Trajectory Optimization of Autonomous Vehicles in a Single Intersection

Jeroen van Riel

April 2025

Contents

1 Background

1.1 Job-shop scheduling

Constructive heuristics A common approach for developing such heuristics found in the scheduling literature is to try and construct a good schedule in a step-by-step fashion, to which may be called *constructive heuristics.* For the crossing time scheduling problem, we may consider methods that incrementally construct a vehicle ordering.

1.2 Reinforcement learning

Generally speaking, RL methods that use the current learned policy for data collection are referred to as *on-policy* methods, methods that use a fixed separate policy for data collection are referred to as *off-policy.*

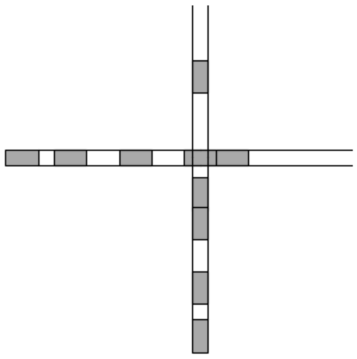

Figure 1: Illustration of a single intersection with vehicles drawn as grey rectangles. Vehicles approach the intersection from the east and from the south and cross it without turning. Note that the first two waiting vehicles on the south lane kept some distance before the intersection, such that they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

2 Problem definition and decomposition

We start by defining the *vehicle trajectory optimization problem* for a single intersection. Consider the single intersection illustrated in Figure 1, which has 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. Each route is followed by a fixed set of vehicles, so we consider a closed system, in the sense that no more vehicles arrive in the future. We identify vehicles by their route and by their relative order on this route, by defining the vehicle index set

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

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_i and width W_i and we define its longitudinal position $x_i(t)$ along its route to be the distance between its front bumper and the start of the lane. The position of a vehicle over time is governed by the well-known *double integrator* model

$$\begin{aligned} \dot{x}_i(t) &= v_i(t), \\ \dot{v}_i(t) &= u_i(t), \\ 0 \leq v_{\max} &\leq v_{\max}, \\ |u_i(t)| &\leq a_{\max}, \end{aligned} \tag{2}$$

where $v_i(t)$ is the vehicle's velocity and $u_i(t)$ its acceleration. Let $D_i(s_{i,0})$ denote the set of all trajectories $x_i(t)$ satisfying these dynamics, given some initial state $s_{i,0} = (x_i(0), v_i(0))$. In order to maintain a safe distance between consecutive vehicle on the same lane, vehicle trajectories need to satisfy the *safe headway constraints*

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

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 $x_i(t) \in \mathcal{E}_i$. Using this notation[1], 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 $r(i) \neq r(j)$, to which we will refer as the set of *conflicts*, denoted by $\mathcal{D}$.

Suppose we have some performance criterion $J(x_i)$ that takes into account travel time and energy efficiency of the trajectory of vehicle i. Suppose there is some central decision making entity that controls the acceleration of all vehicles in the system at once, then the trajectory optimization problem for a single intersection can be compactly written as

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

where all constraints apply at all times 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 $t \in \{0, \Delta t, \ldots, K\Delta t\}$. Following the approach in [1], the collision-avoidance constraints between lanes can be formulated using the well-known big-M technique with binary decision variables, by defining 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 some sufficiently large number. Finally, the safe headway constraints can simply be added as

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

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

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

where v_d is some reference velocity and t_f denotes the final time. This objective can be roughly interpreted as trying to maintain a velocity close to v_d, while simultaneously minimizing the energy consumption required for acceleration and deceleration. For a problem

[1] Note that it would make sense to let B_i be equal for all vehicles i that follow the same route, which we do not emphasize with the notation to keep it simple.

with five identical vehicles with initial conditions as given in Table 1, 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	6	12	10

Table 1: Example of initial states $s_{i,0} = (x_i(0), v_i(0))$ for problem (5) with five identical vehicles.

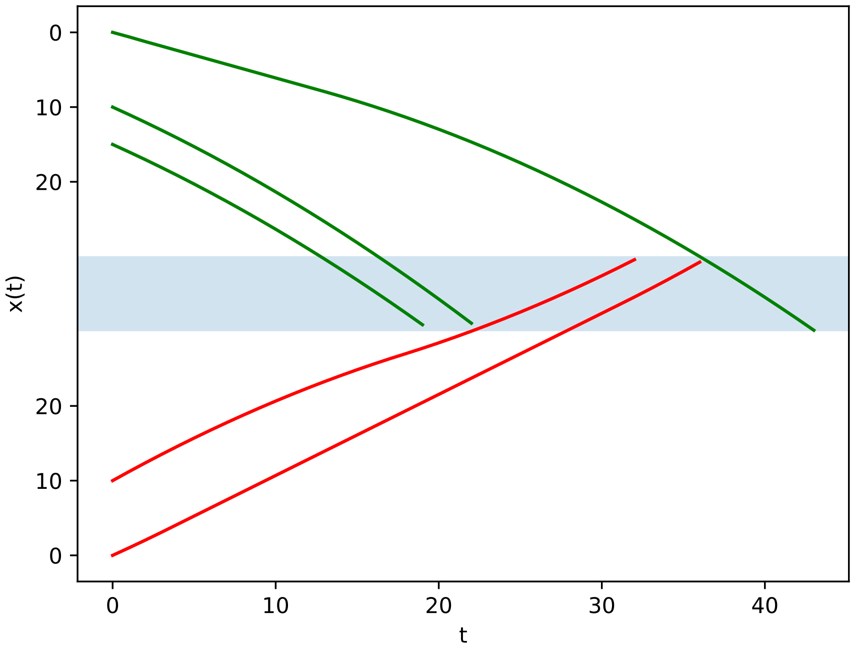

Figure 2: Example of optimal trajectories obtained using the direct transcription method with $L_i \equiv L = 5$, $\mathcal{E}_i \equiv \mathcal{E} = [50, 70]$, $v_d = 20$, $T = 120$, $\Delta t = 0.1$ and initial conditions as given in Table 1. The y-axis is split such that each part corresponds to one 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.

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. The trajectories can be generated separately for each route, which leads to some degree of decomposition of the problem. We will use decision variable y_i for the crossing time and write $y_i + \sigma_i$ for the exit time, so σ_i is the amount of time vehicle i occupies the intersection. Let $\mathcal{N}_r = \{i \in \mathcal{N} : r(i) = r\}$ be all vehicles on route r and write vector $y_r = [y_i : i \in \mathcal{N}_r]$ and similarly define σ_r, x_r and $s_{r,0}$. Using this notation, the problem can be decomposed into the upper-level *crossing time scheduling* problem

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

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

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

Note that the set of feasible parameters $\mathcal{S}_r$ implicitly depends on the initial states $s_{r,0}$ and the global system parameters.

2.3 Decomposition for delay objective

Assume that the trajectory performance criterion is based on the amount of delay experienced by vehicles. Observe that $a_i := (B_i - x_i(0))/v_{\max}$ is the earliest time at which vehicle i can enter the intersection, so we will define the delay of vehicle i to be $J(x_i) = y_i - a_i$. This assumption makes the problem significantly simpler, because we now have

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

Furthermore, we assume that vehicles have full speed initially and when crossing the intersection, so $v_i(0) = v_i(y_i) = v_{\max}$, and we assume that all vehicles are identical, so $L_i = L$ and $W_i = W$ for all $i \in \mathcal{N}$, 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. 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, to which we will refer as the *processing time.* It can now be shown that $y_r \in \mathcal{S}_r$ holds whenever

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

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

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

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. Given optimal crossing time schedule y^*, any set of trajectories $[x_i(t) : i \in \mathcal{N}]$ that satisfies

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

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

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

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

The previous section showed that, given a crossing time schedule y, trajectories can be efficiently computed using a direct transcription method. Hence, we will show how to leverage the branch-and-cut framework to solve the crossing time scheduling problem (6) by formulating it as a Mixed-Integer Linear Program (MILP) and defining three types of cutting planes. We investigate the computational complexity of this approach for different classes of problems in Section 3.2 To illustrate an alternative solution approach, Section **??** briefly discusses a local search heuristic.

3.1 Branch-and-cut

To obtain a MILP, we rewrite 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

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

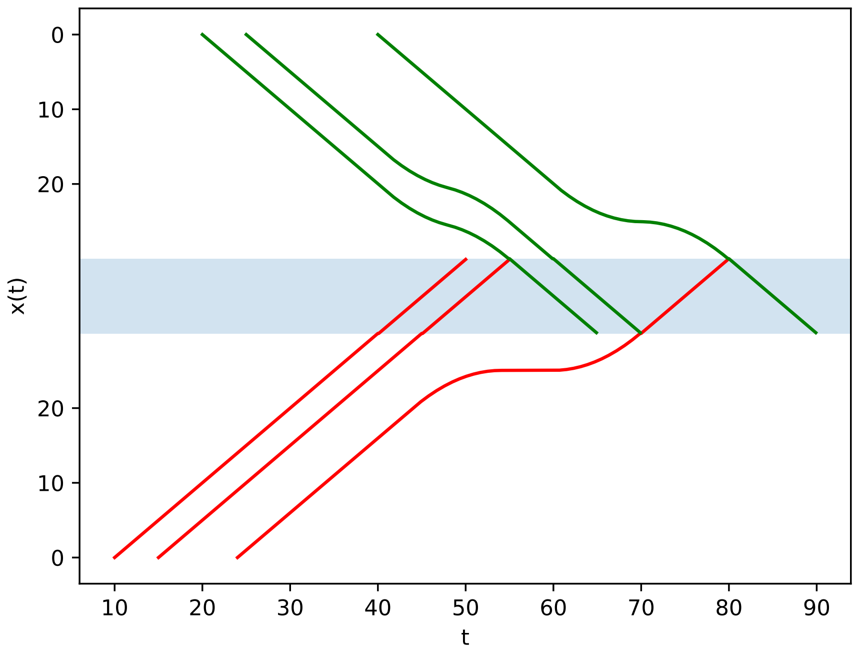

Figure 3: Example trajectories of vehicles from a single route, for some random arrival times and scheduled crossing times, computed by solving the linear program obtained from direct transcription of problem `MotionSynthesize`. We used the parameters $v_{\max} = 1, a_{\max} = 0.5, L = 5$. The "queueing 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.

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

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

where $M > 0$ is some sufficiently large number. Next, we will discuss three 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

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

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

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

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

Apart from the redundancy that stems from the way the conflicts 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 [2], as stated by the following result, whose proof can be found in Appendix A.

Proposition 1 (Platoon Preservation Theorem)**.** *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 $y_{i^*} + \rho = y_{j^*}$.*

We will now use this result to define two types of additional cutting planes. In order to model this 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 1 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 of branch-and-cut

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

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

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

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

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 60 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.

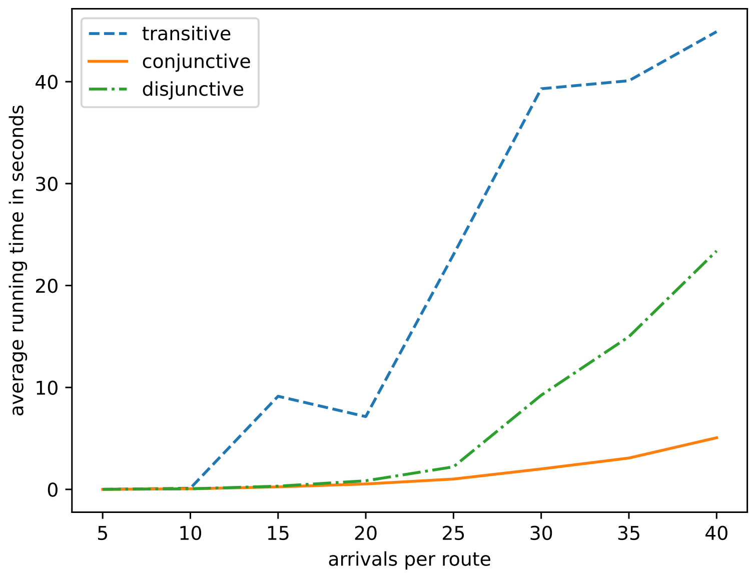

Figure 4: The average (censored) running time of the branch-and-cut procedure is plotted as a function of the number of arriving vehicles per route, for each of the three indicated cutting planes. Each average is computed over 20 problem instances. All instances use $\rho = 4$ and $\sigma = 1$. The arrivals 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.

3.3 Local search

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 *neighboring 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 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*.

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

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	$(0, 1, 1, 0, 0, 0, 1, 1, 1, 0)$	

4 Learning to schedule

Methods that rely on the branch-and-cut framework are guaranteed to find an optimal solution, but as we showed in Section 3.2, the running time scales very poorly when increasing instance sizes. For this reason, we are interested in developing heuristics to obtain good approximations in limited time. Specifically, we will consider heuristics in terms of autoregressive models [3].

Equivalent representations of schedules Observe that the space of feasible solutions of (6) can be reduced to finitely many decisions regarding the disjunctive constraints (6d). Specifically, each feasible schedule y can be written as the solution to some linear program

$$\begin{aligned} \min_{y} \quad & \sum_{i \in \mathcal{N}} y_i \\ \text{s.t.} \quad & a_i \leq y_i && \text{for all } i \in \mathcal{N}, \\ & y_i + \rho \leq y_j, && \text{for all } (i,j) \in \mathcal{C}, \\ & y_i + \sigma \leq y_j, && \text{for all } (i,j) \in \mathcal{O}, \end{aligned}$$

for some selection $\mathcal{O}$ of the disjunctive constraints. To see when such a selection is allowed, we can think of this linear program as a weighted directed graph on nodes $\mathcal{N}$ with conjunctive arcs $(i,j) \in \mathcal{C}$ with weights $w(i,j) = \rho$ and disjunctive arcs $(i,j) \in \mathcal{O}$ with weights $w(i,j) = \sigma$. Now observe that the definition of y through the linear program is equivalent to defining y through

$$y_j = \max\{a_j, \max_{i \in \mathcal{N}^-(j)} y_i + w(i,j)\}, \tag{7}$$

where $\mathcal{N}^-(j)$ denotes the set of in-neighbors of node j. It is now easy to see that y is well-defined if and only if the so-called *complete disjunctive graph* $G = (\mathcal{N}, \mathcal{C} \cup \mathcal{O})$ is acyclic. When G is acyclic, there is a unique topological ordering of its nodes. Hence, any valid selection $\mathcal{O}$ of disjunctive constraints is equivalent to a sequence π of vehicles. It is also equivalent to a sequence η of routes, because the ordering of vehicles on the same route is already fixed. From now on, we will mainly be working with η, because it is in some sense the simplest representation of a schedule and we write y^η to denote the induced crossing times.

Autoregressive modeling of schedules Given some problem instance s, we now want to model the sequence η such that y^η is an optimal schedule. To this end, we model the conditional probability of η given s by considering autoregressive models of the form

$$p(\eta | s) = \prod_{t=1}^{N} p(\eta_t | s, \eta_{1:t-1}), \tag{8}$$

where N denotes the total number of vehicles. Now consider some distribution of problem instances $\mathcal{X}$, then our goal is to minimize the expected value

$$\mathbb{E}_{s\sim\mathcal{X},\eta\sim p(\eta|s)}L(s,\eta),$$

of the total vehicle delay

$$L(s,\eta)=\sum_{i\in\mathcal{N}}y_i^\eta-a_i.$$

For inference, we would ideally want to calculate the maximum likelihood estimator

$$\arg\max_\eta p(\eta|s),$$

but this is generally very expensive to compute, because this could require $O(|\mathcal{R}|^N)$ evaluations of $p(\eta_t|s,\eta_{1:t-1})$ when we do not make additional structural model assumption. Therefore, one often uses *greedy rollout*, which means that we pick η_t with the highest probability at every step. Other inference strategies have been proposed in the context of modeling combinatorial optimization problems, see for example the "Sampling" and "Active Search" strategies in the seminal paper [4].

4.1 Model parameterization

We can consider different ways of parameterizing $p(\eta|s)$ in terms of $p(\eta_{t+1}|s,\eta_{1:t})$. Here, each partial sequence $\eta_{1:t}$ represents some partial schedule, which can equivalenty be defined in terms of the sequence of *scheduled* vehicles $\pi_{1:t}$. However, it is more convenient to define a partial schedule in terms of the *partial disjunctive graph* $G_t=(\mathcal{N},\mathcal{C}\cup\mathcal{O}_t)$, which is defined by the unique selection $\mathcal{O}_t$ such that for each $i\in\pi_{1:t}$ and each $\{i,j\}\in\mathcal{D}$, we have either $(i,j)\in\mathcal{O}_t$ or $(j,i)\in\mathcal{O}_t$ with $j\in\pi_{1:t}$. To emphasize this parameterization in terms of the partial disjunctive graph, we can alternatively write (8) as

$$p(\eta|s)=\prod_{t=1}^N p(\eta_t|G_{t-1}).$$

There is a natural extensions of expression (7) for partial disjunctive graphs. Given G_t, let the *earliest crossing time* of each vehicle $i\in\mathcal{N}$ be recursively defined as

$$\beta_t(j)=\max\{a_j,\max_{i\in\mathcal{N}_t^-(j)}\beta_t(i)+w(i,j)\},$$

where $\mathcal{N}_t^-(j)$ denotes the set of in-neighbors of node j in G_t. For empty schedules, we have $\beta_0(i)=a_i$ for all i. For complete schedules, we have $\beta_N(i)=y^\eta(i)$ for all i. We have $\beta_t(i)\leq\beta_{t+1}(i)$ for all i, because $\mathcal{O}_t\subset\mathcal{O}_{t+1}$. Hence, β_t can be interpreted as providing the best lower bounds $\beta_t(i)\leq y^\eta(i)$, regardless of how the partial schedule is completed.

4.1.1 Threshold heuristic

We know from Proposition 1 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 heuristic based on this idea.

In this case, the model does not specify a distribution over η, but selects a single candidate by selecting a single next route in each step, so with $\mathbb{1}\{\cdot\}$ denoting the indicator function, we have $p(\eta_{t+1}|G_t)=\mathbb{1}\{\eta_{t+1}=p_\tau(G_t)\}$, selecting the next route at step t as

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

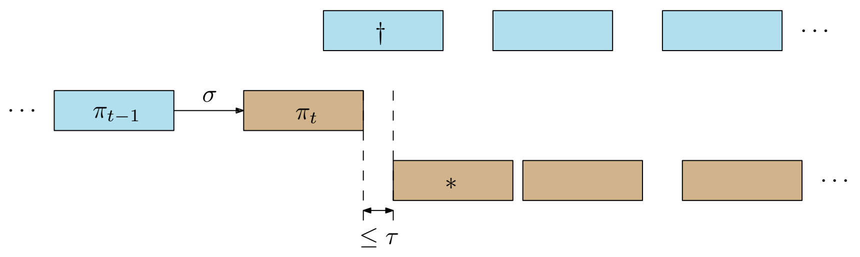

Figure 5: Illustration of how the threshold heuristic is evaluated at some intermediate step t to choose the next route η_{t+1}. The top and bottom row contains the unscheduled vehicles from route 1 and route 2, respectively, drawn at their earliest crossing times $\beta_t(i)$. The middel row represents the current partial schedule. Vehicle π_{t-1} is from route 1 and the last scheduled vehicle π_t is from route 2 and the disjunctive constraint for them happens to be tight in this case, illustrated by the arrow. Whenever the indicated distance is smaller than τ, the threshold rule selects vehicle $*$ to be scheduled next. Otherwise, vehicle † will be chosen.

where $\texttt{next}(\eta_t)$ denotes some arbitrary route other than η_t with unscheduled vehicles left. This definition is illustrated in Figure 5.

4.1.2 Neural heuristic

We will now consider a parameterizaton that directly generalizes the threshold heuristic. Instead of looking at the earliest crossing time of the next vehicle in the current lane, we now consider the earliest crossing times of all unscheduled vehicles across lanes. In the following definitions, we will drop the step index t to avoid cluttering the notation. For every route r, let π^r denote the sequence of unscheduled vehicles and consider their crossing time lower bounds $\beta(\pi^r) = (\beta(\pi^r_1), \beta(\pi^r_2), \dots)$. Let the minimum crossing time lower bound among unscheduled vehicles be denoted by T, then we call $h_r = \beta(\pi^r) - T = (\beta(\pi^r_1) - T, \beta(\pi^r) - T, \dots)$ the *horizon* of route r.

Next, we define some neural embedding $\bar{h}_r$ of each horizon. Observe that horizons can be variable length. We could fix the length by using padding, but this can be problematic for states that are almost done. Therefore, we employ a recurrent neural network. 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. These horizon embeddings are arranged into a vector h_t by the following cycling rule. At position k of vector h_t, we put the embedding for route

$$k - \eta_t \bmod |\mathcal{R}|$$

where η_t denotes the last selected route. Using this cycling, we make sure that the embedding of the last selected route is always kept at the same position of the vector. Using some fully connected neural network f_θ and a softmax layer, this global embedding is then finally mapped to a probability distribution as

$$p_\theta(\eta_{t+1}|G_t) = \text{softmax}(f_\theta(h_t)),$$

where θ denotes the parameters of f and of the recurrent neural networks. After θ has been determined, we can apply greedy rollout by simply ignoring routes that have no unscheduled vehicles left and take the argmax of the remaining probabilities.

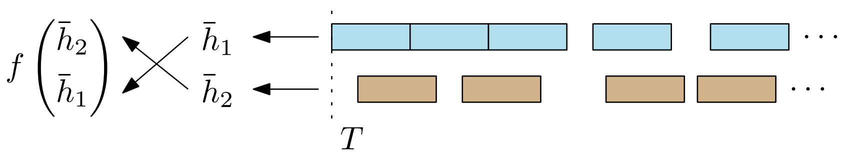

Figure 6: Schematic overview of parameterization of the neural heuristic. The distribution over the next route is parameterized as $p_\theta(\eta_{t+1}|G_t)$, which is computed from the current route horizons h_r via embeddings $\bar{h}_r$ and the cyling trick. Observe that the particular cycling shown in the figure corresponds to a situation in which the current route is $\eta_t = 2$.

4.2 Supervised learning

We will now explain how model parameters can be tuned using some sample of problem instances $X \sim \mathcal{X}$. Given some instance s, let $\eta_\tau(s)$ be the schedule produced by the threshold heuristic. Note that $L(s, \eta^\tau(s))$ is not differentiable with respect to τ, so we cannot use gradient-based optimization methods. However, we only have a single parameter, so we can simply select the value of τ that minimizes the average empirical loss

$$\min_{\tau \geq 0} \sum_{s \in X} L(s, \eta_\tau(s)),$$

which can be computed through a simple grid search. This approach is no longer possible when $p(\eta\,|s)$ is a proper distribution, so we need to something different for the neural parameterization.

Consider some instance $s \in X$ and let η^* denote some optimal route sequence, which can for example be computed by solving the MILP from Section 3. For each such optimal schedule, we can compute the sequence $G_0, \eta_1, G_1, \eta_2, \dots, \eta_N, G_N$. The resulting set of pairs $\{(G_t, \eta_{t+1}) : t = 1, \dots, N-1\}$ can be used to learn p_θ in a supervised fashion by treating it as a classification task and computing the maximum likelihood estimator $\hat{\theta}$. Interpreting G_t as *states* and η_{t+1} as *actions*, we see that this approach is equivalent to *imitation learning* in the context of finding policies for Markov decision processes from so-called *expert demonstration*. Let Z denote the set of all state-action pairs collected from all training instances X. We make the procedure concrete for the case of two routes $\mathcal{R} = \{1, 2\}$, which is slightly simpler. Let $p_\theta(G_t)$ denote the probability of choosing the first route, then we can use the binary cross entropy loss, given by

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

where $\mathbb{1}\{\cdot\}$ denotes the indicator function. Now we can simply rely on some gradient-descent optimization procedure to minimize $L_\theta(Z)$ with respect to θ.

4.3 Reinforcement learning

Instead of using state-action pairs as examples to fit the model in a supervised fashion (imitation learning), we can also choose to use the reinforcement learning paradigm, in which the data collection process is guided by some policy.

The reinforcement learning approach depends on the definition of a reward. For each step $G_{t-1} \xrightarrow{\eta_t} G_t$, we can define a corresponding reward, effectively yielding a deterministic Markov decision process. Specifically, we define the reward at step t to be

$$R_t = \sum_{i \in \mathcal{N}} \beta_{t-1}(i) - \beta_t(i).$$

Let the return at step t be defined as

$$\hat{R}_t = \sum_{k=t+1}^{N} R_k.$$

Hence, when the *episode* is done after N steps, the total episodic reward is given by the telescoping sum

$$\hat{R}_0 = \sum_{t=1}^{N} R_t = \sum_{i \in \mathcal{N}} \beta_0(i) - \beta_N(i) = \sum_{i \in \mathcal{N}} a_i - y_i = -L(s, \eta),$$

Therefore, maximizing the episodic reward corresponds to minimizing the scheduling objective, as desired.

Policy-based methods work with an explicit parameterization of the policy. The model parameters are then tuned based on experience, often using some form of (stochastic) gradient descent to optimize the expected total return. Therefore, the gradient of the expected return plays a central role. The following identity is generally known as the Policy Gradient Theorem:

$$\begin{aligned}
\nabla \mathbb{E}_p \hat{R}_0 &\propto \sum_s \mu_p(s) \sum_a q_p(s, a) \nabla p(a|s, \theta) \\
&= \mathbb{E}_p \left[\sum_a q_p(S_t, a) \nabla p(a|S_t) \right] \\
&= \mathbb{E}_p \left[\sum_a p(a|S_t) q_p(S_t, a) \frac{\nabla p(a|S_t)}{p(a|S_t)} \right] \\
&= \mathbb{E}_p \left[q_p(S_t, A_t) \frac{\nabla p(A_t|S_t)}{p(A_t|S_t)} \right] \\
&= \mathbb{E}_p \left[\hat{R}_t \log \nabla p(A_t|S_t) \right].
\end{aligned}$$

The well-known REINFORCE estimator is a direct application of the Policy Gradient Theorem. At each step t, we update the parameters θ using a gradient ascent update

$$\theta \leftarrow \theta + \alpha \hat{R}_t \nabla \log p_\theta(\eta_t | G_t),$$

with some fixed learning rate α. To reduce variance of the estimator, we can incorporate a so-called *baseline*, which is an estimate of the expected return of the current state. In the context of combinatorial optimization, the value of the baseline may be interpreted as estimating the relative difficulty of an instance?

4.4 Results

The evalutation of model performance 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 here, because we have both training time as well as inference time for autoregressive models. We study the effect of the problem instance distribution $\mathcal{X}$ by varying the number of routes and number of arrivals per route, distribution of interarrival times, arrival intensity per route and degree of platooning.

Let $N(s)$ denotes the total number of vehicles in instance s. To enable a fair comparison across instances of various sizes, we report the quality of a solution in terms of the average delay per vehicle $L(\eta, s)/N(s)$. Given some problem instance s, let η^* denote the schedule computed using branch-and-bound. We use a fixed time limit of 60 seconds per instance for

Table 3: Performance evaluation of the branch-and-cut (MILP) approach and the threshold heuristic for different classes of instances with two routes. The first two columns specify the instance class based on the number of vehicles n per route and the type of arrival distribution for each route. These arrival distributions are chosen such that the arrival intensity is the same, only the degree of platooning varies. Performance is measured in terms of $L(s, \eta)/N(s)$, averaged over 100 test instances. The optimality gap is shown in parentheses for the heuristics. The threshold heuristic is fitted based on 100 training instances and the training time is indicated. For branch-and-cut the average inference time is indicated. Note that we used a time limit of 60 seconds for all the branch-and-cut computations.

n	type	MILP	time	exhaustive (gap)	threshold (gap)	time
10	low	5.29	0.05	9.49 (79.45%)	7.78 (47.08%)	3.87
30	low	8.60	2.01	12.72 (47.88%)	11.31 (31.49%)	21.43
50	low	11.03	13.78	16.56 (50.20%)	14.60 (32.41%)	52.13
10	med	4.46	0.06	7.20 (61.32%)	6.39 (43.15%)	3.86
30	med	6.99	1.88	9.43 (34.97%)	8.96 (28.29%)	21.59
50	med	8.55	15.11	11.50 (34.40%)	10.77 (25.93%)	52.23
10	high	4.47	0.06	6.35 (42.04%)	5.70 (27.51%)	3.95
30	high	6.90	1.90	8.92 (29.19%)	8.58 (24.25%)	21.67
50	high	7.37	14.99	9.36 (26.94%)	8.88 (20.42%)	52.13

the branch-and-bound procedure, in order to bound the total analysis time. Therefore, it might be that η^* is not really optimal for some of the larger instances. Given some possibly suboptimal schedule η, we define its *optimality gap* as

$$L(s, \eta)/L(s, \eta^*) - 1.$$

For each heuristic, we report the average optimality gap over all test instances.

The performance of the threshold heuristic is evaluated based on optimal solutions obtained using MILP in Table 3. With the specific choice $\tau = 0$, the threshold rule is related to the so-called *exhaustive policy* for polling systems, which is why we consider this case separately. We plot the average objective for the values of τ in the grid search, see Figure 10.

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. The results are listed in Table **??**. For the neural heuristic, we plot the training and validation loss, see Figure 11. It can be seen that the model converges very steadily in all cases.

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.

[3] J. M. Tomczak, *Deep Generative Modeling.* Cham: Springer International Publishing, 2024.

[4] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural Combinatorial Optimization with Reinforcement Learning," Jan. 2017.

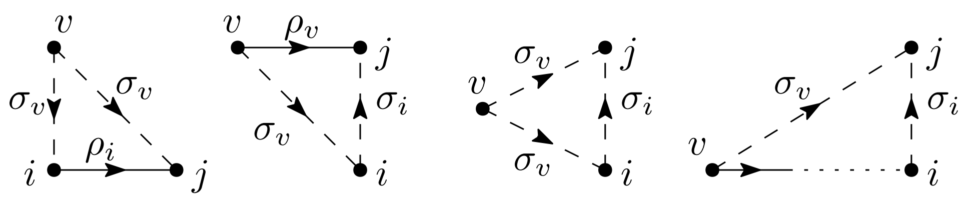

Figure 7: Sketch of the four cases distinguished in the proof of Lemma 1. Arc weights are given and disjunctive arcs $\mathcal{O}(\pi)$ are drawn with a dashed line.

A Proof of Proposition 1

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*

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

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

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

so that $\beta_\pi(v) + w(v,j) \leq \beta_\pi(i) + w(i,j)$ 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

$$\beta_\pi(v) + w(v,j) = \beta_\pi(v) + \rho_v \leq \beta_\pi(v) + \sigma_v + \sigma_i \leq \beta_\pi(i) + w(i,j),$$

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

$$\beta_\pi(v) + w(v,j) = \beta_\pi(v) + w(v,i) \leq \beta_\pi(i) \leq \beta_\pi(i) + w(i,j).$$

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

$$\beta_\pi(v) + w(v,j) = \beta_\pi(v) + \rho_v + s \leq \beta_\pi(i) + s \leq \beta_\pi(i) + w(i,j).$$

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

Proposition 1 (Platoon Preservation Theorem)**.** *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 $y_{i^*} + \rho = y_{j^*}$.*

Proof. Suppose the ordering π of y is such that $\pi^{-1}(i^*) + 1 < \pi^{-1}(j^*)$. Let $\mathcal{I}(i,j) = \{i, \pi(\pi^{-1}(i)+1), \ldots, j\}$ 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 **??** 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 **??** and Lemma 1 follows that $y_{i^*} + \rho = y_{j^*}$. □

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

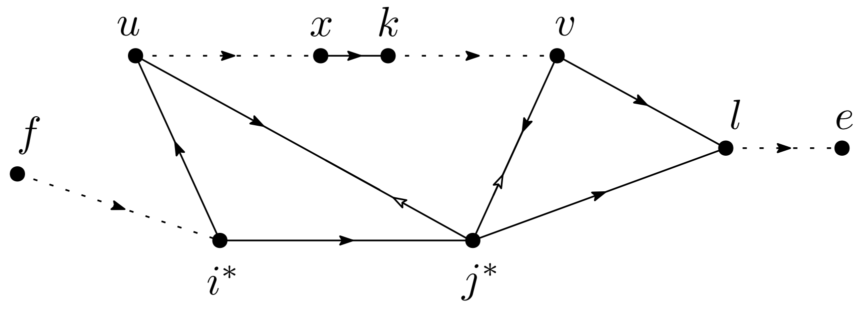

Figure 8: Sketch of the nodes and most important arcs used in the proof of Proposition 1. Dashed arcs represent chains of unspecified length. The two open arrows indicate the new direction of their arc under ordering π'.

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 4.1.2. Note that all three of the above functions are fixed, in the sense that they do not rely on any learnable parameters. The reason for treating these fixed parts of the model separately is that prevents us from recomputing these transformations, because it allows us to store the transformed state-action pairs to disk to use across different training runs. Figure 9 shows an overview of the state and action transform functions and how they are related to each other. 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`.

$$\begin{tikzcd}
 & & \texttt{obs} \arrow[d, "\texttt{model()}"'] & & \texttt{automaton} \arrow[ll, "\texttt{state\_transform()}"'] \\
\texttt{target} \arrow[dr, dashed] & & \texttt{logit} \arrow[dl, dashed] \arrow[rr, shift left, "\texttt{action\_transform()}"] & & \texttt{action} \arrow[ll, shift left, "\texttt{inverse\_action\_transform()}"] \\
 & \texttt{loss} & & &
\end{tikzcd}$$

Figure 9: Schematic of the state and action transform functions used for imitation learning. Given some optimal state-action training pair, the `target` logit is computed from the optimal action using the inverse action transform.

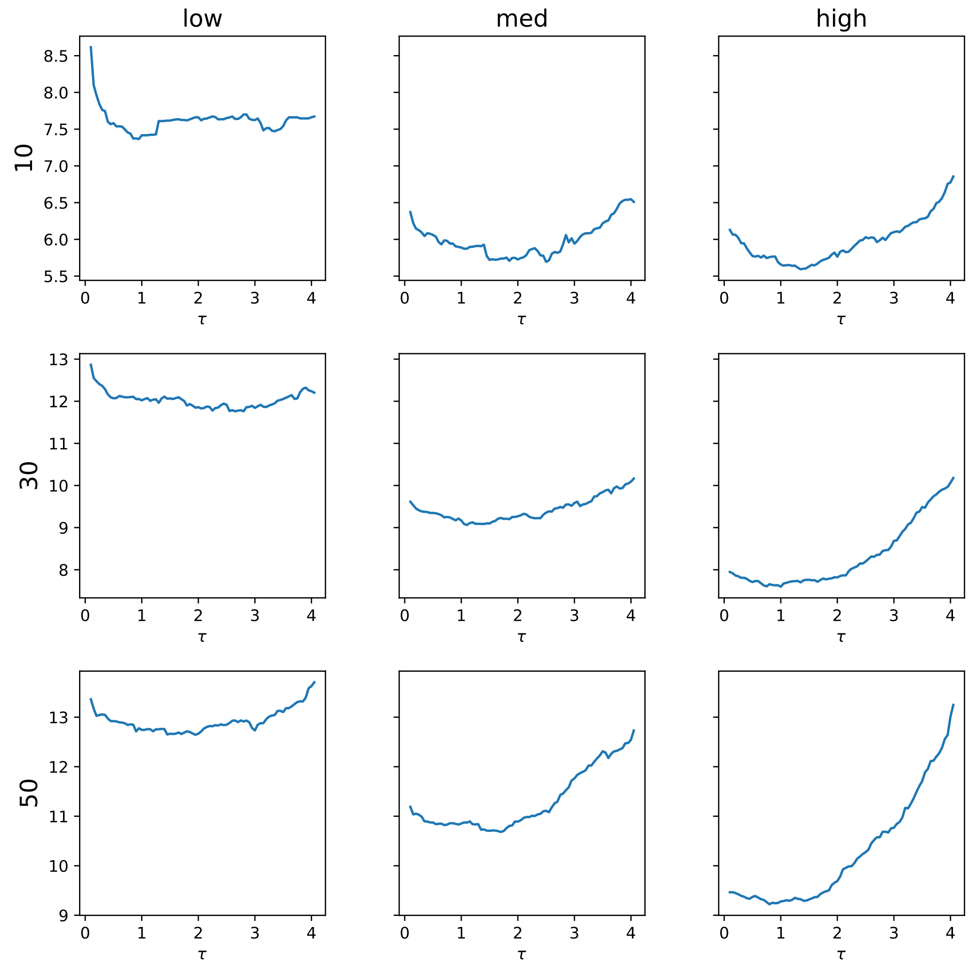

Figure 10: Charts to verify threshold heuristic model fit. For each indicated instance training set, the average delay $\sum_{s\in\mathcal{X}} \mathrm{obj}(y_\tau(s))\,/\,|\mathcal{X}|$ is plotted as a function of the threshold τ. We use these plots to verify that the range of possible candidates for 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 $\sigma = 1$.

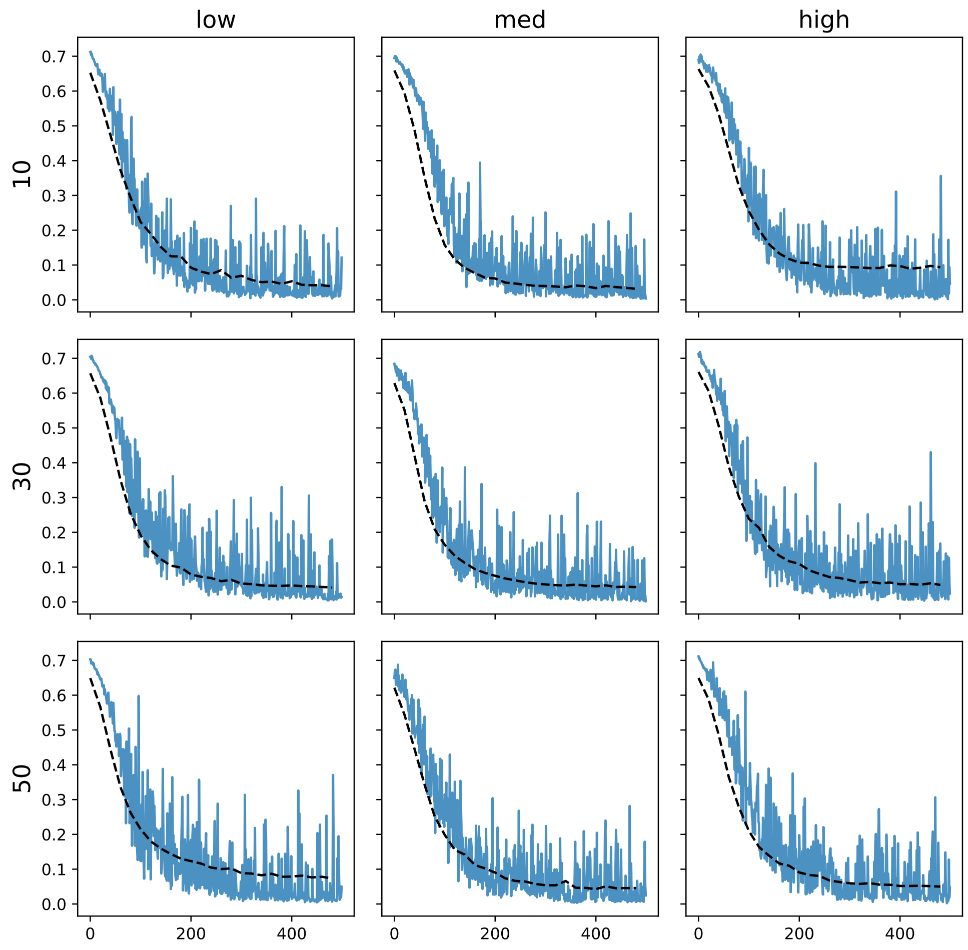

Figure 11: Loss plots to verify neural heuristic model fit. For each indicated instance training set, the training loss is plotted for each training step in blue and the validation loss is plotted as the dashed line. Recall that the validation loss is the average binary cross entropy loss after a given number of training steps. The training loss is plotted for each individual step without smooting.