot{v}_i(t) &= u_i(t), \\ 0 \leq v_{\max} &\leq v_{\max}, \\ |u_i(t)| &\leq a_{\max}, \end{aligned} \tag{1}$$

where $v_i(t)$ is the vehicle's velocity and $u_i(t)$ its acceleration, which is set by a single central controller. Let $D_i(s_{i,0})$ denote the set of all trajectories $x_i(t)$ satisfying these dynamics, given some initial state $s_{i,0} = (x_i(0), v_i(0))$.

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

$$\mathcal{N} = \{(r, k) : k \in \{1, \dots, n_r\}, r \in \mathcal{R}\}, \tag{2}$$

where n_r denotes the number of vehicles following route r. Smaller values of k correspond to reaching the intersection earlier. Given vehicle index $i = (r, k) \in \mathcal{N}$, we also use the notation $r(i) = r$ and $k(i) = k$. We assume that each vehicle is represented as a rectangle of length L and width W and that its position $x_i(t)$ is measured as the distance between its front bumper and the start of the lane. In order to maintain a safe distance between consecutive vehicle on the same lane, vehicle trajectories need to satisfy

$$x_i(t) - x_j(t) \geq L, \tag{3}$$

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

$$(x_i(t), x_j(t)) \notin \mathcal{E}_i \times \mathcal{E}_j, \tag{4}$$

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

$$\begin{aligned}
\min_{\mathbf{x}(t)} \quad & \sum_{i \in \mathcal{N}} J(x_i) && && \text{(5a)} \\
\text{s.t.} \quad & x_i \in D_i(s_{i,0}), && \text{for all } i \in \mathcal{N}, && \text{(5b)} \\
& x_i(t) - x_j(t) \geq L, && \text{for all } (i,j) \in \mathcal{C}, && \text{(5c)} \\
& (x_i(t), x_j(t)) \notin \mathcal{E}_i \times \mathcal{E}_j, && \text{for all } \{i,j\} \in \mathcal{D}, && \text{(5d)}
\end{aligned}$$

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

2.1 Direct transcription

Although computationally demanding, problem (5) can be numerically solved by direct transcription to a non-convex mixed-integer linear program by discretization on a uniform time grid. Let K denote the number of discrete time steps and let Δt denote the time step size. Using the forward Euler integration scheme, we have

$$\begin{aligned}
x_i(t + \Delta t) &= x_i(t) + v_i(t)\Delta t, \\
v_i(t + \Delta t) &= v_i(t) + u_i(t)\Delta t,
\end{aligned}$$

for each $t \in \{0, \Delta t, \dots, K\Delta t\}$. Following the approach in [1], the collision-avoidance constraints between lanes can be formulated using the well-known big-M technique by the constraints

$$\begin{aligned}
x_i(t) &\leq B_i + \delta_i(t) M, \\
E_i - \gamma_i(t) M &\leq x_i(t), \\
\delta_i(t) + \delta_j(t) + \gamma_i(t) + \gamma_j(t) &\leq 3,
\end{aligned}$$

where $\delta_i(t), \gamma_i(t) \in \{0, 1\}$ for all $i \in \mathcal{N}$ and M is a sufficiently large number. Finally, the follow constraints can simply be added as

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

for each $t \in \{0, \Delta t, \dots, K\Delta t\}$ and each pair of consecutive vehicles $(i, j) \in \mathcal{C}$ on the same lane. For example, consider the objective functional

$$J(x_i) = \int_{t=0}^{t_f} \left((v_d - v_i(t))^2 + u_i(t)^2 \right) dt,$$

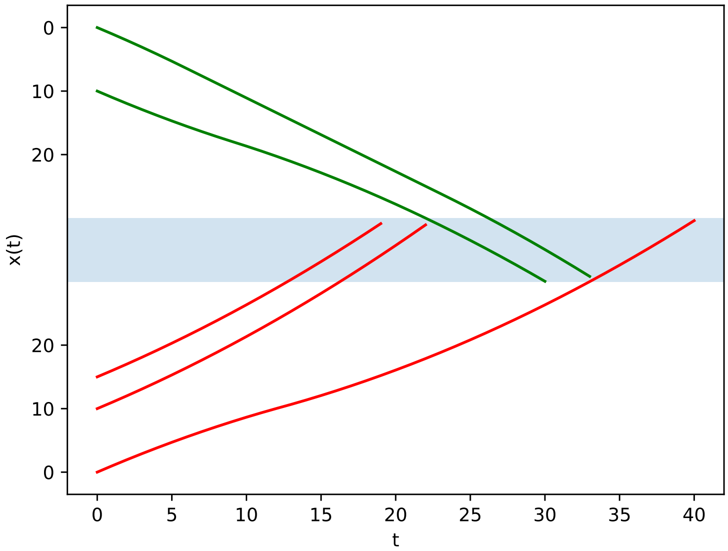

Figure 2: Example of optimal trajectories obtained using the direct transcription method with $L = 5$, $\mathcal{E}_i \equiv \mathcal{E} = [50, 70]$, $v_d = 20$, $T = 120$, $\Delta t = 0.1$ and initial conditions as given in Table 1. The y-axis is split such that each part corresponds to one of the two lanes and the trajectories are inverted accordingly and drawn with separate colors. The intersection area $\mathcal{E}$ is drawn as a shaded region. Whenever a vehicle has left the intersection, we stop drawing its trajectory for clarity.

where v_d is some reference velocity and t_f denotes the final time, then the optimal trajectories are shown in Figure 2.

i	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)
$x_i(0)$	15	10	0	10	0
$v_i(0)$	10	10	10	10	10

Table 1: Example initial conditions $s_{i,0} = (x_i(0), v_i(0))$ for problem (5).

2.2 General decomposition

For the case where only a single vehicle is approaching the intersection for each route, so $n_r = 1$ for each route $r \in \mathcal{R}$, it has been shown that problem (5) can be decomposed into two coupled optimization problems, see Theorem 1 in [1]. Roughly speaking, the *upper-level problem* optimizes the time slots during which vehicles occupy the intersection, while the *lower-level problem* produces optimal safe trajectories that respect these time slots. When allowing multiple vehicles per lane, we show without proof that a similar decomposition is possible. Given $x_i(t)$, the *crossing time* of vehicle i, when the vehicle first enters the intersection, and the corresponding *exit time* are respectively

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

The upper-level problem is to find a set of feasible occupancy timeslots, for which the lower-level problem generates trajectories. We will use decision variable $y(i)$ for the crossing time and write $y(i) + \sigma(i)$ for the exit time. It turns out that trajectories can be generated separately for each route, which yields the decomposition

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

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

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

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

2.3 Decomposition for delay objective

Assume that the trajectory performance criterion is exactly the crossing time, so $J(x_i) = \inf\{t : x_i(t) \in \mathcal{E}_i\}$. This assumption makes the problem significantly simpler, because we have

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

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

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

Therefore, we ignore the part related to σ in the set of feasible parameters $\mathcal{S}_r$, which can be shown that to have a particularly simple structure under these assumptions. Observe that $a_i := (B_i - x_i(0))/v_{\max}$ is the earliest time at which vehicle i can enter the intersection. Let $\rho := L/v_{\max}$ be such that $y(i) + \rho$ is the time at which the rear bumper of a crossing vehicle reaches the start line of the intersection. We will refer to a_i and ρ as the *arrival time* and *processing time*, respectively. It can now be shown that $y_r \in \mathcal{S}_r$ holds whenever

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

Therefore, under the stated assumptions, the offline trajectory optimization problem (5) reduces to the following *crossing time scheduling* problem

$$\begin{aligned} \min_{y} \quad & \sum_{i \in \mathcal{N}} y(i) && && \text{(6a)} \\ \text{s.t.} \quad & a_i \leq y(i), && \text{for all } i \in \mathcal{N}, && \text{(6b)} \\ & y(i) + \rho \leq y(j), && \text{for all } (i,j) \in \mathcal{C}, && \text{(6c)} \\ & y(i) + \sigma \leq y(j) \text{ or } y(j) + \sigma \leq y(i), && \text{for all } \{i,j\} \in \mathcal{D}. && \text{(6d)} \end{aligned}$$

This problem can be solved using off-the-shelf mixed-integer linear program solvers, after encoding the *disjunctive constraints* (6d) using the big-M technique, which we will demonstrate in Section 3.1. Given optimal crossing time schedule y^*, any set of trajectories $[x_i(t) : i \in \mathcal{N}]$ that satisfies

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

forms a valid solution. These trajectories can be computed with an efficient direct transcription method. Note that each route may be considered separately. Therefore, trajectories can be computed by solving the time-discretized version of the optimal control problem

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

where τ is the required crossing time, B denotes the distance to the intersection, s_0 is the initial state of the vehicle and x' denotes the trajectory of the vehicle preceding the current vehicle.

3 Crossing time scheduling

Given a crossing time schedule y, trajectories can be efficiently computed using a direct transcription method. Hence, we focus on solving the crossing time scheduling problem (6). Before we start discussing various solution techniques, let us first introduce an alternative way of representing instances of (6) by means of a graph. Once we extend the current model to networks of intersection, this encoding will be particularly helpful.

Instances and solutions of the crossing time optimization problem (6) can be represented by their *disjunctive graph* $(\mathcal{N}, \mathcal{C}, \mathcal{O})$, which is a directed graph with nodes $\mathcal{N}$ and the following two types of arcs. The *conjunctive arcs* encode the fixed order of vehicles driving on the same lane. For each $(i, j) \in \mathcal{C}$, an arc from i to j means that vehicle i reaches the intersection before j due to the follow constraints (6c). The *disjunctive arcs* are used to encode the decisions regarding the ordering of vehicles from distinct lanes, corresponding to constraints (6d). For each pair $\{i, j\} \in \mathcal{D}$, at most one of the arcs (i, j) and (j, i) can be present in $\mathcal{O}$.

When $\mathcal{O} = \varnothing$, we say the disjunctive graph is *empty*. Each feasible schedule satisfies exactly one of the two constraints in (6d). When $\mathcal{O}$ contains exactly one arc from every pair of opposite disjunctive arcs, we say the disjunctive graph is *complete*. Note that such graph is acyclic and induces a unique topological ordering π of its nodes. Conversely, every ordering π of nodes $\mathcal{N}$ corresponds to a unique complete disjunctive graph, which we denote by $G(\pi) = (\mathcal{N}, \mathcal{C}, \mathcal{O}(\pi))$.

We define weights for every possible arc in a disjunctive graph. Every conjunctive arc $(i, j) \in \mathcal{C}$ gets weight $w(i, j) = \rho_i$ and every disjunctive arc $(i, j) \in \mathcal{O}$ gets weight $w(i, j) = \sigma_i$.

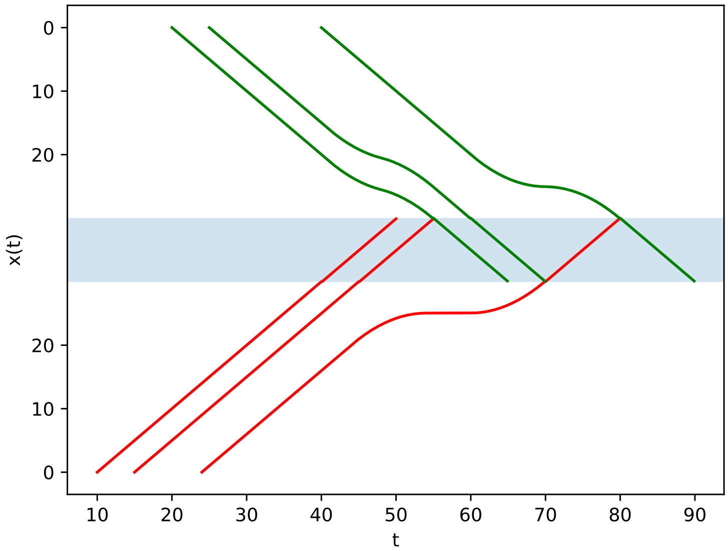

Figure 3: Example trajectories of vehicles from a single route, for some random arrival times and scheduled crossing times, computed by solving the linear program obtained from direct transcription of problem `MotionSynthesize`. We used the parameters $v_{\max} = 1, a_{\max} = 0.5, L = 5$. The "queueing behavior" that we also see in traffic jams is visible for the trajectories in the upper part. This is due to the particular choice of optimization objective, which essentially tries to keep all vehicles as close to the intersection as possible at all times.

Given some vehicle ordering π, for every $j \in \mathcal{N}$, we recursively define the lower bound

$$\beta(j) = \max\{a_j, \max_{i \in N_\pi^-(j)} \beta(i) + w(i,j)\}, \tag{7}$$

where $N_\pi^-(j)$ denotes the set of in-neighbors of node j in $G(\pi)$. Observe that this quantity is a lower bound on the crossing time, i.e., every feasible schedule y with ordering π must satisfy $y_i \geq \beta(i)$ for all $i \in \mathcal{N}$. As the following result shows, it turns out that this lower bound is actually tight for optimal schedules, which allows us to calculate the optimal crossing times y^* once we know an optimal ordering π^* of vehicles, so we can concentrate on finding the latter.

Proposition 1. *If y is an optimal schedule for* (6) *with ordering π, then*

$$y_i = \beta(i) \quad \textit{for all } i \in \mathcal{N}. \tag{8}$$

3.1 Branch-and-cut

Optimization problem (6) can be turned into a Mixed-Integer Linear Program (MILP) by rewriting the disjunctive constraints using the well-known big-M method. We introduce a binary decision variable γ_{ij} for every disjunctive pair $\{i,j\} \in \mathcal{D}$. To avoid redundant variables, we first impose some arbitrary ordering of the disjunctive pairs by defining

$$\bar{\mathcal{D}} = \{(i,j) : \{i,j\} \in \mathcal{D},\ l(i) < l(j)\},$$

such that for every $(i,j) \in \bar{\mathcal{D}}$, setting $\gamma_{ij} = 0$ corresponds to choosing disjunctive arc $i \to j$ and $\gamma_{ij} = 1$ corresponds to $j \to i$. This yields the following MILP formulation

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

where $M > 0$ is some sufficiently large number. Next, we will discuss two types of cutting planes that can be added to this formulation, which we hope improve the solving time.

Consider some disjunctive arc $(i,j) \in \bar{\mathcal{D}}$. Let $pred(i)$ denote the set of indices of vehicles that arrive no later than i on route $r(i)$. Alternatively, we could say these are all the vehicles from which there is a path of conjunctive arcs to i. Similarly, let $succ(j)$ denote the set of indices of vehicles that arrive no later than j on route $r(j)$. Now suppose $\gamma_{ij} = 0$, so the direction of the arc is $i \to j$, then any feasible solution must also satisfy

$$p \to q \equiv \gamma_{pq} = 0 \text{ for all } p \in pred(i),\ q \in succ(j).$$

Written in terms of the disjunctive variables, this gives us the following cutting planes

$$\sum_{\substack{p \in pred(i) \\ q \in succ(j)}} \gamma_{pq} \leq \gamma_{ij} M,$$

for every disjunction $(i,j) \in \bar{\mathcal{D}}$. We refer to these as the *transitive cutting planes*.

Apart from the redundancy that stems from the way the disjunctions are encoded, the next proposition shows that there is some less obvious structure in the problem. Roughly speaking, whenever a vehicle can be scheduled immediately after its predecessor, this should happen in any optimal schedule. We will use these necessary conditions to define two types of additional cutting planes.

Proposition 2. *If y is an optimal schedule for* (6)*, satisfying $y_i + \rho \geq a_j$ for some $(i, j) \in \mathcal{C}$, then j follows immediately after i, so $y_i + \rho = y_j$.*

In order to model this type of necessary condition, we introduce for every conjunctive pair $(i, j) \in \mathcal{C}$ a binary variable $\delta_{ij} \in \{0, 1\}$ that satisfies

$$\begin{aligned} \delta_{ij} = 0 &\iff y_i + \rho < a_j, \\ \delta_{ij} = 1 &\iff y_i + \rho \geq a_j, \end{aligned}$$

which can be enforced by adding to the constraints

$$\begin{aligned} &y_i + \rho < a_j + \delta_{ij} M, \\ &y_i + \rho \geq a_j - (1 - \delta_{ij}) M. \end{aligned}$$

Now observe that Proposition 3 for (i, j) is modeled by the cutting plane

$$y_i + \rho \geq y_j - (1 - \delta_{ij}) M.$$

We refer to these cutting planes as *necessary conjunctive cutting planes.* Using the definition of δ_{ij}, we can add more cutting planes on the disjunctive decision variables, because whenever $\delta_{ij} = 1$, the directions of the disjunctive arcs $i \to k$ and $j \to k$ must be the same for every other vertex $k \in \mathcal{N}$. Therefore, consider the following constraints

$$\begin{aligned} &\delta_{ij} + (1 - \gamma_{ik}) + \gamma_{jk} \leq 2, \\ &\delta_{ij} + \gamma_{ik} + (1 - \gamma_{jk}) \leq 2, \end{aligned}$$

for every $(i, j) \in \mathcal{C}$ and for every $k \in \mathcal{N}$ with $r(k) \neq r(i) = r(j)$. We will refer to these types of cuts as the *necessary disjunctive cutting planes.*

3.2 Runtime analysis

For each route $r \in \mathcal{R}$, we model the sequence of earliest crossing times $a_r = (a_{r1}, a_{r2}, \dots)$ as a stochastic process, to which we refer as the *arrival process.* Recall that constraints (6c) ensure a safe following distance between consecutive vehicles on the same route. Therefore, we want the process to satisfy

$$a_{(r,k)} + \rho_{(r,k)} \leq a_{(r,k+1)},$$

for all $k = 1, 2, \dots$. We start by assuming that all vehicles share the same dimensions so that $\rho_i = \rho$ for all $i \in \mathcal{N}$. Let the interarrival times be denoted as X_n with cumulative distribution function F and mean μ, assuming it exists. We define the arrival times $A_n = A_{n-1} + X_n + \rho$, for $n \geq 1$ with $A_0 = 0$. The arrival process may be interpreted as an renewal process with interarrivals times $X_n + \rho$. Let N_t denote the corresponding counting process, then by the *renewal theorem*, we obtain the *limiting density* of arrivals

$$\mathbb{E}(N_{t+h}) - \mathbb{E}(N_t) \to \frac{h}{\mu + \rho} \quad \text{as } t \to \infty,$$

for $h > 0$. Hence, we refer to the quantity $\lambda := (\mu + \rho)^{-1}$ as the arrival intensity.

In order to model the natural occurence of platoons, we model F as a mixture of two random variables, one with a small expected value μ_s to model the gap between vehicles within the same platoon and one with a larger expected value μ_l to model the gap between vehicles of different platoons. For example, consider a mixture of two exponentials, such that

$$\begin{aligned} F(x) &= p(1 - e^{-x/\mu_s}) + (1 - p)(1 - e^{-x/\mu_l}), \\ \mu &= p\mu_s + (1 - p)\mu_l, \end{aligned}$$

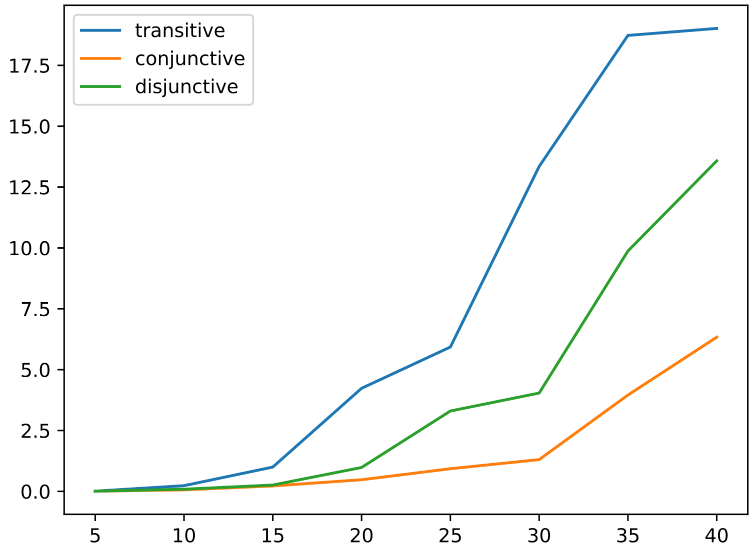

Figure 4: The average (censored) running time of the branch-and-cut procedure is plotted as a function of the number of arriving vehicles per route, for each of the three indicated cutting planes. Each average is computed over 10 problem instances. All instances use $\rho = 4$ and $\sigma = 1$. The arrivals of each of the two routes are generated using the bimodal exponential interarrival times with $p = 0.5, \mu_s = 0.1, \mu_l = 10$. This figure clearly shows that the conjunctive cutting planes provide the most runtime improvement.

assuming $\mu_s < \mu_l$. Observe that the parameter p determines the average length of platoons. Consider two intersecting routes, $\mathcal{R} = \{1, 2\}$, with arrival processes $a_1 = (a_{11}, a_{12}, \dots)$ and $a_2 = (a_{21}, a_{22}, \dots)$, with arrival intensities $\lambda^{(1)} = \lambda^{(2)}$. We keep $\lambda_s = 0.5$ constant, and use

$$\mu_l = \frac{\mu - p\mu_s}{1 - p}$$

to keep the arrival rate constant accross arrival distributions.

We now assess which type of cutting planes yields the overall best performance. The running time of branch-and-cut is mainly determined by the total number of vehicles in the instance. Therefore, we consider instances with two routes and measure the running time of branch-and-cut as a function of the number of vehicles per route. In order to keep the total computation time limited, we set a time limit of 20 seconds for solving each instance. Therefore, we should be careful when calculating the average running time, because some observations may correspond to the algorithm reaching the time limit, in which case the observation is said to be *censored*. Although there are statistical methods to rigorously deal censored data, we do not need this for our purpose of picking the best type of cutting planes. Figure 4 shows the average (censored) running time for the three types of cutting planes. Observe that the necessary conjunctive cutting planes seem to lower the running time the most.

4 Constructive heuristics

Methods that rely on the branch-and-cut framework are guaranteed to find an optimal solution, but their running time scale very badly with increasing instance sizes. Therefore, we are interested in developing heuristics to obtain good approximations in reasonable time. A common approach for developing such heuristics in the scheduling literature is to try and construct a good schedule in a step-by-step fashion. For our crossing time scheduling problem, we will consider methods to incrementally construct a vehicle ordering, to which we will refer as *constructive heuristics*.

In what follows, we will work with a partial ordering π, considered to be a permutation of the set of scheduled vehicles, denoted as $\mathcal{N}(\pi) \subset \mathcal{N}$. Observe that ordering vehicles is equivalent to ordering the routes, due to the conjunctive constraints, so we will define constructive heuristics in terms of repeatedly choosing the next route. Hence, it may be helpful to model this process as a deterministic finite-state automaton, where the set of route indices acts as the input alphabet $\mathcal{R} = \{1, \dots, n\}$. Let S denote the state space and let $\delta : S \times \mathcal{R} \to S$ denote the transition function. Let s denote an instance of crossing time scheduling problem (6). Formally, we must consider s to be a fixed part of the state, so it does not change with transitions. The other part of the state is the current partial ordering π. Let $(s, \pi) \in S$ denote the current state and let $r \in \mathcal{R}$ denote the next input. Let $i \in \mathcal{N} \setminus \mathcal{N}(\pi)$ denote the next unscheduled vehicle on route r, then the system transitions to $(s, \pi + i)$, where we use $+$ to denote sequence concatenation. If no such vehicle exists, the transition is undefined. We will use η to denote a valid sequence of route indices, we let $y_\eta(s)$ denote the schedule that is obtained by executing η on the automaton. We say that route order η is optimal whenever $y_\eta(s)$ is optimal. In the following, we will use $\text{obj}(y)$ to denote the objective value of schedule y, which is just the sum of the crossing times of all the vehicles.

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

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

yields a final state $s_N(s, \pi^*)$, where π^* is an optimal ordering and $N = \sum_{r \in \mathcal{R}} n_r$ denotes the total number of steps required to arrive at a final state. Observe that such a mapping must exist, because we can always set $p(s_t) = \eta^*_{t+1}$ for every step t, given some optimal lane order η^*. However, we do not hope to find an explicit representation of p, because it would in general be very complex, so our aim is to find good approximations.

The approximations that we will propose are based on the the earliest crossing times of the unscheduled vehicles at each state s_t. Note that definition in (7) still makes sense for partial orderings; the only difference is that the corrsponding disjunctive graph $G(\pi)$ is not complete. We will use β_t to refer to the earliest crossing times, using t to emphasize its dependency on the current state $s_t = (s, \pi)$.

4.1 Threshold heuristics

We know from Proposition 3 that whenever it is possible to schedule a vehicle immediately after its predecessor on the same route, then this must be done in any optimal schedule. Based on this idea, we might think that the same holds true whenever a vehicle can be scheduled *sufficiently* soon after its predecessor. Although this is not true in general, we can define a simple constructive heuristic based on this idea. For every route $r \in \mathcal{R}$, let $k(r)$ denote the number of vehicles that have already been scheduled. Hence, let $i = (\eta_t, k(\eta_t))$ denote the vehicle that was scheduled in the last step of the automaton. We define the *threshold policy*

$$p_\tau(s_t) = \begin{cases} \eta_t & \text{if } \beta_t(i) + \rho + \tau \geq a_j \text{ and } (i, j) \in \mathcal{C}, \\ \texttt{next}(\eta_t) & \text{otherwise,} \end{cases}$$

for some threshold $\tau \geq 0$, where the expression $\texttt{next}(\eta_t)$ represents some route other than η_t with unscheduled vehicles left. Let $y_\tau(s)$ denote the schedule that is obtained by executing the heuristic with threshold τ on some instance s. We pick the value of τ that minimizes the average empirical objective over some set of training instances $\mathcal{X}$, by defining

$$\hat{\tau} = \arg\min_{\tau \geq 0} \sum_{s \in \mathcal{X}} \text{obj}(y_\tau(s)).$$

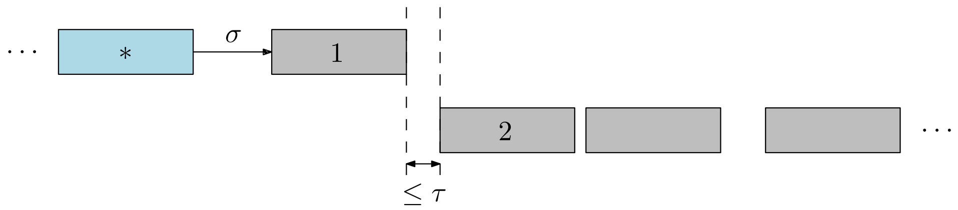

Figure 5: Illustration of the threshold heuristic for a specific situation during scheduling. The upper row represents the current partial schedule, where the vehicle marked with $*$ is from route 1, whereas all other vehicles are from route 2. Hence, the arrow indicates the switch time σ. The bottom row shows each unscheduled vehicles i of route 2, placed at the earliest crossing times $\beta_t(i)$. Whenever the indicated distance is smaller than τ, the threshold rule requires vehicle 2 to be scheduled next.

With the specific choice $\tau = 0$, the threshold rule is related to the *exhaustive policy* for polling systems. More precisely, executing this threshold rule on the automaton may be interpreted as running a discrete event simulation of a polling system with an exhaustive polling policy.

4.2 Neural heuristic

We will now consider a class of heuristics that directly generalizes the threshold heuristic. Instead of looking at the earliest crossing time of the next vehicle in the current lane, we will now consider the earliest arrival times of all unscheduled vehicles across lanes. Because we want to tune the heuristic based on problem instances, we will formulate it as a model of the conditional probability $p_\theta(\eta_{t+1}|s_t)$ of the next route, given the current partial order. Here, θ denotes the parameters that need to tuned to the specific class of problem instances.

We will first explain how we parameterize p_θ as a function of the current non-final state $s_t = (s, \pi_t)$ of the automaton. In the following definitions, the dependence on s_t will be left implicit to avoid cluttering the notation and we let $k(r)$ denote again the number of scheduled vehicles of route r. Observe that the overall earliest arrival time of any unscheduled vehicle in the system is given by

$$T = \min_{r \in \mathcal{R}} \beta(r, k(r) + 1).$$

We define the *horizon* of route r to be the sequence of relative lower bounds

$$h(r) = (\beta(r, k(r) + 1) - T, \beta(r, k(r) + 2) - T, \ldots, \beta(r, n_r) - T).$$

Next, we compute for each horizon an embedding $\bar{h}(r)$. Observe that horizons can be of arbitrary dimension. Therefore, we could restrict each horizon to a fixed length by using zero padding. To avoid the zero padding operation, which can be problematic for states that are almost done, we employ a recurrent neural network for variable-length sequences, to obtain a model that is agnostic to the number of remaining unscheduled vehicles. Each horizon $h(r)$ is simply transformed into a fixed-length embedding by feeding it in reverse order through a plain Elman RNN. Generally speaking, the most recent inputs tend to have greater influence on the output of an RNN, which is why we feed the horizon in reverse order, such that those vehicles that are due first are processed last, since we expect those should have the most influence on the decision. Finally, the horizon embeddings are arranged into a vector H, defined as

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

for every $r \in \mathcal{R}$, with η_t denoting the current route (of the last scheduled vehicle). By employing this kind of *cycling trick*, we make sure that the embedding of the current route

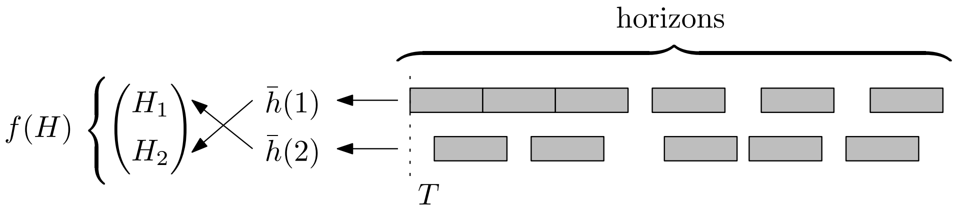

Figure 6: Schematic overview of how $p_\theta(\eta_{t+1}|s_t) = f(H(s_t))$, the conditional distribution over the next route to be chosen, is computed from the current route horizons via embedding and the cyling trick. Observe that this particular cycling corresponds to a situation in which the current route is $\eta_t = 2$.

is always kept at the same position of the vector. Using some fully connected neural network f, this global embedding is mapped to the probability distribution $p_\theta(\eta_{t+1}|s_t) = f(H(s_t))$.

Note that the above model depends on the parameters of f and possibly on the parameters of the embedding, when using the recurrent approach. Let θ denote the vector of all these parameters. Consider an instance s and some optimal route sequence η with corresponding states defined as $s_{t+1} = \delta(s_t, \eta_{t+1})$. The resulting set of pairs $\mathcal{X} := \{(s_t, \eta_{t+1}) : t = 1, \dots, N\}$ can be used to learn p_θ in a supervised fashion by treating it as a classification task and computing the maximum likelihood estimator $\hat{\theta}$. We make this concrete for the case of two routes $\mathcal{R} = \{1, 2\}$. Let $p_\theta(s_t)$ denote the probability of choosing the first route and use the binary cross entropy loss, defined as

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

where $\mathbb{1}\{\cdot\}$ denotes the indicator function. Now we can simply rely on some (stochastic) gradient-descent optimization method to determine

$$\hat{\theta} = \arg\min_\theta L_\theta(\mathcal{X}).$$

Once we determined the model parameters, schedules can be generated by employing greedy inference as follows. The model p_θ provides a distribution over lanes. We ignore lanes that have no unscheduled vehicles left and take the argmax of the remaining probabilities. We will denote the corresponding complete schedule by $\hat{y}_\theta(s)$.

4.3 Performance evaluation

The assessment of the various solution methods is roughly based on two aspects. Of course, the quality of the produced solutions is important. Second, we need to take into account the time that the algorithm requires to compute the solutions. We need to be careful about reporting the time requirements of the constructive heuristics, because they need both training time as well as inference time. Only reporting the latter time does not provide a fair comparison.

We consider several problem instance distributions, because these may possibly affect the performance of algorithms. The most important aspects that characterize the distribution of problem instances are number of routes and number of arrivals per route, distribution of interarrival times, arrival intensity per route and degree of platooning.

We will refer to the total number of vehicles in an instance as its *size*. In the following, let $\mathcal{X}$ and $\mathcal{Y}$ denote a set of *training instances* and *test instances*, respectively. To enable a fair comparison across different instance classes, we report the quality of a solution in terms of the average delay divided by the instance size. More precisely, from now on we define the objective of some schedule y as

$$\mathrm{obj}(y) = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} y_i - a_i.$$

Ideally, we want to report the average *approximation ratio* of each heuristic, which is defined as

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

where $y^*(s)$ denotes some optimal solution for instance s and $\hat{y}(s)$ denotes some solution computed by a heuristic fitted to $\mathcal{X}$. Again, we use a fixed time limit of 60 seconds per instance to bound the total computation time of the analysis. Therefore, it might be that we cannot compute $y^*(s)$ using the branch-and-cut procedure for some of the larger instances. Instead of reporting the actual approximation ratio α_{approx}, we report the relative optimality gap, where we use the best solution that branch-and-cut obtains within the time limit as proxy for $y^*(s)$.

The heuristics can be fitted to the training data without requiring much manual tweaking. The threshold heuristics is fitted by choosing the best value of τ from a fixed set of candidates through a simple search. The neural heuristics is trained for a fixed number of training steps. At regular intervals, we compute the average validation loss and store the current model parameters. At the end of the training, we pick the model parameters with the smallest validation loss. We create some plots that allow us to manually inspect the model fit. For the threshold heuristics, we plot the average objective for the values of τ in the grid search, see Figure 9. For the neural heuristic, we plot the training and validation loss, see Figure 10. It can be seen that the model converges very steadily in all cases.

```
## Error in `select()`:
## !  Can't subset columns that don't exist.
## x Column `opt_time` doesn't exist.
## Error in dimnames(x) <- dn:  length of 'dimnames' [2] not equal to array extent
```

5 Other approaches

5.1 Local search

The previous section showed that constructive heuristics perform reasonably well on average and sometimes even produce optimal solutions. To further increase optimality of the heuristic solutions, without relying on systematic search methods like branch-and-bound, we can try to use some kind of local search. Specifically, compute a solution using one of the methods from the previous section and then explore some *neighbor solutions*, that we define next.

As seen in the previous sections, vehicles of the same route occur mostly in groups, to which we will refer as *platoons*. For example, consider for example the route order $\eta = (0, 1, 1, 0, 0, 1, 1, 1, 0, 0)$. This example has 5 platoons of consecutive vehicles from the same route. The second platoon consists of two vehicles from route 1. The basic idea is to make little changes in these platoons by moving vehicles at the start and end of a platoon to the previous and next platoon of the same route. More precisely, we define the following two types of modifications to a route order. A *right-shift* modification of platoon i moves the last vehicle of this platoon to the next platoon of this route. Similarly, a *left-shift* modification of platoon i moves the first vehicle of this platoon to the previous platoon of this route. We construct the neighborhood of a solution by performing every possible right-shift and

platoon id	left-shift	right-shift
1		(1, 1, 0, 0, 0, 1, 1, 1, 0, 0)
2	(1, 0, 1, 0, 0, 1, 1, 1, 0, 0)	(0, 1, 0, 0, 1, 1, 1, 1, 0, 0)
3	(0, 0, 1, 1, 0, 1, 1, 1, 0, 0)	(0, 1, 1, 0, 1, 1, 1, 0, 0, 0)
4	(0, 1, 1, 1, 0, 0, 1, 1, 0, 0)	(0, 1, 1, 0, 0, 1, 1, 0, 0, 1)
5	(0, 1, 1, 0, 0, 0, 1, 1, 1, 0)	

Table 2: Neighborhood of route order $\eta = (0, 1, 1, 0, 0, 1, 1, 1, 0, 0)$.

left-shift with respect to every platoon in the route order. For illustration purposes, we have listed a full neighborhood for some example route order in Table 2.

Now using this definition of a neighborhood, we must specify how the search procedure visits these candidates. In each of the following variants, the value of each neighbor is always computed. The most straightforward way is to select the single best candidate in the neighborhood and then continue with this as the current solution and compute its neighborhood. This procedure can be repeated for some fixed number of times. Alternatively, we can select the k best neighboring candidates and then compute the combined neighborhood for all of them. Then in the next step, we again select the k best candidates in this combined neighborhood and repeat. The latter variant is sometimes referred to as *beam search*.

References

[1] 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.

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

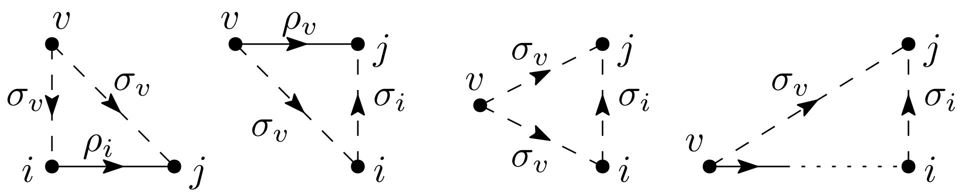

Figure 7: Sketch of the four cases distinguished in the proof of Lemma 1. Arc weights are given and disjunctive arcs $\mathcal{O}(\pi)$ are drawn with a dashed line.

A Proof of Proposition 3

First, we prove the following lemma that provides an easier expression for calculating the lower bounds.

Lemma 1. *Let π be some permutation of $\mathcal{N}$. Assume that $\sigma_i = \rho_i + s$, for every $i \in \mathcal{N}$, with $s > 0$. Consider a pair $i, j \in \mathcal{N}$ such that i is the immediate predecessor of j in π, so $\pi^{-1}(i) + 1 = \pi^{-1}(j)$, then*

$$\mathrm{LB}_\pi(j) = \max\{r_j, \mathrm{LB}_\pi(i) + w(i, j)\}. \tag{9}$$

Proof. Suppose $(i, j) \in \mathcal{C}$, see Figure 7, then the incoming disjunctive arcs of j are $N_\pi^-(j) \setminus \{i\} \subset N_\pi^-(i)$. Therefore, we have

$$\max_{v \in N_\pi^-(j)\setminus\{i\}} \mathrm{LB}_\pi(v) + \sigma_v \leq \mathrm{LB}_\pi(i),$$

so that $\mathrm{LB}_\pi(v) + w(v, j) \leq \mathrm{LB}_\pi(i) + w(i, j)$ for all $v \in N_\pi^-(j)$.

Otherwise, we have $(i, j) \in \mathcal{O}(\pi)$. Let $v \in \mathcal{N}$ such that (v, j) is an arc. If $(v, j) \in \mathcal{C}$, then we have

$$\mathrm{LB}_\pi(v) + w(v, j) = \mathrm{LB}_\pi(v) + \rho_v \leq \mathrm{LB}_\pi(v) + \sigma_v + \sigma_i \leq \mathrm{LB}_\pi(i) + w(i, j),$$

where the second inequality follows from $(v, i) \in \mathcal{O}(\pi)$. If $(v, j) \in \mathcal{O}(\pi)$ with $l(v) \neq l(i)$, then $(v, i) \in \mathcal{O}(\pi)$, so

$$\mathrm{LB}_\pi(v) + w(v, j) = \mathrm{LB}_\pi(v) + w(v, i) \leq \mathrm{LB}_\pi(i) \leq \mathrm{LB}_\pi(i) + w(i, j).$$

If $(v, j) \in \mathcal{O}(\pi)$ with $l(v) = l(i)$, then there is a path of conjunctive arcs between v and i, so we must have $\mathrm{LB}_\pi(v) + \rho_v \leq \mathrm{LB}_\pi(i)$. Furthermore, from $w(v, j) = \sigma_v = \rho_v + s$ follows that

$$\mathrm{LB}_\pi(v) + w(v, j) = \mathrm{LB}_\pi(v) + \rho_v + s \leq \mathrm{LB}_\pi(i) + s \leq \mathrm{LB}_\pi(i) + w(i, j).$$

To conclude, we have shown that $\mathrm{LB}_\pi(v) + w(v, j) \leq \mathrm{LB}_\pi(i) + w(i, j)$ for any $v \in N_\pi^-(j)$, from which statement (9) follows. □

Proposition 3. *Consider an instance of* (6) *with $\rho_i = \rho$ and $\sigma_i = \sigma > \rho$ for all $i \in \mathcal{N}$. Suppose y is an optimal schedule with $y_{i^*} + \rho \geq r_{j^*}$, for some $(i^*, j^*) \in \mathcal{C}$, then j^* follows immediately after i^*, so $y_{i^*} + \rho = y_{j^*}$.*

Proof. Suppose the ordering π of y is such that $\pi^{-1}(i^*) + 1 < \pi^{-1}(j^*)$. Let $\mathcal{I}(i, j) = \{i, \pi(\pi^{-1}(i) + 1), \dots, j\}$ be the set of vehicles between i and j. Let $f = \pi(1)$ and $e = \pi(|\mathcal{N}|)$ be the first and last vehicles, respectively, and set $u = \pi^{-1}(i^*) + 1$ and $v = \pi^{-1}(j^*) - 1$, see also Figure 8. Construct new ordering π' by moving vehicle j^* forward by $|\mathcal{I}(u, v)|$ places and let y' denote the corresponding schedule. We have $y_i = y'_i$ for all $i \in \mathcal{I}(f, i^*)$, so these do not contribute to any difference in the objective. Using Proposition 1 and Lemma 1, we compute

$$\begin{aligned}
y'_{j^*} &= \max\{r_{j^*}, y_{i^*} + \rho\} = y_{i^*} + \rho, \\
y_u &= \max\{r_u, y_{i^*} + \sigma\}, \\
y'_u &= \max\{r_u, y_{i^*} + \rho + \sigma\},
\end{aligned}$$

where we used that $y_{i^*} + \rho \geq r_{j^*}$ by assumption. Note that we have $y_{i^*} + \sigma + (|\mathcal{I}(u,v)| - 1)\rho \leq y_v$, regardless of the type of arcs between consecutive vehicles in $\mathcal{I}(u,v)$. Therefore,

$$y_{j^*} - y'_{j^*} \geq y_v + \sigma - y_{i^*} - \rho \geq 2\sigma + (|\mathcal{I}(u,v)| - 2)\rho.$$

We now show that $y'_k \geq y_k$ and $y'_k - y'_{j^*} \leq y_k - y_{i^*}$ for every $k \in \mathcal{I}(u,v)$. For $k = u$, it is clear that $y'_u \geq y_u$ and

$$y'_u - y'_{j^*} = \max\{r_u - (y_{i^*} + \rho), \sigma\} \leq \max\{r_u - y_{i^*}, \sigma\} = y_u - y_{i^*}.$$

Now proceed by induction and let x be the immediate predecessor of k for which the inequalities hold, then

$$y'_k = \max\{r_k, y'_x + w(x,k)\} \geq \max\{r_k, y_x + w(x,k)\} = y_k$$

and the second inequality follows from

$$\begin{aligned}
(y'_k - y'_x) + (y'_x - y'_{j^*}) &= \max\{r_k - y'_x, w(x,k)\} + (y'_x - y'_{j^*}) \\
&\leq \max\{r_k - y_x, w(x,k)\} + (y_x - y_{i^*}) \\
&= (y_k - y_x) + (y_x - y_{i^*}).
\end{aligned}$$

Let l denote the immediate successor of j^*, if there is one. Regardless of whether j^* and l are in the same lane, we have $y_{j^*} + \rho \leq y_l$. We derive

$$y'_v = y'_v - y'_{j^*} + y'_{j^*} \leq y_v - y_{i^*} + y'_{j^*} = y_v + \rho \leq y_{j^*} - \sigma + \rho,$$

from which follows that $y'_v + \sigma \leq y_l$, which means that $y_i \geq y'_i$ for $i \in \mathcal{I}(l,e)$.

We can now compare the objectives by putting everything together

$$\begin{aligned}
\sum_{i \in \mathcal{N}} y_i - y'_i &= y_{j^*} - y'_{j^*} + \sum_{i \in \mathcal{I}(u,v)} y_i - y'_i + \sum_{i \in \mathcal{I}(l,e)} y_i - y'_i \\
&\geq 2\sigma + (|\mathcal{I}(u,v)| - 2)\rho + \sum_{k \in \mathcal{I}(u,v)} (y_k - y_{i^*}) - (y'_k - y'_{j^*}) \\
&\qquad - |\mathcal{I}(u,v)|(y'_{j^*} - y_{i^*}) \\
&\geq 2\sigma - 2\rho > 0
\end{aligned}$$

which contradicts the assumption that y and π were optimal. Finally, from Proposition 1 and Lemma 1 follows that $y_{i^*} + \rho = y_{j^*}$. □

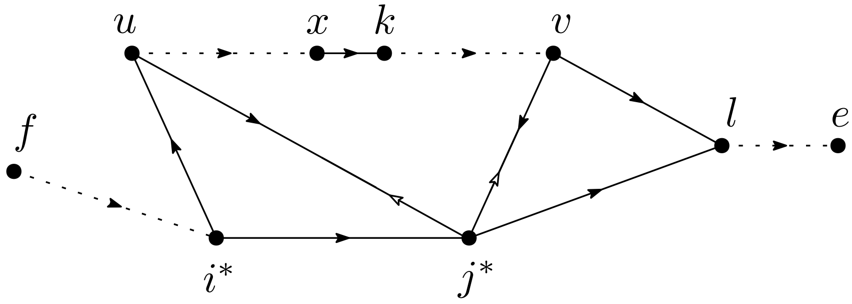

Figure 8: Sketch of the nodes and most important arcs used in the proof of Proposition 3. Dashed arcs represent chains of unspecified length. The two open arrows indicate the new direction of their arc under ordering π'.

B Implementation details

We have an `ActionTransform` base class with implementations `PaddedEmbeddingModel` and `RecurrentEmbeddingModel`. The `ActionTransform` class takes care of transforming actions in terms of staying on the same lane or moving to the next lane to actions in terms of absolute lane identifiers. Specifically, `action_transform()` takes a logit of the binary choice model and outputs a lane identifier and `inverse_action_transform()` takes a lane identifier and outputs either zero or one.

Both embedding models have a `state_transform()` method that constructs a numerical observation tensor based on the state encoded in the automaton, as explained in Section 4.2. We explicitly store the length of the horizon as the first entry of the tensor. Maybe we should use the `torch.nn.utils.rnn.pack_sequence()` function that is available in `torch`.

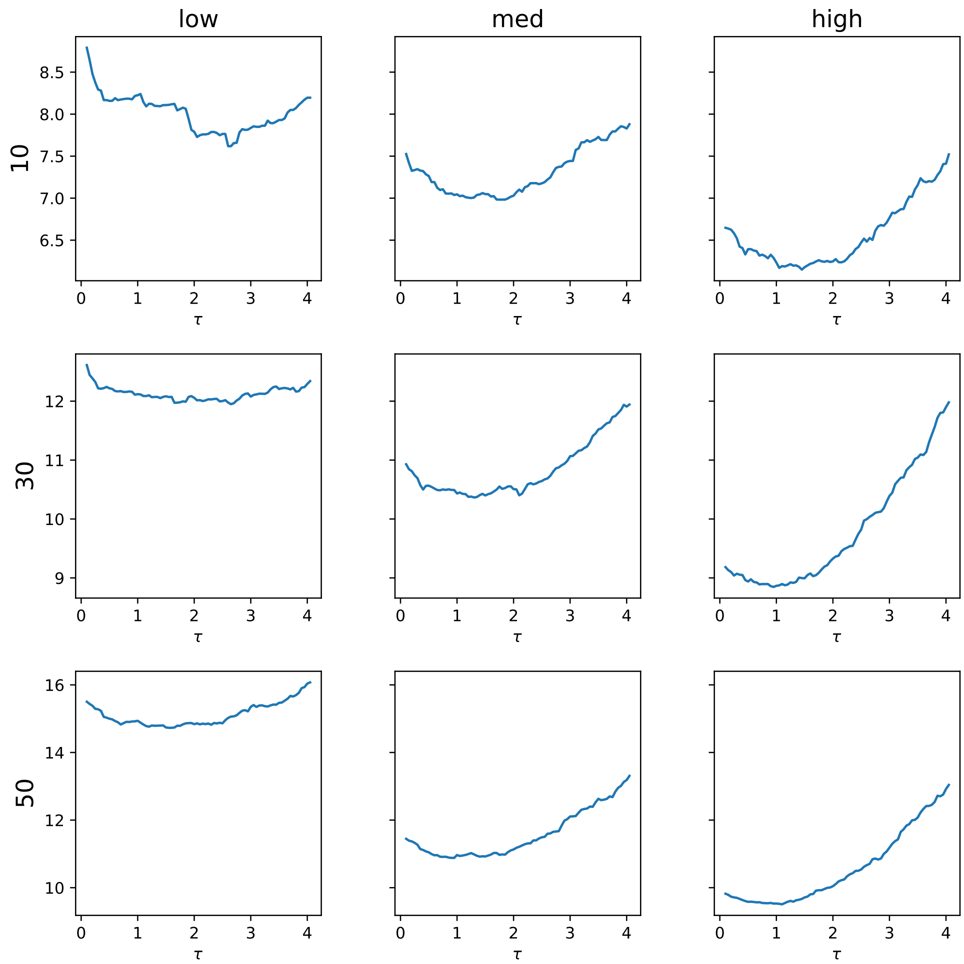

Figure 9: Charts to verify threshold heuristic model fit. For each indicated instance training set, the average delay $\sum_{s \in \mathcal{X}} \mathrm{obj}(y_\tau(s)) \,/\, |\mathcal{X}|$ is plotted as a function of the threshold τ. We use these plots to verify that the range of possible candidates for τ has been chosen wide enough, which is probably the case when the graphs are somewhat smooth and convex. Observe that larger instances show smoother curves. Furthermore, observe that the class of instances with high arrival intensity shows an apparent optimal value of τ around 1, which might be related somehow to our choice of $\sigma = 1$.

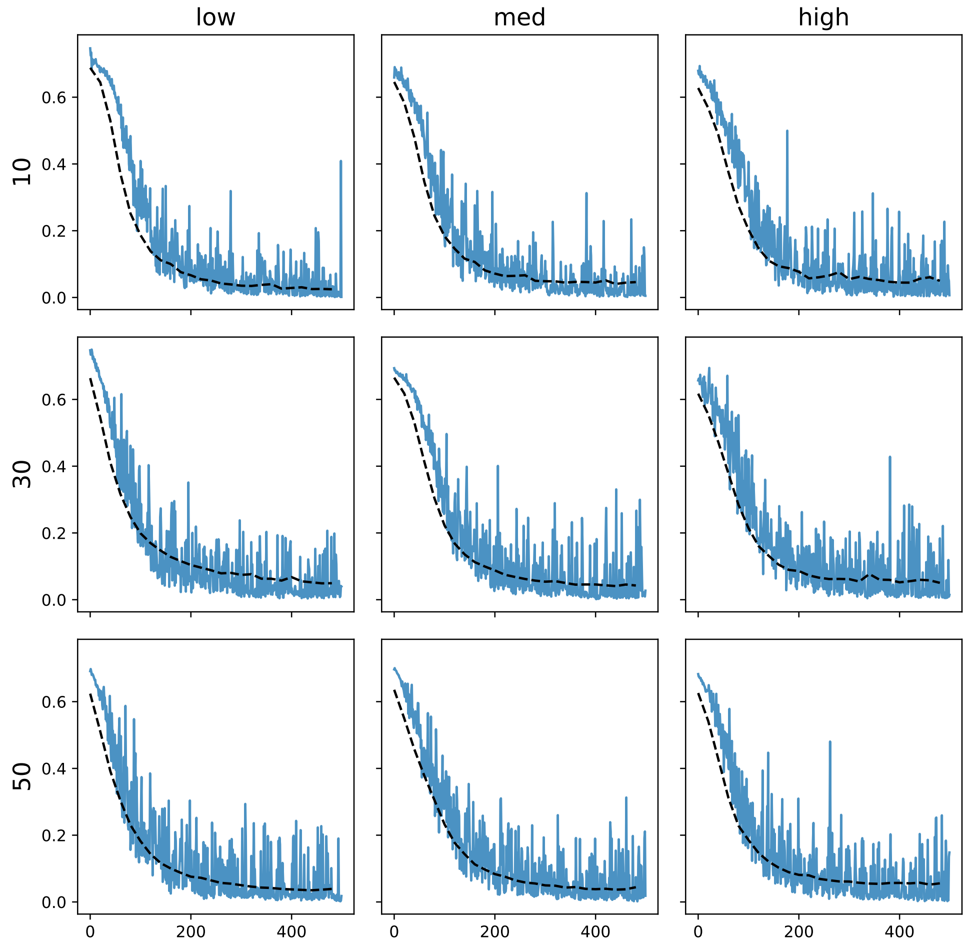

Figure 10: Loss plots to verify neural heuristic model fit. For each indicated instance training set, the training loss is plotted for each training step in blue and the validation loss is plotted as the dashed line. Recall that the validation loss is the average binary cross entropy loss after a given number of training steps. The training loss is plotted for each individual step without smooting.