Traffic Scheduling

Jeroen van Riel

Oktober 2023

1 Preliminaries

1.1 Single Machine Scheduling

Suppose we have n jobs that need processing on a single machine. The time required for each job $j \in \{1, \dots, n\}$ is called the *processing time* p_j. Once started, jobs may not be *preempted*. In order to obtain a valid *schedule*, we need to determine the start time y_j for each job j, such that at most one job is processing on the machine at all times. Let $C_j = y_j + p_j$ denote the *completion time* of job j. Our objective is to minimize the *total completion time* $C_1 + \cdots + C_n$. By means of an *interchange argument*, it can be easily shown that an optimal solution is given by sorting the jobs according to increasing processing times, which is known as the Shortest Processing Time first (SPT) rule.

Now suppose that job j becomes available at its *release date* r_j, then a valid schedule requires $y_j \geq r_j$. It is not difficult to see that non-uniform release dates may require us to introduce *idle time* to obtain an optimal schedule. For example, 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 has a better total completion time of $\sum C_j = (\epsilon + 1) + (\epsilon + 1 + 2) = 4 + 2\epsilon$. Schedules without idle time are called *non-delay*.

Next, we consider *precedence constraints* between 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 $J_1, \dots, J_k$ be a partition of jobs into k non-empty families. For each family, we require that their jobs $J_l = \{j_1, \dots, j_{n_l}\}$ are processed in the order $j_1 \to j_2 \to \cdots \to j_{n_l}$, without loss of generality. Note that the order between jobs from different families is unspecified, which means that chains may be *merged* arbitrarily.

When job j is directly followed by job l, we might want to introduce *sequence-dependent setup time* s_{jl}, 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.

Formally, for any pair of jobs $j_1 \in J_{l_1}$ and $j_2 \in J_{l_2}$ belonging to distinct families, we require that either

$$y_{j_2} \geq C_{j_1} + s_{l_1,l_2},$$

is satisfied or

$$y_{j_1} \geq C_{j_2} + s_{l_2,l_1}.$$

1.2 Single Intersection Scheduling

We are interested in the task of assigning arriving vehicles to a time slot on a single intersection. Using the modelling ingredients introduced above, we can now pose this task as a scheduling problem. Let vehicles be represented by jobs. We will refer to y_j as the *crossing time*, since it represents the time at which the vehicle starts crossing the intersection, which is modelled by the single machine. The time it takes before the next vehicle can enter the intersection is modelled by the processing time p_j. The earliest possible crossing time of a vehicle is modelled by its release date r_j. All vehicles that arrive to the intersection from the same lane belong to the same family. We assume that vehicles are driving on single-lane roads, which means that we do not allow *overtaking*. We model this by introducing chain precedence constraints based on the order of arrival (release dates). To guarantee safety, we require a setup time s_{l_1,l_2} between vehicles coming from different lanes. As the optimization objective, we consider the total completion time, since this is equivalent to minimizing the total delay experienced by all vehicles.

The problem of minimizing the total completion time with release dates (written as $1|r_j|\sum C_j$ in the three-field notation [1]) has been long known to be NP-hard [2], so there is no hope in finding a polynomial algorithm for our more general problem unless $\mathcal{P} = \mathcal{NP}$. Therefore, we must resort to exhaustive branch-and-bound methods or heuristics.

From here on, we will assume uniform processing time p and uniform setup time s, which is accurate enough to model vehicles that cross the intersection without turning. For each job family $J = \{1, \ldots, n\}$, observe that we may assume that $r_i = \max\{r_i, r_{i-1} + p\}$, without loss of generality, because of the chain precedence constraints. Before discussing solution methods, let us first consider some simple examples.

Example 1.1. Consider the situation with $J_1 = \{1\}, J_2 = \{2\}$. 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 earlier release date, then it is easily seen that an optimal schedule requires vehicle 1 to go first.

Example 1.2. Consider the situation with $J_1 = \{1\}, J_2 = \{2, 3\}$. We are interested in how the release dates influence the order of the jobs in an optimal

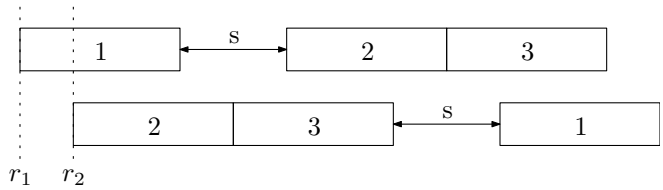

Figure 1: Illustration of the two possible sequences in Example 1.2.

schedule. We 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 (*relate this to Algorithm 3.1.4 of Pinedo*). Now assume that $r_2 < r_1 + p + s$, otherwise $1, 2, 3$ is simply optimal, because there is no *conflict* between the two lanes (*we will make this more precise later*). Furthermore, set $r_1 = 0$, without loss of generality. We compare the sequence $1, 2, 3$ with $2, 3, 1$. 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 first sequence is optimal if and only if

$$r_2 >= s/3,$$

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

1.3 Branch-and-Bound

Formulate the problem as a MIP and solve using branch-and-bound. Argue that imposing maximum vehicle delay allows us to limit the set of disjunctions that we need to consider.

lazy row generation
define conflicts

1.4 Insertion Heuristic

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

2 Traffic Scheduling in Networks

2.1 Job Shop Scheduling

This section shortly introduces the classical *job shop* scheduling problem. Assume there are m machines and n jobs. A job j consists of exactly m operations,

one for each of the machines. We let (i, j) 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}. The operations of a job need to be executed in a fixed given order, which may be different among jobs, and an operation may only start once its predecessor has completed processing. Each machine can process at most one operation at the same time and, once started, operations cannot be preempted.

Let the set of all operations be denoted by N. Furthermore, we encode the job routes by defining the set A of precedence constraints $(i, j) \to (k, j)$. Let y_{ij} denote the start of operation (i, j). The completion time of job j is defined as $C_j := y_{lj} + p_{lj}$, where l is the last machine on which j must be processed. 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 this example, let us consider the well-known makespan objective $C_{\max} := \max_j C_j$, which is often related to efficient use of the available machines. Minimizing the makespan can now be formulated as a Mixed-Integer Program (MIP) as follows:

$$
\begin{aligned}
&\text{minimize } C_{\max} \\
&\qquad y_{ij} + p_{ij} \leq y_{kj} && \text{for all } (i, j) \to (k, j) \in A \\
&\qquad y_{il} + p_{il} \leq y_{ij} \text{ or } y_{ij} + p_{ij} \leq y_{il} && \text{for all } (i, l) \text{ and } (i, j), i = 1, \ldots, m \\
&\qquad y_{ij} + p_{ij} \leq C_{\max} && \text{for all } (i, j) \in N \\
&\qquad y_{ij} \geq 0 && \text{for all } (i, j) \in N
\end{aligned}
$$

The first set of constraints enforce the order of operations belonging to the same job. The second set of constraints are called *disjunctive*, because they model that we need to choose between jobs j and l to be scheduled first on machine i. The next constraints are used to define the makespan and the last line enforces non-negative start times.

2.2 MIP Formulation

We now turn to vehicles traveling through a network of intersections. The network may be thought of as a weighted directed graph $G = (V, E)$, with nodes and arcs representing intersections and roads, respectively. **For now, we assume that the graph is acyclic**. 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. Furthermore, we assume the graph is connected. Each node of degree one is called an *external node* and models the location where vehicles enter (*entrypoint*) or exit (*exitpoint*) the network. A node of degree at least three is called an *internal node* and models an intersection.

Each vehicle j enters the network at some external node s and follows a predetermined sequence of arcs $R = ((s, i_1), (i_1, i_2), \ldots, (i_{n-1}, i_n), (i_n, d))$ towards an external node d where it leaves the network. 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 uniform maximum speed among vehicles, $d(x, y)$ can be directly interpreted as this minimum travel time.

Let $y(i, j)$ denote the time vehicle j enters intersection i. Crossing an intersection takes p time per vehicle and at most one vehicle can cross an intersection at the same time. When two consecutive vehicles crossing an intersection originate from the same arc, they may pass immediately after each other. However, when a vehicle j_1 that wants to cross comes from a different arc than the vehicle j_0 that last crossed the intersection, we require that there is at least a *switch-over time* S between the moment j_0 leaves and the moment j_1 enters the intersection.

Assuming that arrival times and routes of all vehicles are fixed and given, our task is to determine when individual vehicles should cross intersections by setting values for $y(i, j)$. This problem is similar to job shop scheduling, with intersections and vehicles now taking the roles of machines and jobs, and can also be formulated as a MIP, as we will show below.

The main difference with job shop scheduling is related to the ordering of vehicles. In the job shop model, every order $o \in \sigma_n$ of jobs on a machine i was allowed. By assuming that vehicles cannot overtake each other, we are limiting the valid orderings. Given two routes R_j and R_l, we define a common path $p = (i_1, \ldots, i_L)$ as a substring of both routes. We refer to the first node i_1 of a common path as a *merge point*. The set of all common paths is denoted by P_{jl}.

Consider a common path p whose merge point i_1 is an external node. In this case, i_1 must be the entrypoint of both vehicles. That means that the order on p is determined by the order or arrival. Let r_j denote the arrival time (called the *release date* in scheduling) of vehicle j. If $r_j < r_l$, then we require that j goes first, formally stated as

$$y_{ij} + p_{ij} \leq y_{il} \text{ for all } i \in p.$$

Now suppose that j and l have a mergepoint $i_1 \in p$ that is not an external node, then these vehicles approach i_1 from different arcs, which means that we have what we will call a *conflict* between j and l, because the scheduler must choose which vehicle crosses the intersection first. Furthermore, the switch-over time should be respected at i_0. Requiring that vehicle j must cross first can be formally stated as

$$y_{i_1 j} + p_{i_1 j} + S \leq y_{i_1 l} \text{ and } y_{ij} + p_{ij} \leq y_{il} \text{ for all } i_1 \neq i \in p.$$

In order to model the above decision making with a MIP, we introduce binary decision variables $s_{i,j,l}$ to encode the relative order of j and l at some common node $i \in p \in P_{jl}$. A value of zero corresponds to j crossing first and a value of one indicates that l crosses first. In the case of i_1 being an external node, we

treat $s_{i,j,l}$ as a fixed parameter of the MIP. For each common path $p \in P_{jl}$, we must have

$$s_{i_1,j,l} = s_{i_2,j,l} = \cdots = s_{i_L,j,l}.$$

Let N denote the set of operations like in the job shop example, but now pairs correspond to vehicles and the intersections they encounter along their route. Also similarly, let A encode the route constraints of each vehicle. We use p_1 to denote the merge point of path p. Let P^e_{jl} denote all the common paths that have an external merge point and let P_{jl} denote the other common paths. By implicitely letting the indices j and l run over all the *ordered pairs* of vehicles, we write

$$\begin{aligned}
&\text{maximize } P(y) && && \text{(1a)}\\
&y_{ij} + p_{ij} + t_{ik} \le y_{kj}, && \text{for all } (i,j) \to (k,j) \in A, && \text{(1b)}\\
&y_{ij} + p_{ij} \le y_{il}, \text{ for all } i \in p, && \text{for all } p \in P^e_{jl}, s_{p_1 jl} = 1, && \text{(1c)}\\
&y_{p_1 j} + p_{p_1 j} + S \le y_{p_1 l}, && \text{for all } p \in P_{jl}, s_{p_1 jl} = 1, && \text{(1d)}\\
&y_{ij} + p_{ij} \le y_{il}, \text{ for all } i \ne p_1, && \text{for all } p \in P_{jl}, s_{p_1 jl} = 1, && \text{(1e)}\\
&y_{ij} \ge 0, && \text{for all } (i,j) \in N, && \text{(1f)}
\end{aligned}$$

where $P(y)$ denotes some unspecified performance metric, leaving for now the question of what it means for a schedule to be *good*. In order to solve this program using existing solvers, we use the well-known *big-M* method to encode which constraints from sets (1c), (1d) and (1e) are active for specific values of the binary variables s.

2.3 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. (*TODO: 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". (*TODO: 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.?

2.4 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.

References

[1] 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.

[2] 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.