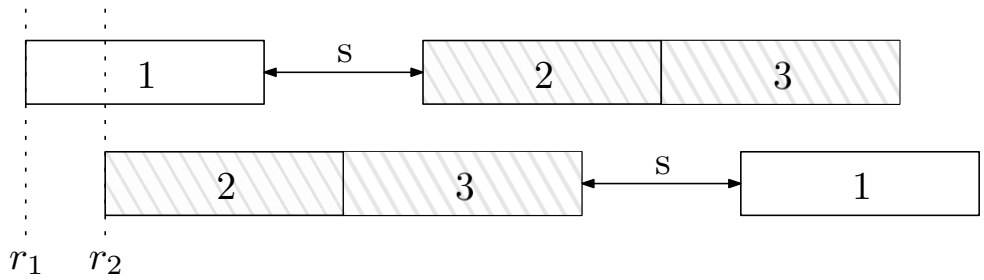
Trajectory Planning

Jeroen van Riel

February 2024

Offline crossing time scheduling



- given schedule y, there always exist trajectories x that are *safe*
- solve a MILP to obtain optimal y
- heuristic methods
  - polling policies such as exhaustive, gated or k-limited
  - *learning* method such as RL

Online trajectory planning (Miculescu & Karaman)

- recalculate trajectories upon new arrivals
- *regular* polling policy
- MotionSynthesize produces *safe* trajectories

$$
\begin{aligned}
&\texttt{MotionSynthesize}(z_{i,k}(t_0'), t_0', t_f', y) := \\
&\underset{x:\,[t_0',t_f']\to\mathbb{R}}{\arg\min} \quad \int_{t_0}^{t_f} |x(t)|dt \\
&\text{subject to} \quad \ddot{x}(t) = u(t), \text{ for all } t \in [t_0', t_f']; \\
&\qquad\qquad 0 \le \dot{x}(t) \le v_m, \text{ for all } t \in [t_0', t_f']; \\
&\qquad\qquad |u(t)| \le a_m, \text{ for all } t \in [t_0', t_f']; \\
&\qquad\qquad |x(t) - y(t)| \ge l, \text{ for all } t \in [t_0', t_f]; \\
&\qquad\qquad x(t_0') = x_{i,k}(t_0'); \quad \dot{x}(t_0') = \dot{x}_{i,k}(t_0'); \\
&\qquad\qquad x(t_f') = 0; \quad \dot{x}(t_f') = v_m,
\end{aligned}
$$

Offline crossing time scheduling in network

- problem like classical job-shop
- solve as MILP
- heuristic methods
  - Zhang et al. train an RL agent to constructs a job-shop schedule by adding/removing arcs to the corresponding disjunctive arc
  - their method might also work in an online setting

Trajectory planning in network

- need to consider finite space between intersection
- finite buffers can be modeled in the MILP (under some assumptions on vehicle routes)

- it is not clear that safe trajectories always exist for some crossing time schedule y, as was shown for the single intersection case

Online trajectory planning in network

- generate trajectories based on a crossing time schedule
- alternatively, the control problem can be stated directly in terms of (approximations of) trajectories
  - idea: determine the waiting time on fixed *locations*