# Bounded lane capacity

Up to this point, we have not taken into account the fact that lanes between intersection have finite capacity. We need to incorporate this aspect in order to develop a model that could be used for practical applications. Under high traffic loads, lanes with finite buffer capacity can prevent vehicles from crossing upstream intersections. Therefore, the traffic controller needs to take into account these additional constraints.

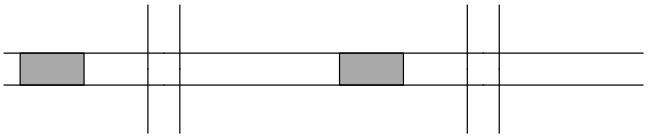

Figure 1: Illustration of two intersections in tandem.

We start with the simplest extension of the single intersection model by considering two intersections in tandem, as illustrated in Figure 1. Let the $v$ denote the left intersection and $w$ the right intersection and assume that traffic drives from left to right. Let the length and width of vehicle $i$ be denoted by $L_i$ and $W_i$, respectively. We measure the position of a vehicle at the front bumper and we let position $x = 0$ be at the stopline of intersection $v$. Let the position of intersection's $w$ stopline be denoted by $x = d(v, w)$. To simplify the following discussion, we make the following assumption on the vehicle sizes.

**Assumption 1.** *All vehicles have the same length $L_i = L$ and width $W_i = W$.*

First, we investigate the trajectories generated by the `MotionSynthesize` procedure and derive crossing time constraints that are sufficient to maintain safety. The minimum time required for a vehicle to come to a full stop or, equivalently, to accelerate to full speed from a stop, is given by

$$\hat{t} = \frac{v_{\max}}{a_{\max}}.$$

The trajectory of full stop is given by

$$x(t) = \frac{t^2 a_{\max}}{2},$$

so the minimum distance required for a vehicle to fully accelerate or decelerate is given by

$$x(\hat{t}) = \frac{v_{\max}^2}{2a_{\max}}.$$

We require that vehicles drive at full speed as long as they occupy an intersection. Therefore, a vehicle crossing $v$ can only start decelerating after $x(t) \geq L + W$. Suppose that we want to design the tandem network such that the lane segment $(v, w)$ has capacity for at least $c(v, w)$ stationary vehicles, then we must have

$$d(v, w) \geq L + W + 2x(\hat{t}) + (c(v, w) - 1)L.$$

Conversely, given lane length $d(v, w)$, this bound allows us to compute the maximum capacity as

$$c(v, w) = \texttt{floor}\left(\frac{d(v, w) - W - 2x(\hat{t})}{L}\right). \tag{1}$$

**Remark 1.** *Without Assumption 1, we cannot derive such a simple formula for the capacity, because it depends on the specific lengths of the vehicles that are arriving to the network.*

In order to guarantee that crossing time schedule $y$ allows feasible trajectories, we need to define additional constraints involving both the crossing time at $v$ and $w$ simultaneously. Like for the single intersection, let $\rho = L/v_{\max}$ denote the minimum time between two crossing times of vehicles of the same class. We show that the following type of constraints are sufficient for feasibility.

**Proposition 1.** *If we have*

$$y(i, w) - \frac{d(v, w)}{v_{\max}} + c(v, w)\rho \leq y(j, v), \tag{2}$$

*for every vehicle $i, j \in \mathcal{N}(l)$ with $k(i)+c(v, w) = k(j)$, then the MotionSynthesize procedure yields feasible trajectories.*

*Proof.* Any trajectory $x_i(t)$ of vehicle $i$ must satisfy

$$x_i(t) \geq \left(t - y(i, w) + \frac{d(v, w)}{v_{\max}}\right) v_{\max}.$$

For any vehicle $i_\tau = (l(i), k(i) + \tau)$, the corresponding trajectory $x_\tau(t)$ satisfies

$$x_{\tau+1}(t) \leq x_\tau(t) + \rho.$$

Now show that this means that at least the first "stopping position" is reachable for vehicle $j$. □

**Remark 2.** *The condition in Proposition 1 is not always necessary. There are situations in which vehicle $j$ is able to cross $v$ earlier.*

## Network scheduling

We will now show how the trajectory optimization problem can be transformed into a crossing time scheduling problem. We first introduce the notation that is used to capture how vehicles drive through a network. After we show how the MotionSynthesize procedure can be extended to networks, we present the general form of the crossing time scheduling problem.

For each vehicle $i = (l, k)$, we will refer to $l$ as the *vehicle class*, because vehicles are no longer bound to a unique lane. We define a graph $(V, E)$ with *labeled edges* as follows. Let $V$ denote the indices of the intersections. Let $E$ denote the set of ordered triples $(l, v, w)$ for each class $l$ whose vehicles travel from intersection $v$ to $w$. Let $d(v, w)$ denote the distance between intersections $v$ and $w$ and let $c(v, w)$ denote the resulting capacity as given by equation (1).

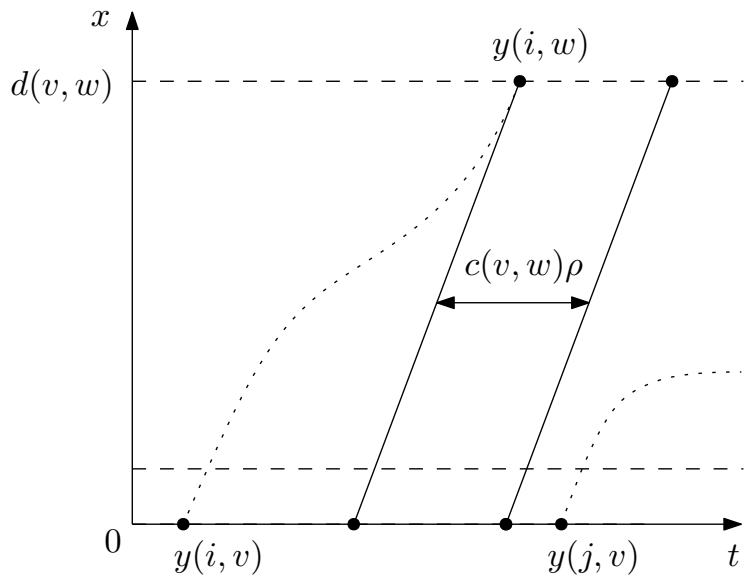


Figure 2: Illustration of the buffer constraint between $y(i, w)$ and $y(j, v)$. The slope of the two parallel lines is $v_{\max}$. The two dotted lines are examples of trajectories $x_i(t)$ and $x_j(t)$.

Let $\mathcal{N}(l)$ denote all the vehicles of class $l$. Let $\mathcal{R}(l)$ denote the sequence of intersections visited by vehicles from class $l$ and let $v_0(l)$ denote the first intersection on this route.

Recall that trajectories for the single intersection problem can be generated by the `MotionSynthesize` procedure, which is based on solving a linear program. We present a natural extension of this procedure such that it can be used when a vehicle encounters multiple intersections along its route. For vehicles from class $l$, consider the sequence of checkpoints

$$\tau = ((t_0, x_0), \ldots, (t_m, x_m)),$$

where $t_n$ is a crossing time and $x_n$ is the position of the $n$th intersection on $\mathcal{R}(l)$, relative to the first intersection. Given the trajectory $y$ of the vehicle ahead, the trajectory for a vehicle with checkpoint sequence $\tau$ is computed by the procedure

$$\begin{aligned}
&\texttt{CheckpointTrajectory}(\tau, y) := \\
&\qquad \arg\min_{x:[t_0,t_m]\to\mathbb{R}} \int_{t_0}^{t_m} |x(t)|dt \\
&\qquad\qquad \text{subject to } \ddot{x}(t) = u(t), && \text{for all } t \in [t_0, t_m], \\
&\qquad\qquad\qquad 0 \le \dot{x}(t) \le v_{\max}, && \text{for all } t \in [t_0, t_m], \\
&\qquad\qquad\qquad |u(t)| \le a_{\max}, && \text{for all } t \in [t_0, t_m], \\
&\qquad\qquad\qquad |x(t) - y(t)| \ge L, && \text{for all } t \in [t_0, t_m], \\
&\qquad\qquad\qquad x(t_n) = x_n, && \text{for all } n = 0, \ldots, m, \\
&\qquad\qquad\qquad \dot{x}(t_n) = v_{\max}, && \text{for all } n = 0, \ldots, m.
\end{aligned}$$

From now on, we will assume $v_{\max} = 1$, without loss of generality. We assume that vehicle arrive to the network at full speed and are *safe* by requiring that $r_i + L \leq r_j$ for every conjunctive pair $(i, j) \in \mathcal{C}$. Furthermore, in order to keep the presentation of the scheduling problem simple, we make the following assumption on vehicles routes.

**Assumption 2** (Edge-Disjoint Routes)**.** *Every lane $(v, w)$ is visited by at most one vehicle class. Formally, $(l_1, v, w) \in E$ and $(l_2, v, w) \in E$ implies $l_1 = l_2$.*

We now show how to obtain schedules in this model by formulating a MILP. Let $y(i, v)$ denote the crossing time of vehicle $i$ at intersection $v \in V$. Let $\mathcal{C}$ be the conjunctive pairs and let $\mathcal{D}^v$ denote the disjunctive pairs at intersection $v \in V$. Writing `conj(...)` and `disj`$(\dots)$ for the usual conjunctive and disjunctive constraints, we propose to solve the optimization problem

$$\min_{y} \quad \sum_{i \in \mathcal{N}} \sum_{v \in \mathcal{R}(l(i))} y(i, v) \tag{3a}$$

$$\text{s.t.} \quad r_i \leq y(i, v_0(l(i))) \qquad \text{for } i \in \mathcal{N}, \tag{3b}$$

$$\texttt{conj}(y(i, v), y(j, v)) \qquad \text{for } (i, j) \in \mathcal{C}, v \in V, \tag{3c}$$

$$\texttt{disj}(y(i, v), y(j, v)) \qquad \text{for } \{i, j\} \in \mathcal{D}^v, v \in V, \tag{3d}$$

$$y(i, v) + d(v, w) \leq y(i, w) \qquad \text{for } i \in \mathcal{N}(l), (l, v, w) \in E, \tag{3e}$$

$$y(i, w) + \rho(v, w) \leq y(j, v) \qquad \text{for } (i, j, v, w) \in \mathcal{F}, \tag{3f}$$

where $\mathcal{F}$ is defined as

$$\mathcal{F} = \{(i, j, v, w) : i, j \in \mathcal{N}(l), k(i) + c(v, w) = k(j), (l, v, w) \in E\}$$

and we have $\rho(v, w) = c(v, w)L - d(v, w)$ from Proposition 1. Each $(i, j, v, w) \in \mathcal{F}$ represents a pair of vehicles driving on the same lane $(v, w)$, for which the first vehicle must have made enough space in the lane before vehicle $j$ can enter.

## Overlapping routes

In problem 3, note that the conjunctions $\mathcal{C}$ did not depend on the intersection like the disjunctions, because the conjunctive "chains" stays the same along the route. Whenever we drop Assumption 2, it becomes possible for vehicles from different classes to merge onto the same lane. Once one the same lane, they also need to maintain their relative order, so conjunctive constraints need to be introduced depending on which disjunctive decisions were made at an upstream intersection.

Note that the formulation of the capacity constraints (3f) hinges on the chains of conjunctive arcs in the disjunctive graph. In other words, the pairs of vehicles involved in these "delayed conjunctive constraints" does not depend on the ordering of vehicles at intersections. Once we drop Assumption 2, these delayed constraints can also appear among vehicles from different routes, depending on their ordering. In principle, it is possible to add these conditional constraints to the MILP. However, we expect that even encoding all of them requires too much computing time and memory.

## Related scheduling problems

We now show that the above problem can be interpreted as a variation of the classical job-shop scheduling problem (JSSP) with setup-times. Intersections play the role of machines and each vehicle is modeled by a job, whose operations correspond to crossing the intersections along its route. The following three aspects distinguish the current problem from classical JSSP.

First of all, we assume that vehicles cannot overtake other vehicles on the same lane, so the order in which vehicles arrive to a lane needs to be maintained at all downstream intersections. This introduces a set of additional precedence constraints that are not required in general JSSP.

Furthermore, note that constraints (3e) model travel delays between intersections. For some networks, the problem can be transformed back to JSSP without travel delays by an appropriate time shifting procedure. In the general case, such a transformation is also possible by introducing additional machines that explicitly model the *travel operation*. Such travel delays have been studied before in a model where a fleet of autonomous vehicles moves operations between a set of machines [**?**].

The third extensions involves the limited buffer size at machines, which has been considered before [**?**]. With these kind of constraints, the problem becomes stricly more general than JSSP.

## Constructive neural heuristic

Like we did for the single intersection, we are now going to define heuristics in terms of choosing which vehicle may cross next on which intersection. Again, we model this process as a deterministic finite-state automaton, with input alphabet given by

$$\Sigma = \{(l, v) : l \in \{1, \dots, n\}, v \in V\},$$

where $n$ is the number of classes. Let $v \in V$ be some intersection, then $\pi(v)$ denotes the local crossing order of intersection $v$, analogously to $\pi$ in the single intersection case. Let $s$ be an instance of network problem (3), then the state $(s, \pi)$ encodes the local crossing order $\pi(v)$ for each intersection $v$. Let $n(l, v)$ denote the number of scheduled vehicles from class $l$ at intersection $v$. Every action $(l, v)$ is a pair of a class $l$ and some intersection $v \in \mathcal{R}(l)$ and corresponds to appending the next unscheduled vehicle from class $l$ to the local ordering $\pi(v)$.

We keep track of the lower bounds $\mathrm{LB}_\pi(i, v)$ for every vehicle-intersection pair, where $\pi$ denotes the current global (partial) ordering, consisting of the collection of all local orderings $\{\pi(v) : v \in V\}$. Because all constraints are of the form

$$y(i, v) + \alpha(i, v, j, w) \leq y(j, w),$$

they can be thought of as arcs with weight $\alpha(i, v, j, w)$ in the disjunctive graph, which has a node for every vehicle-intersection pair $(i, v)$. In this way, the lower bounds are simply defined as

$$\mathrm{LB}_\pi(i, v) = \max_{(j,w) \in N_\pi^-(i,v)} \mathrm{LB}_\pi(j, w) + \alpha(i, v, j, w).$$

As in the single intersection problem, a complete ordering $\pi$ provides us with a crossing time schedule $y(i,v) = \mathrm{LB}_\pi(i,v)$. For every (partial) ordering $\pi$, the lower bounds can simply be computed by solving a linear program. Alternatively, the structure of the disjunctive graph can be used for a more direct calculation, as we show next.

**Proposition 2.** *The empty schedule $\pi = \varnothing$ has lower bounds*

$$\mathrm{LB}_\varnothing(i,v) = r_i + d(v_0(l(i)), v). \tag{4}$$

*Proof.* Since there are no disjunctive constraints in $G(\varnothing)$, we can consider each class $l$ separately. Because we assumed that $r_i + L \leq r_j$ for every $(i,j) \in \mathcal{C}$, the conjunctive constraints at $v_0(l)$ must hold. Furthermore all travel constraints are tight in this definition of $\mathrm{LB}_\varnothing$. We show that conjunctive constraints at downstream intersections are satisfied by induction on $v \in \mathcal{R}(l)$. Let $(i,j) \in \mathcal{C}$ and let $(v,w)$ be an edge of route $\mathcal{R}(l)$, then

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

It remains to show that all buffer constraints are satisfied. Consider a pair of vehicles $i = (l,k)$ and $j = (l, k+c(v,w))$. By the chain of conjunctive constraints between $i$ and $j$, we have

$$\mathrm{LB}_\varnothing(i,v) + c(v,w)L \leq \mathrm{LB}_\varnothing(j,v).$$

From the travel constraints, we have

$$\mathrm{LB}_\varnothing(i,w) - d(v,w) = \mathrm{LB}_\varnothing(i,v).$$

Together, these two equations imply the buffer constraint

$$\mathrm{LB}_\varnothing(i,w) + c(v,w)L - d(v,w) \leq \mathrm{LB}_\varnothing(j,v). \qquad \square$$

Starting from $\mathrm{LB}_\varnothing$, we update the lower bounds every time a new vehicle is added to $\pi$. Often, only a subset of the lower bounds is affected, so we propagate only the necessary changes through the disjunctive graph. For $i = (l,k) \in \mathcal{N}(l)$, let $\mathrm{next}(i) = (l, k+1)$ denote its successor in the conjunctive chain, if it exists. Similarly, we use the notation $\mathrm{prev}(v)$ and $\mathrm{next}(v)$ for the previous and next intersection, respectively, on the route $\mathcal{R}(l)$, if they exist. We write the assignment operation $x \xleftarrow{\max} y$ as a shorthand for $x \leftarrow \max(x,y)$.

**Proposition 3.** *Algorithm 1 maintains the correct values of $LB_\pi$ whenever some vehicle $i$ is added to local ordering $\pi(v)$.*

*Proof.* For all lower bounds involving $l(i)$, nothing changes. For each class at $v$ other than $l(i)$, we process the disjunctive arc towards the next unscheduled vehicle $j$ by setting

$$\mathrm{LB}_\pi(j,v) \xleftarrow{\max} \mathrm{LB}(i,v) + \sigma.$$

Then for each $l(j)$ of these other classes, we propagate the changes through the disjunctive graph as follows. $\square$

**Algorithm 1** Update $\mathrm{LB}_\pi$ after vehicle $i$ is added to $\pi(v)$, by propagating changes over the disjunctive graph, under Assumption 2.

**Parameterization**

Show how to build a suitable embedding of the current state, based on lower bounds $\mathrm{LB}_\pi$.

# Traffic as job-shop scheduling

The Job-Shop Scheduling Problem (JSSP) is a widely studied problem in which a set of $n$ jobs must be assigned to non-overlapping time slots on a set of $m$ machines. Each job $i$ has a set of $n_i$ operations $O_{i1}, \ldots, O_{in_i}$ that need to be executed in this order. Each operation $O_{ij}$ requires $p_{ij}$ processing time on machine $M_{ij}$. Each machine can process at most one operation and early preemption is not allowed. The task of the scheduler is to determine a valid schedule of start times $y_{ij}$ for each operation, while minimizing some objective function. Let $C_{ij} = y_{ij} + p_{ij}$ denote the *completion time* of operation $O_{ij}$, then the *makespan* objective is given by the latest completion time

$$\max_{i,j} C_{ij}.$$

Another objective is the *total completion time*, given by

$$\sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij},$$

which may be intuitively thought of as representing some sort of total cost of inventory, assuming we need to physically store jobs somewhere.

A commonly used representation of JSSP instances is the *disjunctive graph* $G = (\mathcal{O}, \mathcal{C}, \mathcal{D})$, with vertices $\mathcal{O} = \{O_{ik} : 1 \leq i \leq n, 1 \leq k \leq n_i\}$ corresponding all the operations. The set of *conjunctive arcs* $\mathcal{C}$ encodes all the precedence constraints $O_{i,k} \to O_{i,k+1}$ among each job's operations. The set of *disjunctive edges* $\mathcal{D}$ consists of undirected edges between each pair of operations from distinct jobs that need to be processed on the same machine, effectively encoding all such *conflicts*. Each valid schedule induces an ordering of operations on machines that is encoded by fixing the direction of each disjunctive edge such that we obtain a direct acyclic graph.

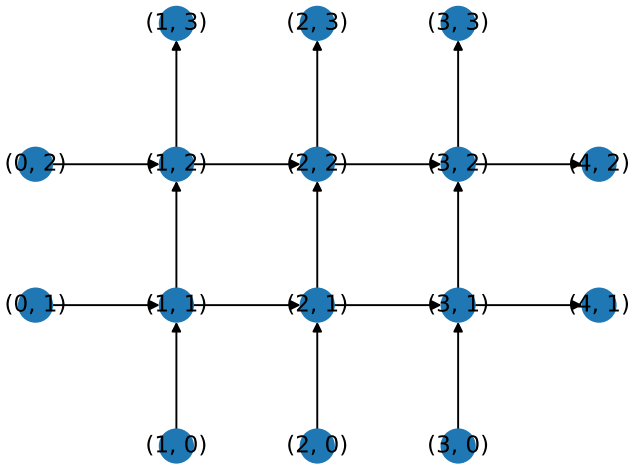


Figure 3: Example of connectivity graph of traffic network.

## Implementation details

We set the capacity of each lane according to equation (1), except for lanes from an entry point. These so-called *entry lanes* have unlimited capacity, which allows us to specify any kind of arrival time pattern.