Traffic Scheduling

Jeroen van Riel

February 2024

1 Single Intersection

Consider a single isolated intersection with vehicles arriving from different lanes. Vehicles arrive at the system at some fixed distance h from the intersection at some moment in time a_j. The earliest time at which a vehicle j can reach the conflict zone, assuming that the vehicle can drive at maximum speed $v_{\max}$ and does not encounter other vehicles, is denoted by $r_j = a_j + h/v_{\max}$. It would be great if we could allow each vehicle j to arrive at this *earliest crossing time*. In general, however, vehicles need to be delayed by some time $d_j \geq 0$ in order to meet safety constraints. We consider the following two types of constraints.

Let the actual *crossing time* of vehicle j be denoted by $y_j = r_j + d_j$. Consider a pair j, l of consecutive vehicles on the same lane and assume j drives before l, which means that $r_j < r_l$. We assume that vehicles on the same lane do not *overtake* each other, so we require $y_j < y_l$. Even stronger, in order to guarantee safety, we require a minimum amount of time p_j after the moment vehicle j crosses, which may depend on the particular vehicle, e.g., a long truck needs more time than a short passenger car. The first type of constraint is given by

$$y_j + p_j \leq y_l. \tag{1}$$

Now consider a pair of vehicles j, l originating from distinct lanes. In order to guarantee safety, we require an additional minimum amount of time between the crossing moments. In general, this time may depend on the particular pair of vehicles, so the second type of constraint is given by

$$y_j + p_j + s_{jl} \leq y_l \quad \text{or} \quad y_l + p_l + s_{lj} \leq y_j. \tag{2}$$

This contraint features a choice, because the order of the two vehicles is left to the traffic controller to decide.

It is the task of the traffic controller to determine y that satisfies these constraints, while optimizing some kind of performance metric. We first need to make assumptions on information that is available to the traffic controller. In the most straightforward case, the traffic controller is given full information about a finite number of arrivals, to which we will refer as *offline optimization*. It is also possible to assume that arrival times a_j become gradually known to the

traffic controller as time passes. We will not consider such an *online optimization* setting.

In the context of offline optimization, we can simply define an objective in terms of the y, which is a finite vector in this case. The difference $y_j - r_j = d_j$ between the earliest crossing time and the actual crossing time defines the *delay* experienced by vehicle j. A sensible choice for the optimization objective is to minimize the *total delay*

$$\sum_j d_j = \sum_j y_j - r_j. \tag{3}$$

The total delay objective does not take into account issues of fairness. This means that optimal solutions are not guaranteed to have an even distribution of delay among vehicles, which may not be desirable in practice. In online optimization, these issues can become even more pronounced, as simple examples are easily constructed in which it is optimal to have at least a single vehicle wait indefinetely.

Note that determining y does not determine the exact trajectories of the vehicles, i.e., the continuous control of the speed of each vehicle is not yet determined. For the current project, we assume that a safe trajectory can be computed from y by applying an appropriate Speed Profile Algorithm (SFA), as discussed in [1]. Under this assumption, it turns out that the problem can be adequately stated in terms of a single machine scheduling problem. Therefore, we will first introduce some basic modeling elements from the scheduling literature, before restating the problem in this framework.

1.1 Single Machine Scheduling

Suppose we have n jobs that need processing on a single machine. The machine can process at most one job at the same time. The time required for each job $j \in \{1, \ldots, n\}$ is called the *processing time* p_j. Once started, jobs may not be *preempted*, which means that they occupy the machine for exactly p_j units of time. By determining the start times $y_j \geq 0$ for each job j, such that at most one job is processing on the machine at all times, we obtain a feasible *schedule*.

Let $C_j = y_j + p_j$ denote the *completion time* of job j. A common objective is to minimize the *total completion time*

$$\sum_{j=1}^{n} C_j.$$

By means of an *interchange argument*, it can be easily shown that, for this objective, an optimal solution is given by sorting the jobs according to non-decreasing processing times, which is known as the Shortest Processing Time first (SPT) rule [2].

Now suppose that job j becomes available at its *release date* r_j (sometimes also written as $r(j)$), then a valid schedule y requires $y_j \geq r_j$. It is not difficult

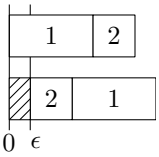

Figure 1: Illustration of the two possible schedules for Example 1.1. The first schedule is a non-delay schedule. The unforced idle time in the second schedule is shaded.

to see that this might mean we need to introduce some *idle time*. We speak of *unforced idle time* whenever the machine is kept idle at time t while at least one job j is already available for processing, i.e., $t \geq r_j$. Schedules without such unforced idle time are called *non-delay*.

Example 1.1. Consider $n = 2$ jobs with processing times $p_1 = 2, p_2 = 1$ and release dates $r_1 = 0, r_2 = \epsilon > 0$. Processing job 1 before job 2 results in a schedule with total completion time $\sum C_j = 2 + 3 = 5$. However, scheduling job 2 first requires us to introduce ϵ idle time, but for $\epsilon < 1/2$, it has a better total completion time of $\sum C_j = (\epsilon + 1) + (\epsilon + 1 + 2) = 4 + 2\epsilon$. See Figure 1 for an illustration of both schedules. □

Next, we consider *precedence constraints* between jobs, which can be used to fix a certain order among jobs. Suppose that job j needs to be processed before job l, denoted as $j \to l$, then we simply require that $y_l \geq C_j$ in any feasible schedule. In particular, we may consider *chains* of precedence constraints. Let $F_1, \ldots, F_K$ be a partition of jobs into K non-empty families. For each family F_k, we require that their jobs $F_k = \{J_k(1), \ldots, J_k(n_k)\}$ are processed in the order

$$J_k(1) \to J_k(2) \to \cdots \to J_k(n_k),$$

such that $J_k(m)$ denotes the m'th job of family k. With a little abuse of notation, we will use

$$F_k = (J_k(1), \ldots, J_k(n_k))$$

to immediately fix the order of a family. Note that the order between jobs from different families is unspecified, which means that the scheduler may decide to *merge* these chains arbitrarily.

When job j is directly followed by job l, we might want to introduce some *sequence-dependent setup time* $s_{jl} \geq 0$, by requiring that $y_l \geq C_j + s_{jl}$. For our purposes, we will only consider setup times depending on the family to which jobs belong. For $k_1 \neq k_2$ and any pair of jobs $j_1 \in F_{k_1}$, $j_2 \in F_{k_2}$, either the constraint

$$C_{j_1} + s_{k_1,k_2} \leq y_{j_2}$$

is satisfied or

$$C_{j_2} + s_{k_2,k_1} \leq y_{j_1}.$$

1.2 Problem Formulation

Using the modeling elements introduced above, we now pose the traffic control problem as a single machine scheduling problem. The intersection is represented as the single machine. Let each of the n vehicles be represented by some job j with release date r_j. The starting time y_j models the planned crossing time of vehicle j, which needs to be determined by the traffic controller. We will now show how the two types of constraints are formulated.

Let n_k denote the total number of vehicles arriving to lane k. For each lane, we define a family F_k consisting of all vehicles that arrive to it. We assume, without loss of generality, that the jobs are ordered according to non-decreasing release dates. Therefore, the first type of constraints (1) for each lane is modeled by a chain of precedence constraints on each family by assuming that

$$r(J_k(n)) < r(J_k(m)) \implies n < m,$$

for any pair of jobs $J_k(n), J_k(m) \in F_k$.

It is not difficult to see that the second set of constraints (2) for vehicles from different lanes can be directly modeled using setup times s_{k_1,k_2}. In what follows, we will also refer to this setup time as *switch-over* time, because it is similar to the time between switching *phases* in convential traffic lights.

Because the release dates r_j and processing time p_j are assumed to be known a priori, observe that minimizing the total delay (3) experienced by all vehicles is equivalent to minimzing the total completion time, so this is the objective that we will use from here one.

Definition 1.1 (Single Intersection Scheduling Problem)**.** *The single machine scheduling problem with release dates, job families with chain precedence constraints, family-dependent setup times and total completion time objective is called the* single intersection scheduling problem*. In the three-field notation [3], we will denote it as* $1 \,|\, r_j, fmls, chains, s_{gh} \,|\, \sum C_j$.

When vehicles turn at the intersection, the safe switch-over time between vehicles depends on the involved lanes, because some turns take more time to complete than others, see Figure 2 for an illustration. For the moment, we will assume that vehicles cross the intersection without turning, which allows us to make the following assumption.

Assumption 1.1. *The processing times* $p_j = p$ *and setup times* $s_{k_1,k_2} = s$ *are all identical, for* $j \in \{1, \ldots, n\}$ *and any pair of lanes* k_1 *and* k_2.

The chain precedence constraints require a minimum of p time units between the crossing times of two jobs from the same lane. This give rise to an additional assumption that can be made on the specification of the release dates.

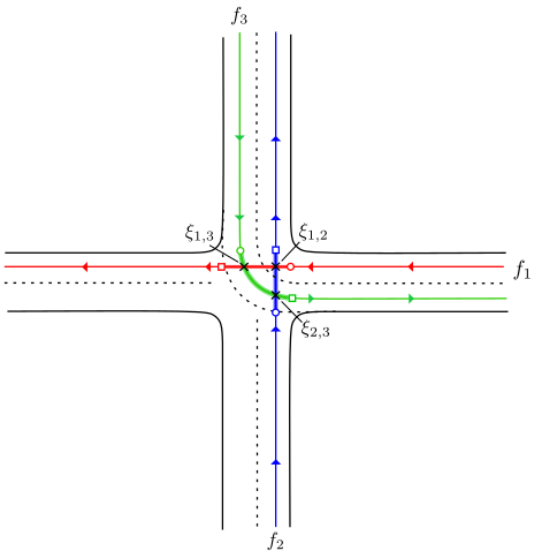

Figure 2: Illustration of how lane-switching constraints need to be adapted in case of vehicle turning (taken from [4] with permission of the author).

Assumption 1.2. *For each job family $F_k = \{J_k(1), \ldots, J_k(n_k)\}$, the release dates satisfy $r(J_k(m)) \geq r(J_k(m-1)) + p$, for all $m \in \{2, \ldots, n_k\}$, without loss of generality.*

1.2.1 Mixed-Integer Linear Progam

A common method of solving combinatorial problems in general and scheduling problems in particular is by formulating a Mixed-Integer Linear Program (MILP) and solving it using branch-and-bound methods. Let $\mathcal{C}$ denote the set of all pairs of jobs (j, l) involved in single precedence constraints $j \to l$, collected from all family chains. Furthermore, let

$$\mathcal{D} = \{\{j, l\} : j \in F_{k_1}, l \in F_{k_2}, k_1 \neq k_2\} \tag{4}$$

denote all pairs of jobs from distinct families, to which we will refer as *conflict pairs*. The problem from Definition 1.1 under Assumptions 1.1 and 1.2 can be stated as

$$\text{minimize} \quad \sum_{j=1}^{n} C_j \tag{5a}$$

$$\text{s.t.} \quad r_j \leq y_j \qquad \text{for all } j = 1, \ldots, n, \tag{5b}$$

$$C_j \leq y_l \qquad \text{for all } (j, l) \in \mathcal{C}, \tag{5c}$$

$$C_j + s \leq y_l \text{ or } C_l + s \leq y_j \qquad \text{for all } \{j, l\} \in \mathcal{D}. \tag{5d}$$

Observe that the above so-called *natural formulation* is not yet a proper MILP, because of disjunctive constraints (5d). One way to resolve this issue is to explicitly encode the ordering decisions. For each $\{j, l\} \in \mathcal{D}$, we could introduce a binary decision variable o_{jl}. A value of $o_{jl} = 0$ indicates that vehicle j goes before vehicle l and a value of $o_{jl} = 1$ indicates that l goes before j. However, note that this introduces unnecessary redundancy into the formulation, because $o_{jl} = 1 - o_{lj}$ always hold. Therefore, we only need one variable for each (unordered) pair of vehicles, so we make an explicity selection by defining

$$\bar{\mathcal{D}} = \{(j, l) : \{j, l\} \in \mathcal{D}, j < l\}. \tag{6}$$

Let $M > 0$ denote a sufficiently large constant (which may depend on the current problem instance), then using the well-known *big-M method*, we transform the above natural formulation into

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} C_j && && \text{(7a)}\\
\text{s.t.} \quad & r_j \le y_j && \text{for all } j = 1, \dots, n, && \text{(7b)}\\
& C_j \le y_l && \text{for all } (j, l) \in \mathcal{C}, && \text{(7c)}\\
& C_j + s \le y_l + o_{jl} M && \text{for all } (j, l) \in \bar{\mathcal{D}}, && \text{(7d)}\\
& o_{jl} \in \{0, 1\} && \text{for all } (j, l) \in \bar{\mathcal{D}}, && \text{(7e)}
\end{aligned}$$

which is now a proper MILP.

1.2.2 Maximum vehicle delay

Suppose we require that vehicles may not be delayed for longer than a fixed maximum time $d_{\max} \ge 0$. This means that the delay must satify $y_j - r_j \le d_{\max}$, which implies that $r_j \le y_j \le r_j + d_{\max}$. A potential benefit of introducing this maximum delay is that it could reduce the number of conflicts between vehicles, which could make solving the MILP easier. However, it can also cause the problem to become infeasible.

Consider a pair of conflicting vehicles j, l and assume $r_j \le r_l$, without loss of generality. Observe that for $d_{\max} < r_l - r_j - p - s$, we do not need to worry about the conflict anymore, because $C_j + s = y_j + p + s \le r_j + d_{\max} + p + s < r_l \le y_l$, so the disjunctive constraint in (19c) can only hold for $o_{jl} = 0$. In other words, the order of j and l is already fixed, because the schedule is infeasible otherwise. Therefore, there is no real conflict between vehicles j and l, so $\{j, l\} \notin \mathcal{D}$.

1.2.3 Complexity

The problem of minimizing the total completion time with release dates (written as $1 \,|\, r_j \,|\, \sum C_j$ in the three-field notation [3]) has been long known to be NP-hard [5], so there is no hope in finding a polynomial algorithm for the current more general problem unless $\mathcal{P} = \mathcal{NP}$. Therefore, we must resort to exhaustive

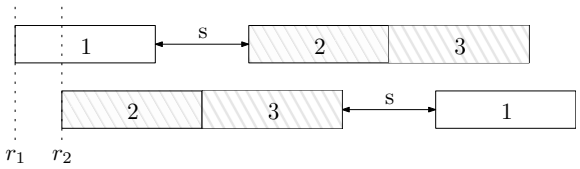

Figure 3: Illustration of the two possible sequences in Example 1.3.

branch-and-bound methods or heuristics. However, note that this only holds for the problem in its full generality. It might be worthwhile to further study the parameterized complexity, i.e., the complexity of the problem as a function of one or more of its parameters [6]. Specifically, it would be interesting to verify the complexity in case of a fixed number of lanes.

1.3 Problem structure

Let us consider a couple of simple examples that might help discover some structure in the problem.

Example 1.2. Consider the situation with $F_1 = \{1\}, F_2 = \{2\}$. When both vehicles have the same release dates $r_1 = r_2 = r$, the order in which they cross the intersection does not influence the optimal total completion time of $\sum C_j = p+(p+s+p) = 3p+s$. Now, assume that vehicle 1 has a strictly earlier release date, then it is easily seen that vehicle 1 must go first in the optimal schedule. □

Example 1.3. Consider the situation with $F_1 = \{1\}, F_2 = (2, 3)$. We are interested in how the release dates influence the order of the jobs in an optimal schedule. We set $r_1 = 0$, without loss of generality, and assume that $r_3 = r_2 + p$. Suppose $r_1 = r_2$, then we see that $2, 3, 1$ is optimal, which resembles some sort of "longest chain first" rule. Now suppose that $r_1 < r_2$. For $r_2 \geq r_1 + p + s$, the sequence $1, 2, 3$ is simply optimal. For $r_2 < r_1 + p + s$, we compare the sequence $1, 2, 3$ with $2, 3, 1$, which are illustrated in Figure 3. The first has optimal $\sum C_j = p+(p+s+p)+(p+s+p+p) = 6p+2s$, while the second sequence has optimal $\sum C_j = (r_2 + p) + (r_2 + p + p) + (r_2 + p + p + s + p) = 3r_2 + 6p + s$. Therefore, we conclude that the second sequence is optimal if and only if

$$r_2 \leq s/3, \tag{8}$$

which roughly means that the "longest chain first" rule becomes optimal whenever the release dates are "close enough". □

In the previous example, we see that it does not make sense to schedule vehicle 1 between vehicles 2 and 3, because that would add unnecessary switch-over time. This raises the question whether splitting such "platoons" of vehicles is ever necessary to achieve an optimal schedule. Let us first give a precise definition of this important notion, before slightly generalizing the example.

Definition 1.2. *A sequence of consecutive vehicles $J_k(m), J_k(m+1), \ldots, J_k(m+l)$ from some lane k is called a* platoon *if and only if*

$$r(J_k(j)) + p = r(J_k(j+1)) \tag{9}$$

for $n \leq j < n+m$ (under Assumption 1.2). We say that the platoon is split *in a given schedule y, if*

$$y(J_k(j)) + p < y(J_k(j+1)) \tag{10}$$

for some $m \leq j < m+l$.

Example 1.4. Suppose we have exactly two platoons of vehicles $F_A = (1, \ldots, n_A)$, $F_B = (n_A + 1, \ldots, n_A + n_B)$. To clarify notation, we write $r_A = r_1$ and $r_B = r_{n_A+1}$. We assume $r_A = 0$, without loss of generality, and suppose that $n_A < n_B$ and $r_A \leq r_B < n_A p + s$. Consider the ways the two platoons can merge by splitting A. Let k denote the number of vehicles of platoon A that go before platoon B and let $\sum C_j(k)$ denote the resulting total completion time. See Figure 1.4 for an illustration of the situation in case of $r_A = r_B$. For $0 < k \leq n_A$, we have

$$\sum C_j(k) = \max\{s, r_B - kp\}(n_B + n_A - k) + s(n_A - k) + \sum_{j=1}^{n_A+n_B} jp,$$

so when platoon A goes completely before platoon B, we get

$$\sum C_j(n_A) = sn_B + \sum_{j=1}^{n_A+n_B} jp, \tag{11}$$

since $\max\{s, r_B - kp\} = s$ by the assumption on r_B. It is easily seen that we have $\sum C_j(k) > \sum C_j(n_A)$ for $0 < k < n_A$. In words, if we decide to put one of platoon A's vehicles before platoon B, it is always even better to put all of them in front. After this example, we will show that this principle holds in general.

For $k = 0$, so when we schedule platoon A completly after platoon B, the total completion time becomes

$$\sum C_j(0) = r_B(n_A + n_B) + sn_A + \sum_{j=1}^{n_A+n_B} jp.$$

Comparing this to (11), we conclude that placing B in front is optimal whenever

$$r_B \leq (n_B - n_A)s/(n_A + n_B),$$

which directly generalizes the condition (8) that we derived for the case with $n_A = 1$ and $n_B = 2$. □

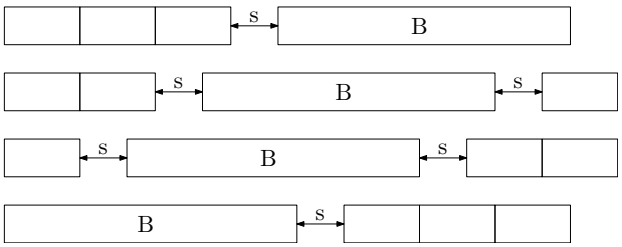

Figure 4: Illustration of splitting platoon A (with $n_A = 3$ and some arbitrary, not shown, n_B) in Example 1.4.

As the examples above show, the computation of the total completion time always includes a common term

$$\sum C_j = \sum_{j=1}^{n} jp + \dots,$$

which happens because of identical processing times (Assumption 1.1). The other contribution to the total completion time is due to the idle time between jobs.

Definition 1.3. *Any feasible schedule y induces an ordering of the jobs, so let $\pi_y(j)$ denote the jth job in this order. We define the* gap after the jth vehicle *as*

$$g_y(j) = y(\pi_y(j+1)) - y(\pi_y(j)), \tag{12}$$

for $0 \leq j < n$ and $g_y(n) = 0$.

Using this definition, we see that the total completion time for a schedule y can be expressed (under Assumption 1.1) as

$$\sum_{j=1}^{n} C_j(y) = \sum_{j=1}^{n} jp + (n-j)g(j). \tag{13}$$

This observation is the basis of the following theorem.

Theorem 1.1. *Under Assumption 1.1, every Single Intersection Scheduling Problem has an optimal schedule y in which no platoon is split.*

Proof. Consider an optimal schedule y in which some platoon is *split*, so assume there exists some smallest $j \geq 1$ such that $y(J_k(j)) + p < y(J_k(j+1))$. To simplify notation, let $i^* = J_k(j)$ and $j^* = J_k(j+1)$. We show how to derive a schedule y' that satisfies $y'(i^*) + p = y'(j^*)$ but has no larger total completion time. Exhaustively applying this argument proves the theorem.

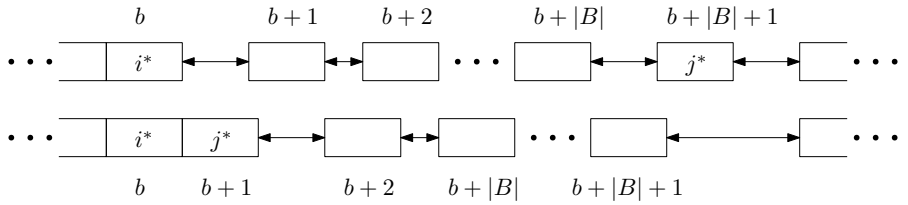

Figure 5: Sketch of schedules y (top) and y' (bottom) used in the proof of Theorem 1.1. The position numbers of the jobs are written above and below.

We let the selected vehicle *join the platoon* by setting $y'(j^*) = y(i^*) + p$. Consider the set of vehicles B that were scheduled between i^* and j^* in the original schedule. For $|B| = 0$, it is clear that y' is feasible and has strictly lower total completion time. When $|B| > 0$, we additionally move these jobs forward by setting $y'(j) = y(j) + p$ for $j \in B$. Consider the gaps in this new schedule, please see Figure 5 for a sketch of the situation. Let b denote the position of i^*, so formally $b = \pi_y^{-1}(i^*)$. The jobs at positions

$$l \in \{1, \dots, b-1\} \cup \{b + |B| + 2, \dots, n\},$$

did not change, so we have $g_{y'}(l) = g_y(l)$. Furthermore, we have $g_{y'}(b) = 0$ (because j^* follows i^* immediately) and for

$$l \in \{b+1, \dots, b+|B|\},$$

we have $g_{y'}(l) = g_y(l-1)$. Finally, we have $g_{y'}(b + |B| + 1) = g_y(b + |B|) + g_y(b + |B| + 1)$. We can now simply determine the change in total completion time, by calculating (almost telescoping sum)

$$\begin{aligned}
\sum_{j=1}^{n} C_j(y) - \sum_{j=1}^{n} C_j(y') &= \sum_{j=b+1}^{b+|B|} (n-j)(g_y(j) - g_y(j-1)) \\
&\quad + (n-b)g_y(b) - (n-b-|B|-1)g_y(b+|B|) \\
&= \sum_{j=b+1}^{b+|B|-1} g_y(j) \\
&\quad - (n-b-1)g_y(b) + (n-b-|B|)g_y(b+|B|) \\
&\quad + (n-b)g_y(b) - (n-b-|B|-1)g_y(b+|B|) \\
&= \sum_{j=b}^{b+|B|} g_y(j) \geq 0,
\end{aligned}$$

showing that it did indeed not increase. □

A more general version of this theorem, without Assumption 1.1, was proven by Limpens [4].

1.4 Single Platoon Positioning

Focus on the problem with two lanes F_A and F_B where we assume that there is only a single platoon arriving on lane B, i.e., the *single platoon positioning problem*. In this case, the optimal schedule can be easily computed by considering all positions. However, we use this simple case to obtain some insight into when vehicles that are close act as if they were platoons.

Assume that the first vehicle on the intersection arrives from lane A. We provide a criterium for vehicles from A to be candidates for going before the B-platoon, with length (number of vehicles) n_B.

Proposition 1.1. *Assume there is a vehicle $0 \in F_A$ starting at $y_0 = 0$. Let x_i denote the finish-start delay between vehicle $i-1$ and i when scheduled as early as possible, so let*

$$x_i = r_i - (r_{i-1} + p).$$

If there exists a smallest integer i' such that $x_{i'} \geq 2s + n_B p$, then any vehicle $i \geq i'$ needs to go after B in any optimal schedule.

illustrate the two job case with the "decision" diagram

1.4.1 Insertion Heuristic

We propose a heuristic that constructs a schedule for two lanes by considering the vehicles in order of arrival and iteratively inserting each next vehicle in the current partial schedule.

As in the single platoon scheduling problem, let F_A and F_B denote two lanes, but now we allow lane B to have arbitrary ariving vehicles. For the first platoon of B, find the single platoon position among vehicles of A. Then for every next platoon of B, solve the single platoon positioning problem, while treating the position of the previously scheduled platoons fixed.

The following example shows that the resulting schedule may not be optimal, because the position of earlier scheduled platoons restricts the possible positions of later platoons.

Example 1.5. come up with example □

2 Online Single Intersection Scheduling

We need to reconsider assumption 1.2 because it becomes relevant in the online setting. Smaller release dates r_j provides information to the controller earlier on.

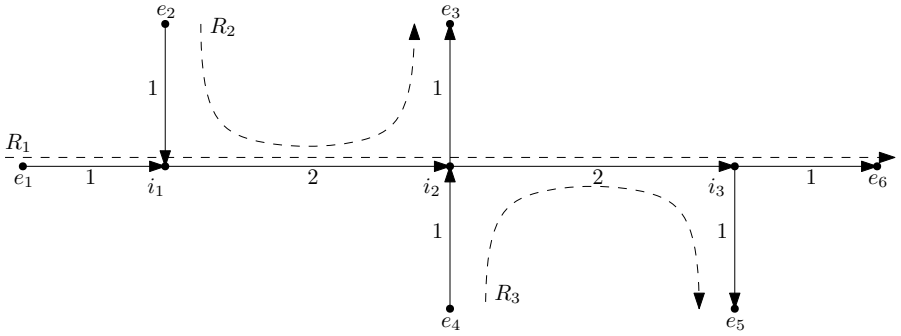

Figure 6: Example graph with three intersection i_1, i_2, i_3 and three vehicles, for which the routes are indicated by R_1, R_2 and R_3. The external nodes are labeled as $e_1, \ldots, e_6$. The weight (distance) of each arc is indicated next to it. The common paths are $P_{12} = \{(i_1, i_2)\}$, $P_{13} = \{(i_2, i_3)\}$ $P_{23} = \{(i_2)\}$ and the corresponding merge points are $M_{12} = \{i_1\}, M_{13} = \{i_2\}, M_{23} = \{i_2\}$.

3 Network of Intersections

We now leave the single intersection case and turn to vehicles traveling through a network of intersections. The network is modeled as a weighted directed graph $G = (V, E)$, with nodes and arcs representing intersections and roads, respectively. Vehicles can travel along a series of arcs that are connected. Let $d(x, y)$ be defined as the *distance* between nodes x and y. We assume there are no nodes of degree two, since their two incident arcs (x, y) and (y, z) could be merged into one arc (x, z) with $d(x, z) = d(x, y) + d(y, z)$, without loss of expressiveness. Each node of degree one is called an *external node* and models the location where vehicles enter (*entrypoint*) or exit (*exitpoint*) the network. A node of degree at least three is called an *internal node* and models an intersection.

Let the route of vehicle j be denoted by R_j, which is encoded as a sequence of nodes. Vehicle j enters the network at some external node $R_j(0)$ at time r_j and then visits the n_j nodes $R_j(1), \ldots, R_j(n_j)$ until it reaches the exitpoint $R_j(n_j + 1)$, where it leaves the network. Given two routes R_j and R_l, we define a *common path* $P = (i_1, \ldots, i_L)$ as a substring of some length L of both routes (including external nodes). We refer to the first node i_1 of a common path as a *merge point*. The set of all common paths for vehicles j and l is denoted by P_{jl}. The set of all merge points for vehicles j and l is denoted by M_{jl}. See Figure 3 for an illustration of the above concepts.

Vehicles are not able to overtake each other when traveling on the same arc. We assume that arcs provide infinite *buffers* for vehicles, meaning that there is no limit on the number of vehicles that are traveling on the same arc at the same time. However, we impose a minimum time required to travel along an arc (x, y). By assuming identical maximum speeds among vehicles, $d(x, y)$ can

be directly interpreted as this minimum travel time.

Let y_{ij} denote the time vehicle j enters intersection i. Crossing the intersection takes p_{ij} time. Like in the single intersection case, we have again two types of constraints. At most one vehicle can cross an intersection at the same time (i), so when two consecutive vehicles crossing an intersection originate from the same arc, they may pass immediately after each other. When a vehicle l that is about to cross next comes from a different arc than the vehicle j that last crossed the intersection, we require that there is at least a *switch-over time* s_{jl} between the moment j leaves and the moment l enters the intersection (ii).

Assuming that arrival times and routes of all vehicles are fixed and given, the task of the traffic controller is to decide when individual vehicles should cross intersections by determining y, while minimizing some measure of optimality. This problem is similar to the classical job shop scheduling setting, with intersections and vehicles now taking the roles of machines and jobs, respectively. Therefore, we will first discuss this classical problem before discussing how it should be extended to fit the above setting.

3.1 Job Shop Scheduling

Assume there are m machines and n jobs that need to be processed on these machines. A job j consists of exactly m operations, one for each of the machines. We use the notation (i, j) to denote the operation of job j that needs to be processed on machine i. The time required for processing operation (i, j) is denoted by p_{ij}. When a job only needs processing on a subset of the machines, we can simply assume that $p_{ij} = 0$ for the machines that are not involved. Like in the single machine case, we may define a release date r_{ij} for each operation, specifying the earliest time processing may start. Let the set of all operations be denoted by N.

The order in which a job visits the machines is called the *machine order* of this job. The machine order is given as part of the problem specification and may be different for each job. Instead of directly writing (i, j), we sometimes use the alternative notation O_{jk} to denote the k'th operation of job j. An operation $O_{j,k+1}$ may only start once its predecessor O_{jk} has completed processing, which means that the precedence constraint $O_{jk} \rightarrow O_{j,k+1}$ must be satisfied. In general, machine order of a job is encoded as a chain of precedence constraints. The set of all the pairs of operations to which such constraint applies is denoted by

$$\mathcal{C} = \{(O_{jk}, O_{j,k+1}) : 1 \leq j \leq n, 1 \leq k < m\}. \tag{14}$$

Each machine i can process at most one operation at the same time and, once started, operations cannot be preempted. Therefore, the scheduler needs to order the operations that need processing on i. We define the set of *conflict pairs*

$$\mathcal{D} = \{\{(i, j), (i, l)\} : (i, j), (i, l) \in N\}, \tag{15}$$

which generalizes the conflict pairs that we defined for the single intersection in Section 1.2.1.

Let y_{ij} denote the time when operation (i, j) starts processing, which needs to be determined by the scheduler. In the alternative notation of operation O_{jk}, we denote the starting time and processing time as $y(O_{jk})$ and $p(O_{jk})$, respectively. The completion time of operation (i, j) is defined as

$$C_{ij} = y_{ij} + p_{ij}, \tag{16}$$

We define the completion time of job j to be

$$C_j = y_{i^*j} + p_{i^*j} \tag{17}$$

where i^* denotes the machine on which the last operation of j is performed.

A valid schedule is given by setting values for y_{ij} such that the above requirements are met. There are various measures for how *good* a given schedule is. For the purpose of the discussion that follows, let us consider the well-known makespan objective

$$\max_{1 \le j \le n} C_j, \tag{18}$$

Minimizing the makespan often results in schedules that make efficient use of the available machines and can now be formulated as

$$\begin{aligned}
&\text{minimize } C_{\max} && && \text{(19a)}\\
&\quad \text{s.t. } C_{ij} \le y_{kj} && \text{for all } ((i,j),(k,j)) \in \mathcal{C}, && \text{(19b)}\\
&\qquad C_{il} \le y_{ij} \text{ or } C_{ij} \le y_{il} && \text{for all } \{(i,l),(i,j)\} \in \mathcal{D}, && \text{(19c)}\\
&\qquad C_{ij} \le C_{\max} && \text{for all } (i,j) \in N, && \text{(19d)}\\
&\qquad y_{ij} \ge r_{ij} && \text{for all } (i,j) \in N. && \text{(19e)}
\end{aligned}$$

The first set of constraints enforces the machine order for each job. The second set of constraints are called *disjunctive*, because they model the choice between jobs j and l to be scheduled first on machine i. The third set of constraints are used to define the makespan and the last line enforces the release dates.

Example 3.1. Consider two jobs on three machines with processing times $p_{\cdot 1} = \begin{pmatrix} 2 & 2 & 4 \end{pmatrix}^T$, $p_{\cdot 2} = \begin{pmatrix} 1 & 2 & 1 \end{pmatrix}^T$ and machine orders $(1,1) \to (2,1) \to (3,1)$ and $(3,2) \to (2,2) \to (1,2)$. It is easily seen that the optimal makespan is given by $C_{\max} = p_{11} + p_{21} + p_{31} = 8$ and an optimal schedule is shown in Figure 7. □

Let us discuss a simple extension to the job shop setting introduced above. Jobs may represent physical objects that need to be processed on machines. Suppose we want to model the *travel time* that is necessary to move the job physically to the next machine. We can use so-called *generalized precedence constraints* for this. We use the notation

$$(i, j) \xrightarrow{d} (k, j)$$

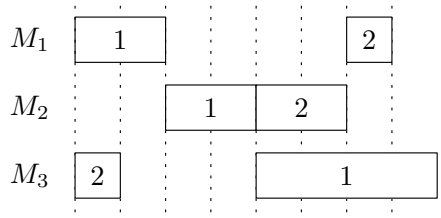

Figure 7: Optimal schedule for Example 3.1. The dashed lines indicate unit time steps. Note that machine 2 is kept idle, while operation $(2, 2)$ could have already been scheduled at time 1.

to indicate that operation (i, j) is followed by operation (k, j) and it takes d time to move the job from machine i to k. This constraint can also be formulated as

$$C_{ij} + d \leq y_{kj}. \tag{20}$$

Clearly, this generalizes the type of precedence constraints that we have been using up till now (the original type is obtained by setting $d = 0$).

To support the following discussion, let us introduce some additional notation. Consider again the set of conflict pairs $\mathcal{D}$. For each conflict pair $\{(i, j), (i, l)\} \in \mathcal{D}$, the scheduler needs to make a binary decision, i.e., it needs to decide whether jobs j goes before job l or vice versa. Similarly to what we did in Section 1.2.1, we define the auxilliary set

$$\bar{\mathcal{D}} = \{(i, j, l) : \{(i, j), (i, l)\} \in \mathcal{D}, j < l\}. \tag{21}$$

For each $(i, j, l) \in \bar{\mathcal{D}}$, we define the binary variable $o(i, j, l)$ to encode the decision in the disjunctive constraints (19c). A value of zero corresponds to j crossing first and a value of one indicates that l crosses first.

3.2 Problem Formulation

The traffic control problem on a network of intersections can be modeled as a job shop problem with generalized precedence constraints modeling the travel times of vehicles and additional restrictions on the ordering of operations. As in the single intersection case, intersections and vehicles are represented by machines and jobs, respectively. Note that external nodes in G are not considered to be intersections.

For each job j, the machine order is given by the route R_j of the corresponding vehicle. The release date r_j models the arrivals of vehicle j at its entrypoint $R_j(0)$. At internal nodes $i \in \{R_j(1), \ldots, R_j(n_j)\}$, the starting time y_{ij} models the planned crossing time of vehicle j at this intersection. The processing time p_{ij} models the amount of time necessary for safe crossing. Consider a pair of consecutive nodes (i, k) on the route R_j. The minimum time required to travel

is modeled by the generalized precedence constraint

$$(i,j) \xrightarrow{d(i,k)} (k,j). \tag{22}$$

Again, we denote the set of all pairs of operations $((R_j(n),j),(R_j(n+1),j))$ on consecutive pairs of nodes by $\mathcal{C}$.

Now consider the decisions that need to be made regarding the ordering of operations on the same machine, encoded by $o \in \{0,1\}^{\bar{\mathcal{D}}}$. First of all, note that some binary vectors do not correspond to a feasible schedule, because they encode a cycle of precedence constraints. By assuming that vehicles cannot overtake each other while driving on the same lane, we are further restricting the possible values of o. Consider a pair of vehicles j,l. On each common path $(i_1,i_2,\ldots,i_L) \in P_{jl}$, the order cannot change, so we must have

$$o(i_1,j,l) = o(i_2,j,l) = \cdots = o(i_L,j,l). \tag{23}$$

A special case is when the merge point i_1 is an external node. In that case, i_1 must be the entrypoint of both vehicles. That means that the order on p is determined by the order of arrival, which is determined by the release dates. Therefore, we require

$$o(i_1,j,l) = 1 \text{ if } r_j > r_l, \tag{24}$$

and $o(i_1,j,l) = 0$ otherwise. Finally, let $\mathcal{O} \subset \{0,1\}^{\bar{\mathcal{D}}}$ denote the set of all valid assignments to o.

We will now explain how the ordering decisions are related to the starting times y. Note that y_{ij} is only defined for internal nodes i. Consider an arbitrary triple $(i,j,l) \in \bar{\mathcal{D}}$. Suppose $i \in M_{jl}$. In that case, the vehicles j and l approach i from different arcs, which means that switch-over time must be enforced, so we have

$$\begin{cases} (i,j) \xrightarrow{s} (i,l) & \text{if } o(i,j,l) = 0, \\ (i,l) \xrightarrow{s} (i,j) & \text{if } o(i,j,l) = 1. \end{cases} \tag{25}$$

When i is not a merge point, but it lies on some common path of j and l, then both vehicles are driving on the same lane, so we simply have

$$\begin{cases} (i,j) \to (i,l) & \text{if } o(i,j,l) = 0, \\ (i,l) \to (i,j) & \text{if } o(i,j,l) = 1. \end{cases} \tag{26}$$

We merge these two cases by defining the switch-over time as

$$s(i,j,l) = \begin{cases} s & \text{if } i \in M_{jl}, \\ 0 & \text{otherwise }, \end{cases} \tag{27}$$

which allows us to now compactly formulate the problem of minimizing total delay in the network as

$$\text{minimize} \quad \sum_{1 \le j \le n} C_j \tag{28a}$$

$$\text{s.t.} \quad y_{ij} \ge r_{ij} \qquad \text{for all } (i,j) \in N, \tag{28b}$$

$$(i,j) \xrightarrow{d(i,k)} (k,j) \qquad \text{for all } ((i,j),(k,j)) \in \mathcal{C}, \tag{28c}$$

$$(i,j) \xrightarrow{s(i,j,l)} (i,l) \qquad \text{if } o(i,j,l) = 0, \text{ for } (i,j,l) \in \bar{\mathcal{D}}, \tag{28d}$$

$$(i,l) \xrightarrow{s(i,j,l)} (i,j) \qquad \text{if } o(i,j,l) = 1, \text{ for } (i,j,l) \in \bar{\mathcal{D}}, \tag{28e}$$

$$o \in \mathcal{O}. \tag{28f}$$

Definition 3.1 (Multi-Intersection Scheduling Problem)**.** *Given an intersection graph G, the job shop problem with release dates, graph travel time constraints* (28c)*, machine-dependent setup times and no-overtaking constraint* (28d)*,* (28e) *and* (28f) *with total completion time objective is called the* multi-intersection scheduling problem*. In the three-field notation, it could be denoted as* $Jm \,|\, r_j, graph, s_{ijk}, no\text{-}overtaking \,|\, \sum C_j$*.*

In what follows, we will also make the following assumption.

Assumption 3.1. *The graph G on which vehicles are travelling is acyclic and connected.*

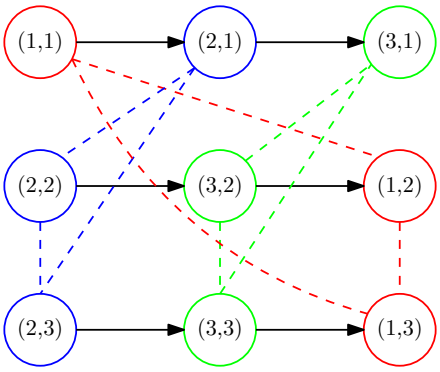

Figure 8: Disjunctive graph corresponding to a job shop instance with three jobs and three machines. Operations that need to be scheduled on the same machine have been given the same color, emphasizing the cliques formed by their disjunctive arcs (drawn as dashed lines).

3.3 Disjunctive graph

As an alternative to the natural formulation (19), job shop instances can also be represented very clearly by their corresponding *disjunctive graph* (see Section 7.1 of [2] for a textbook introduction). Let $(N, \mathcal{C}, \mathcal{D})$ be a directed graph with nodes N corresponding to operations and two types of arcs. The *conjunctive* arcs $\mathcal{C}$ encode the given machine order for each job and this is parallel to (14). If $(i,j) \rightarrow (k,j) \in \mathcal{C}$ then job j needs to be processed on machine i before it may be processed on machine k. The *disjunctive* arcs $\mathcal{D}$ are used to encode the decisions regarding the ordering of jobs on a particular machine, which is parallel to (15). For every conflict pair $\{(i,j),(i,l)\} \in \mathcal{D}$ of distinct jobs j and l that need to be processed on the same machine i, there is a pair of arcs in opposite directions between (i,j) and (i,l). Therefore, each machine corresponds to a clique of double arcs between all the operations that need to be performed on that machine.

For each machine i, the scheduler needs to order the operations that need processing on i. This corresponds to choosing exactly one arc from every pair of opposite disjunctive arcs. Let $\mathcal{D}' \subset \mathcal{D}$ be such a selection of disjunctive arcs and let $G(\mathcal{D}')$ denote the resulting graph, to which we will refer as a *complete* disjunctive graph. It it not hard to see that any feasible schedule corresponds to an acyclic complete disjunctive graph.

Suppose we have $m = 3$ machines and $n = 3$ jobs. The first job has a machine order of 1,2,3 and the other two jobs have a machine order of 2,3,1. Figure 8 shows the structure of the corresponding disjunctive graph for this instance. Note that this graph does not yet fully characterize a problem instance, because the information on processing times is not encoded, The next section shows how to resolve this issue.

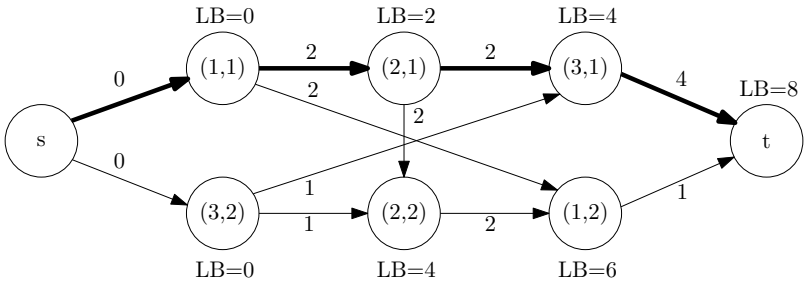

Figure 9: Disjunctive graph corresponding to Example 3.1 augmented with source, sink and weights. The longest path determining the makespan $C_{\max} = LB(t)$ is highlighted.

In addition to encoding operation ordering decisions, the disjunctive graph contains information about the processing times. We add a *source* S and a *sink* T, which are dummy nodes. The source has outgoing arcs with weight zero to every first operation of each job. Every last operation of each job is connected to the sink. Each outgoing arc of a node (i, j) is set to have weight equal to p_{ij}.

Given a complete disjunctive graph, we can recursively calculate a lower-bound $LB(O_{jk})$ on the starting time of operation O_{jk} from the following relationship

$$LB(O_{jk}) = \max_{n \in O_{jk}^-} LB(n) + p_n, \tag{29}$$

using v^- to denote the set of in-neighbors of node v. Put in other words, $LB(O_{jk})$ is the length of the longest path from S to O_{jk}. By similarly computing

$$LB(T) = \max_{n \in T^-} LB(n) + p_n, \tag{30}$$

we obtain the makespan objective for the current schedule, see Figure 9 for an example.

Similarly, the job completion time can be calculated as follows. Instead of t, we add n dummy nodes $t_1, \ldots, t_n$ and connect each t_j to the last operation (i^*, j) of job j by an arc with weight p_{i^*j}. Defining

$$LB(t_j) = LB((i^*, j)) + p_{i^*j}, \tag{31}$$

we obtain the completion time of each job j, which also allows us to express the total completion time.

In case of generalized precedence constraints, the above method should be adapted slightly. Every arc $(i, j) \xrightarrow{d} (k, l)$ (either conjunctive or disjunctive) gets weight $p_{ij} + d$ instead. The arcs to the sink(s) keep their weight.

3.4 Schedule generation

There are some classes of schedules that are relevant for job shop problems. The following definitions are taken from the textbook by Pinedo [2].

Definition 3.2 (Non-Delay Schedule)**.** *A feasible schedule is called non-delay if no machine is kept idle while an operation is waiting for processing.*

Definition 3.3 (Active Schedule)**.** *A feasible non-preemptive schedule is called active if it is not possible to construct another schedule, through changes in the order of processing on the machines, with at least one operation finishing earlier and no operation finishing later.*

Definition 3.4 (Semi-Active Schedule)**.** *A feasible non-preemptive schedule is called semi-active if no operation can be completed earlier without changing the order of processing on any one of the machines.*

It is clear that active schedules are necessarily semi-active, but the converse does not always hold. Furthermore, when preemption is not allowed, non-delay schedules are necessarily active. Consider again Example 3.1. The optimal schedule shown in Figure 7 is not non-delay, because machine 2 is kept idle from time 2 to 3, but operation $(2, 2)$ could be scheduled here.

For some scheduling problems, it can be shown that an optimal schedule exists that belongs to such classes. For job shop problems, the optimal schedule is not necessarily non-delay, as the example illustrates for the makespan objective. However, it can be shown that $Jm||\gamma$ has an active optimal schedule, when γ is a regular objective, which means that it is a non-decreasing function of the completion times $C_1, C_2, \ldots, C_n$. The makespan and total completion time are examples of regular objectives. Because of this result, branch-and-bound methods have been proposed based on enumeration of all active schedules.

also active optimal schedule for $Jm|r_j|\gamma$?

Proposition 3.1. *Every complete disjunctive graph corresponds to a unique semi-active schedule.*

Proof. Given some semi-active schedule y, it is obvious that the disjunctive graph can be constructed by simply encoding the corresponding ordering o of the operations by the disjunctive arcs.

Suppose we are given a complete disjunctive graph encoding some ordering o. Let O_{jk} and $O_{j,k+1}$ be a pair of consecutive operations. Note that $LB(O_{jk})$ is a lower bound on the starting time of operation $O_{j,k+1}$, i.e., every feasible schedule y must satisfy $y(O_{j,k+1}) \geq LB(O_{jk})$. Now observe that by setting $y(O_{j,k+1}) = LB(O_{jk})$ for any such consecutive pairs, we obtain a valid semi-active schedule with ordering o. Therefore, any other schedule that satisfies o must have some $y_{kj} > LB((i, j))$, hence is not semi-active. □

Include here our solution of Exercise 7.11 from Pinedo, i.e., a branching scheme for generating all schedules based on inserting disjunctive arcs. From the resulting disjunctive graph, all semi-active schedules follow by applying the last

position rule, introduced in the next subsection.

3.4.1 Priority dispatching

A *priority dispatching rule* selects a job from a list of currently available (unscheduled) jobs to be scheduled next. This produces a sequence χ of jobs that are placed in a *partial schedule*. Manually crafted priority dispatching rules are mostly based on elementary features of the current partial schedule. Well-known examples include the job with the most/least work remaining (MWR/LWR) or the job with the shortest/longest processing time (SPT/LPT).

Determining the dispatching sequence alone is not enough to obtain a schedule y. The exact starting times are determined by a *placement rule*. A sensible placement rule would be to put the job in the earliest gap of idle time where this job fits, i.e., where it can be placed without violating any constraints. We call this the *earliest gap* rule. Alternatively, consider the straightforward placement rule that chooses the smallest starting time y_{ij} that satisfies $y_{ij} \geq y_{il} + p_{il}$ for all jobs l that have already been placed on machine i in the current partial schedule and $y_{ij} \geq y_{kj} + p_{kj}$, when (k, j) is the preceding operation. We call this the *last position* rule, because the job is placed as the last job on the machine.

We are now interested in whether scheduling with dispatching rules can generate all necessary schedules. More precisely, we wonder whether all active schedules can be generated from dispatching.

Proposition 3.2. *For every feasible semi-active schedule, there exists a sequence χ that generates it using the last position rule.*

Proof. Consider an arbitrary semi-active schedule y and let G be the corresponding complete disjunctive graph, which is unique by Proposition 3.1 and has $y_{jk} = LB(O_{jk})$ for each node $O_{jk} \in N$. Because G must be acyclic, there exists some topological ordering χ of the nodes. Therefore, dispatching in order of χ with the machine makespan rule will produce exactly the schedule with $y_{jk} = LB(O_{jk})$. □

4 Job Shop with Reinforcement Learning

4.1 Zhang et al.

The authors of [7] use reinforcement learning to solve the job shop problem with makespan objective $Jm||C_{\max}$. Specificaly, the method learns dispatching rules, so actions correspond to selecting which (unfinished) job to schedule next. States are represented by a disjunctive graph corresponding to the partial schedule. The policy is parameterized by a graph neural network that reads the current disjunctive graph. They use a dense reward based on the current lower bound of the makespan.

The state transition example shown in Figure 2 of [7] suggests yet another placement rule, because operation O_{32} is placed before O_{22}, but it does not fit the gap. Consequently, operation O_{22} has to be delayed by three time steps. Formalize the placement rule that Zhang et al. use and argue that this does not guarantee that all active schedules can be generated in this way.

4.2 Structure in Network Schedules

Before we start experimenting with the above problem (see if the MIP method scales and later trying to learn policies with (MA)RL), we first should try to find some common structural patterns in solutions or even some simple general scheduling rules.

- Study simple situations with the total completion time objective ($\sum C_j$), in which some kind of LPT rule holds. (*explain why LPT-like schedules arise*)
- Study tandem networks where the $\sum C_j$ objective causes platoon splitting due to some kind of "propagation of delay costs". (*insert the example that I have*)
- Can we study this "delay cost propagation" using some kind of dependency graph, e.g., by recording that delaying job A would require us to also delay job B, which would also require to delay C, etc.?

4.3 Platoon Splitting

In the single intersection case, platoon splitting is never necessary. So in this context, platoon splitting can only become necessary from using a different performance metric, e.g., one that takes into account fairness. Once we start looking at the network-level interactions, however, we can show that platoon splitting is sometimes necessary to obtain an optimal schedule. The main idea is that delaying a job j might require to also delay further downstream jobs. Therefore, it might be cheaper overall to not delay j, but interrupt some already running job l instead.

5 Discussion

We assumed that it is always possible to compute a speed profile, given a feasible schedule y for the scheduling problem. However, I think that certain parameters like maximum acceleration and breaking influence the processing time p and the switch-over time s that are necessary for this assumption to hold.

discuss multiple lanes, opposing lanes on the same arc and consider multiple phases at intersections

6 Overview of Notation

symbol	description
n	number of jobs
m	number of machines
(i, j)	operation of job j on machine i
O_{jk}	k'th operation of job j
p_{ij} or $p(O_{jk})$	processing time of operation (i, j) or O_{jk}, resp.
y_{ij} or $y(O_{jk})$	starting time of operation (i, j) or O_{jk}, resp.
C_{ij}	completion time of operation (i, j)
C_j	completion time of job j
$F_1, \ldots, F_K$	partition of jobs into K families
$n_k = \lvert F_k \rvert$	number of jobs in family k
$J_k(1) \to \cdots \to J_k(n_k)$	job order of family F_k
$G = (V, E)$	intersection graph
$R_j(1), \ldots, R_j(n_j)$	route of vehicle j
M_{jl}	merge points for routes of vehicles j and l
$d(x, y)$	distance between nodes x and y

References

[1] R. W. Timmerman and M. A. A. Boon, "Platoon forming algorithms for intelligent street intersections," *Transportmetrica A: Transport Science*, vol. 17, pp. 278–307, Feb. 2021.

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

[3] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," in *Annals of Discrete Mathematics* (P. L. Hammer, E. L. Johnson, and B. H. Korte, eds.), vol. 5 of *Discrete Optimization II*, pp. 287–326, Elsevier, Jan. 1979.

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

[5] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of Machine Scheduling Problems," in *Annals of Discrete Mathematics* (P. L. Hammer, E. L. Johnson, B. H. Korte, and G. L. Nemhauser, eds.), vol. 1 of *Studies in Integer Programming*, pp. 343–362, Elsevier, Jan. 1977.

[6] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*. Cham: Springer International Publishing, 2015.

[7] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and C. Xu, "Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning," Oct. 2020.