Intersection Coordination with Priority-Based Search for Autonomous Vehicles

Jiaoyang Li,[1] The Anh Hoang,[2] Eugene Lin,[3] Hai L. Vu,[2] Sven Koenig[3]

[1] Robotics Institute, Carnegie Mellon University, USA
[2] Department of Civil Engineering, Monash University, Australia
[3] Computer Science Department, University of Southern California, USA
jiaoyangli@cmu.edu, the.hoang@monash.edu, eugenezlin@gmail.com, hai.vu@monash.edu, skoenig@usc.edu

Abstract

The development of connected and autonomous vehicles opens an opportunity to manage intersections without signals. One promising approach is to use a central autonomous intersection manager to optimize the movement of the vehicles in the intersection. Existing work uses Mixed Integer Linear Programming (MILP) to find optimal solutions for this problem but is time-consuming and cannot be applied in real-time. On the other hand, the coordination of the vehicles is essentially a Multi-Agent Path Finding (MAPF) problem, for which dozens of efficient algorithms have been proposed in recent years. Inspired by these MAPF algorithms, we propose a three-level algorithm called PSL to solve the intersection coordination problem. Theoretically, PSL is complete and polynomial-time in the number of vehicles. Empirically, PSL runs significantly faster with only a slight compromise in the solution quality than the optimal MILP method. It also generates significantly better solutions with a slightly larger runtime than the traditional First-Come-First-Served strategy.

Introduction

The development of Connected and Autonomous Vehicles (CAVs) technology opens a new opportunity to manage vehicles on roads, particularly vehicles crossing intersections. With the assumption that a full fleet of CAVs will be on the road in the future, the concept of signal-free intersections has attracted researchers in the transportation field (Dresner and Stone 2004; Zhong, Nejad, and Lee 2021). The early proposal of signal-free intersections was based on reservation strategies. All approaching vehicles communicate with a central autonomous Intersection Manager (IM). The IM receives reservation requests from vehicles and accepts a request if it has no collisions with the previous reservations. An early implementation of the reservation-based strategies was First Come First Served (FCFS) (Dresner and Stone 2004, 2008; Au and Stone 2010; Au, Shahidi, and Stone 2011; Li et al. 2013), where the IM assets the priority of the vehicles using the temporal order of the received requests.

However, such FCFS strategies can lead to poor solution quality, particularly in the scenario with high traffic demands. For instance, FCFS was shown to increase delays beyond conventional signals in specific scenarios (Levin, Boyles, and Patel 2016). For that reason, several studies proposed optimization-based strategies (Lee and Park 2012; Zohdy and Rakha 2016). Noticeably, Levin and Rey (2017) developed a Mixed Integer Linear Programming (MILP) model to determine the optimal coordination strategies for a single intersection. It produced significantly better solutions than the FCFS strategy but was computationally inefficient and thus cannot be used in real-time. Refer to (Zhong, Nejad, and Lee 2021) for a thorough survey on different coordination strategies for autonomous intersection management.

The core challenge here is to efficiently and effectively resolve collisions among many vehicles, which is close to Multi-Agent Path Finding (MAPF) (Stern et al. 2019), a problem widely studied in AI and robotics. Dozens of MAPF algorithms have been proposed in recent years, which, for example, can find optimal solutions for hundreds of agents (Li et al. 2021a) and near-optimal solutions for a thousand agents (Li, Ruml, and Koenig 2021) within just a minute. Inspired by the MAPF algorithms, we propose a three-level algorithm PBS-SIPP-LP (PSL): Level 1 uses Priority-Based Search (PBS) (Ma et al. 2019) to resolve collisions between vehicles; Level 2 uses a modified version of Safe Interval Path Planning (SIPP) (Phillips and Likhachev 2011) to plan optimal paths for individual vehicles; and Level 3 uses a Linear Programming (LP) model to optimize the entry time and speed of each vehicle.

Our main contribution is as follows: (1) We apply MAPF algorithms to the intersection coordination problem and show promise compared to the existing intersection coordination algorithms. Specifically, PSL runs significantly faster than MILP and is suitable in real-time. It finds near-optimal solutions, which are substantially better than the solutions of FCFS. (2) We combine search- and optimization-based methods to optimize the trajectories and speeds of the vehicles simultaneously (while existing MAPF algorithms do not optimize speeds). (3) We prove that PSL is complete and polynomial-time in the number of vehicles (while PBS for MAPF is neither complete nor polynomial-time).

Background

Intersection Setup

Levin and Rey (2017) introduced a conflict-point representation for single intersections and modeled the intersection co-

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

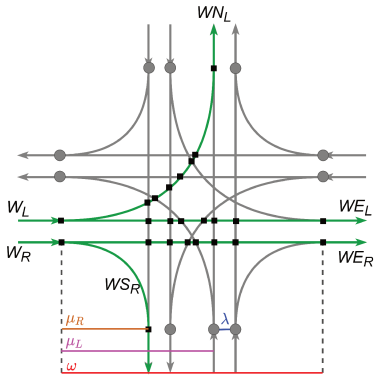


Figure 1: Illustration of a 4-approach, 2-lane intersection, borrowed from (Levin and Rey 2017). Four vehicle trajectories WS_R, WE_R, WE_L, and WN_L are available for the West approach, denoted by the four green lines. We mark all conflict points on each green line with a black square. Vehicles are not allowed to have sharp turns. For example, vehicles coming from Lanes W_L and W_R are not allowed to switch to any grey lines via the conflict points.

ordination problem as a *Conflict Point Intersection Control* (CPIC) problem. Figure 1 shows an example. Each straight or curved line represents a potential trajectory for a vehicle to move from an entry lane to an exit lane. The intersection of any two trajectories is marked as a *conflict point*, which can be occupied by at most one vehicle at any time. Our task is to coordinate the vehicles on this conflict-point graph so that no two vehicles collide at any conflict points.

The communication flow between the IM and a vehicle is as follows: When a vehicle approaches the intersection, it sends a RESERVATION request to the IM, which contains the information of the vehicle type, the entry and exit lanes, and the current position and speed. The IM then calculates the earliest time when it can enter the intersection. After periodically running an intersection coordination algorithm, such as our algorithm PSL, the IM sends the RESERVATION message to the vehicle, which contains necessary information such as entry time and speed to instruct the vehicle to go through the intersection safely. The communication ends with an ACCEPT message from the vehicle to the IM and an ACK message from the IM after that. More details can be found in (Levin and Rey 2017).

Multi-Agent Path Finding (MAPF)

MAPF (Stern et al. 2019) is the problem of planning collision-free paths for a team of agents. The standard MAPF problem takes as input an unweighted graph and a team of agents, each with a start vertex and a goal vertex. An agent can either move to an adjacent vertex or wait at its current vertex at each discretized timestep. A collision happens when two agents try to move to the same vertex or traverse the same edge in opposite directions at the same time. The task is to find a set of collision-free paths with the minimum sum of the travel times that move all agents from their start vertices to their goal vertices.

MAPF and CPIC problems are similar. But, to use MAPF algorithms to solve the CPIC problem, we have two challenges: (1) In MAPF, agents are point agents, while in CPIC, vehicles have shapes and need to keep safety distances from each other. Thus, vehicles can collide with each other even when they are not at the same vertex. (2) In MAPF, time is discretized, and agents move with binary speeds (i.e., either wait or move one vertex per timestep), while, in CPIC, time is continuous, and we need to optimize the speeds of the vehicles. In particular, vehicles are not allowed to wait in the intersection, and a minimum speed is usually required to prevent vehicles from spending arbitrarily long time in the intersection. To our knowledge, the only work that applies MAPF algorithms to the CPIC problem is (Skopkova, Bartak, and Svancara 2020), but it does not consider the above challenges and is thus not suitable for practical deployments.

Problem Formulation

We use the CPIC problem formulation from (Levin and Rey 2017). Consider a set of vehicles $\mathcal{V}$ and a single intersection using the conflict-point graph representation with sets of entry and exit lanes (= points) Γ^- and Γ^+. Each vehicle $i \in \mathcal{V}$ has a traveling request from an *entry lane* $\gamma_i^- \in \Gamma^-$ to an *exit lane* $\gamma_i^+ \in \Gamma^+$ with an *earliest entry time* $e_i \in \mathbb{R}^{\geq 0}$ (i.e., the earliest time when vehicle i can reach γ_i^-). We assume that each vehicle i travels at a uniform speed through the intersection, but its speed $u_i \in [\underline{U}_i, \overline{U}_i]$ is a decision variable, where $\underline{U}_i \in \mathbb{R}^{>0}$ and $\overline{U}_i \in \mathbb{R}^{>0}$ ($\overline{U}_i \geq \underline{U}_i$) are the given minimum and maximum speed for vehicle i, respectively.

We refer to the intersection as the area after the vehicles pass stop lines. That is to say, vehicles are allowed to wait before they reach the entry lanes but must cruise with the given speed once enter the intersection. The graph in Figure 1 is a typical traditional intersection graph where the trajectory (i.e., sequence of conflict points) for every pair of entry and exit lanes is already specified. We will use this graph in the experiments because our baseline algorithms from (Levin and Rey 2017) work for only such graphs. Nevertheless, our algorithm PSL can work for more general graphs where there are multiple different trajectories for every pair of entry and exit lanes.

We use $t_i(c) \in \mathbb{R}^{\geq 0}$ to denote the time when vehicle i reaches (conflict) point c. By definition, we have $t_i(\gamma_i^-) \geq e_i$. We use $d_i(c_j, c_k) \in \mathbb{R}^{>0}$ to denote the distance for which vehicle i travels from points c_j to c_k. That is, $t_i(c_j) - t_i(c_k) = d_i(c_j, c_k)/u_i$. The time duration $\tau_i(c) \in \mathbb{R}^{>0}$ for which vehicle i occupies point c is calculated as $\tau_i(c) = l_i(c)/u_i + l_i(c)/w$, where $l_i(c) \in \mathbb{R}^{\geq 0}$ is the total length that vehicle i goes around point c and $w \in \mathbb{R}^{\geq 0}$ is the congested wave speed.[1] (More details of this equation can be found in (Levin and Rey 2017).) Thus, vehicle i completely passes through point c at time $t_i(c) + \tau_i(c)$. We call $t_i(\gamma_i^-)$ and $t_i(\gamma_i^+) + \tau_i(\gamma_i^+)$ the *entry* and *exit time* of vehicle

[1] Congested wave, also known as shockwave, is a term used in the transportation field that originates from a sudden, substantial change in the traffic flow state. It can be observed, for example, when a vehicle stops at a stop line forming a queue where its rear grows in the opposite direction of the traveling traffic.

i, respectively. When vehicles i and j move through point c, a *collision* occurs iff time intervals $[t_i(c), t_i(c) + \tau_i(c))$ and $[t_j(c), t_j(c) + \tau_j(c))$ overlap. Vehicles are not allowed to overtake each other in the intersection: If vehicles i and j are in the same entry lane with $e_i < e_j$, then $t_i(c) < t_j(c)$ holds for all points c that both vehicles visit.

Our task is to plan for each vehicle i a *path*, namely a sequence of conflict points, starting at γ_i^- and ending at γ_i^+, associated with a *reserved interval* $[t_i(c), t_i(c) + \tau_i(c))$ for each point c in the sequence, so that the vehicles neither overtake nor collide with each other and the sum of their exit times is minimized.

PBS-SIPP-LP (PSL)

Our algorithm PSL consists of three levels: Level 1 uses PBS (Ma et al. 2019) to address the collisions between the paths of the vehicles by generating spatio-temporal constraints, where the paths of the vehicles are generated by Level 2; Level 2 uses a modified version of SIPP (Phillips and Likhachev 2011) to plan an optimal path for a vehicle with respect to the spatio-temporal constraints generated by Level 1, where the entry time and the speed of the path are determined by Level 3; and Level 3 uses an LP model to optimize the entry time and the speed of a vehicle.

Level 1: PBS

Prioritized planning is a widely-used MAPF algorithm. It first sorts the agents by a predefined total priority ordering and then plans a path with the minimum travel time for each agent, from the highest priority to the lowest priority, that avoids collisions with the paths of all higher priority agents. The FCFS strategy can be viewed as a prioritized planning algorithm that sorts the vehicles by their earliest entry times.

Prioritized planning is simple and runs fast. But its solution quality is sensitive to the predefined total priority ordering. Priority-Based Search (PBS) (Ma et al. 2019) overcomes this drawback by searching the space of all possible priority orderings. We choose to use PBS instead of other MAPF algorithms for three reasons: (1) PBS is efficient. For example, it can plan paths for 200 agents within a second in an offline setting (Ma et al. 2019) and for 800 agents within five seconds in an online setting (Li et al. 2021b). So it fits our requirement of real-time planning. (2) PBS finds near-optimal solutions empirically. For example, the sum of the travel times of its solution is always less than 5% worse than the optimal on the MAPF benchmark suite (Ma et al. 2019), which significantly outperforms other prioritized planning methods. So it can coordinate the vehicles in the intersection with high throughput. (3) PBS or, in general, prioritized planning is flexible and can be easily adapted to different agent models without sacrificing its efficiency too much. For example, it can be used for differential drive robots and general agents with nonlinear (Yakovlev, Andreychuk, and Vorobyev 2019) or nonholonomic (Chen et al. 2021) dynamics. So it can be adapted to our vehicle models efficiently.

We adapt PBS to solving our problem as follows. Algorithm 1 shows the pseudo-code. We search a binary *Priority Tree* (PT) in a depth-first manner. Each PT node contains a set of (partial) priority orderings and a set of paths, one for each vehicle, that is *consistent* with its priority orderings. That is, if two vehicles i and j have a priority ordering $i \prec j$ (indicating that vehicle i has higher priority than vehicle j), then the path of vehicle j has to avoid collisions with the path of vehicle i. If two vehicles do not have a priority ordering, then their paths are allowed to collide with other, and the collisions will be resolved in descendant PT nodes.

Algorithm 1: Level 1 of PSL.

```
1  Root ← GENERATEROOT();
2  STACK ← {Root};
3  while STACK ≠ ∅ do
4      N ← STACK.pop();
5      if N.collisions = ∅ then return N.plan;
6      (i, j) ← a collision in N.collisions;
7      N_i, N_j ← N;                          // Two copies of N
8      for (x, y) ∈ {(i, j), (j, i)} do
9          ≺_{N_x} ← ≺_{N_x} ∪ {y ≺ x};
10         UPDATEPLAN(N_x);
11     Insert N_i and N_j into STACK in descending order of
         the sum of the exit times of their paths;
12 return "No Solution";
```

The root PT node [Line 1] contains an initial set of priority orderings with respect to the earliest entry times of the vehicles, namely $i \prec j$ iff $\gamma_i^- = \gamma_j^- \wedge e_i < e_j$. When expanding a PT node, we first check whether its paths contain any collisions. We terminate and return its paths if there are no collisions [Line 5]. Otherwise, we choose one collision, say between vehicles i and j [Line 6], and resolve it by generating two child PT nodes: one with an added priority ordering $i \prec j$ and the other one with an added priority ordering $j \prec i$ [Lines 7 to 9]. We call the UPDATEPLAN function to make the paths of each child PT node consistent with its set of priority orderings [Line 10]. That is, if the paths of any two vehicles x and y with $x \prec y$ have collisions, we call Level 2 to replan the path for vehicle y. In order to avoid replanning the path for the same vehicle twice, we topologically sort the vehicles based on the priority orderings of the child PT node and check the inconsistency in the resulting order. Please refer to the original PBS paper (Ma et al. 2019) for more details. Finally, we finish this iteration and will expand the child PT node whose paths have a smaller sum of the exit times first in the next iteration [Line 11].

Level 2: SIPP

The task of Level 2 is to plan a path with the minimum exit time for a given vehicle $i \in \mathcal{V}$ that does not collide with a given set of paths (which are the paths of the vehicles with higher priority than vehicle i). Different from traditional pathfinding problems in the MAPF literature, our problem needs to determine not only the sequence of conflict points that vehicle i needs to visit but also its entry time and speed (the reserved intervals $[t_i(c), t_i(c) + \tau_i(c))$ for each point c in the sequence can be then computed accordingly). Since vehicles move in continuous time, traditional single-agent pathfinding algorithm space-time A* (Silver 2005) (which

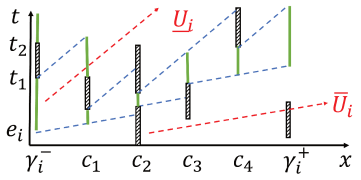


Figure 2: Illustration of our modified SIPP. The x-axis represents the distance traveled by vehicle i, with $\gamma_i^-, c_1, \cdots, c_4, \gamma_i^+$ being the sequence of conflict points. The y-axis represents the time. The shadowed strips are the time intervals reserved by the given paths at every point. The green line segments are the generated safe intervals.

searches in discrete timesteps) cannot be applied. We therefore achieve this goal by first using a modified version of Safe Interval Path Planning (SIPP) (Phillips and Likhachev 2011) to find a sequence of conflict points, each with a safe (time) interval during which vehicle i does not collide with any given path, and then calling Level 3 to specify the entry time and the speed so that vehicle i passes through each point c with its reserved interval $[t_i(c), t_i(c) + \tau_i(c))$ within the corresponding safe interval.

SIPP performs an A* search on a time-interval graph where each state in the graph is defined by two elements: a location (= conflict point) and a safe (time) interval, representing that a particular location is not reserved by any given paths during the safe interval. We use Figure 2 to illustrate how our modified SIPP works. We assume that vehicle i occupies each point c for only an instant of time (instead of a duration of time $\tau_i(c)$) and can vary its speed arbitrarily as long as the speed is always within $[\underline{U}_i, \overline{U}_i]$ (instead of stick to a constant speed). This assumption will be corrected in Level 3. Vehicle i can be at entry lane γ_i^- at or after its earliest entry time e_i. To avoid colliding with the reserved interval $[t_1, t_2)$ at γ_i^-, we generate two safe intervals $[e_i, t_1)$ and $[t_2, +\infty)$. Let us consider the first safe interval $[e_i, t_1)$. When vehicle i moves from γ_i^- to c_1 with a speed within $[\underline{U}_i, \overline{U}_i]$, it reaches c_1 within interval $[e_i + t_{min}, t_1 + t_{max})$, where t_{min} and t_{max} are the travel times of vehicle i from γ_i^- to c with speeds $\overline{U}_i$ and $\underline{U}_i$, respectively. As a result, we generate two safe intervals at c_1, indicated by the two green line segments at c_1 in the figure. If we consider the earlier safe interval at c_1 and repeat this procedure for the remaining points, we will eventually generate a safe interval at exit lane γ_i^+, indicated by the green line segment at γ_i^+ in the figure. We then backtrack from this interval to get a sequence of conflict points from γ_i^- to γ_i^+, each with a safe interval, and pass them to Level 3. Level 3 then determines whether there exists a pair of an entry time and a speed such that vehicle i visits each point within its safe interval. If it exists, then Level 3 returns the pair that minimizes the exit time, and we successfully find a path. We continue searching other SIPP nodes until we provably find an optimal path.

Algorithm 2 shows the pseudo-code. We build a safe interval table $\mathcal{T}$ [Line 1], which is a hash table that maps each conflict point to a set of safe intervals. Specifically, to avoid overtaking, we reserve interval $[0, t_j(c) + \tau_j(c))$ for

Algorithm 2: Level 2 of PSL for vehicle i.

```
1  T ← buildSafeIntervalTable();
2  OPEN ← ∅;
3  for [lb, ub) ∈ T[γi−] : ub > ei do
4      insert Node(γi−, [max{lb, ei}, ub)) into OPEN;
5  p* ← NULL;                          // The best path seen so far
6  while OPEN ≠ ∅ do
7      remove node n with the smallest f-value from OPEN;
8      if f(n) ≥ exit_time(p*) then break;
9      if loc(n) = γi+ then                // Reach the exit lane
10         N ← extractAncesterNodes(n);
11         p ← Level_3(N);
12         if exit_time(p) < exit_time(p*) then p* ← p;
13         continue;
14     for y ∈ getNextConflictPoints(n) do
15         t1 ← lower_bound(n) + di(loc(n), y)/Ui;
16         t2 ← upper_bound(n) + di(loc(n), y)/Ui;
17         for [lb, ub) ∈ T[y] : t1 < ub ∧ lb < t2 do
18             insert Node(y, [max{lb, t1}, min{ub, t2}))
                 into OPEN;
19 return p*;
```

every point c in the path of every vehicle $j \in \mathcal{V}$ subject to $\gamma_j^- = \gamma_i^- \wedge e_j < e_i$. To avoid colliding with any higher priority vehicle j, we reserve interval $[t_j(c), t_j(c) + \tau_j(c))$ for every point c in its path. After we make all reservations, we store the compliments of the reserved intervals in $\mathcal{T}$.

OPEN [Line 2] is a regular open list of A* that sorts its SIPP nodes in ascending order of their f-values. The f-value of a SIPP node n is the sum of its g-value and h-value, where the g-value is the lower bound of its safe interval (which is the earliest time that vehicle i can reach its location $loc(n)$), and the h-value is the minimum travel time from $loc(n)$ to γ_i^+ (i.e., moving from $loc(n)$ to γ_i^+ with speed $\overline{U}_i$). In other words, the f-value of n is a lower bound on the exit times of all *corresponding paths* of n, i.e., all possible paths from entry lane Γ_i^- to exit lane Γ_i^+ that pass through $loc(n)$ and the locations of the ancestor SIPP nodes of n within the corresponding safe intervals. Since vehicle i can be at entry lane γ_i^- at any time no earlier than e_i, we generate a SIPP node for each of the safe intervals at γ_i^- that overlaps with interval $[e_i, +\infty)$ and insert it into OPEN [Lines 3 to 4].

At each iteration, we select the SIPP node n with the smallest f-value from OPEN [Line 7]. The f-value of n is a lower bound on all corresponding paths of all SIPP nodes in OPEN, so when it is no smaller than the exit time of the best path seen so far p^*, p^* is provably an optimal path. We thus terminate [Line 8] and return path p^* [Line 19]. Function *exit_time*(p) returns $+\infty$ if path p does not exist. If n is at the exit lane [Line 9], then we backtrack all its ancestor SIPP nodes [Line 10] and call Level 3 to find the corresponding path with the minimum exit time [Line 11]. We then update p^* if necessary [Line 12]. If SIPP node n is not at the exit lane, we consider each of the next conflict point y that can be reached from n (given the orientation of vehicle i) and compute the earliest and latest arrival times t_1 and t_2

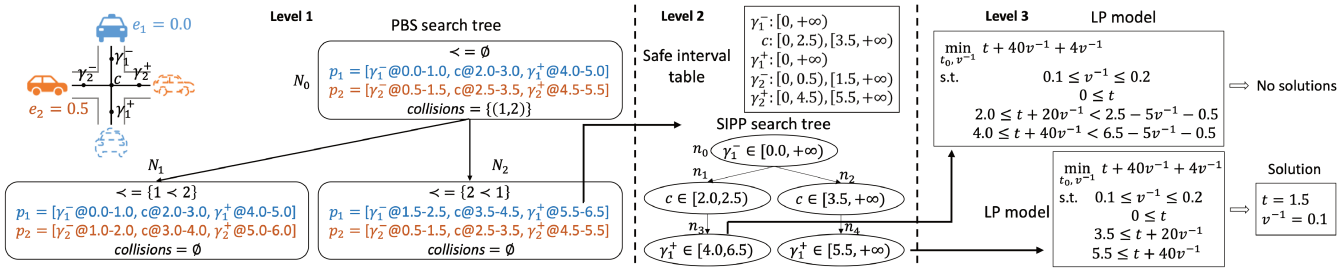


Figure 3: Running example of PSL on a two-vehicle instance shown at the top-left corner. A path entry at point x with reserved interval $[t_1, t_2)$ is marked as $x@t_1 - t_2$. A SIPP node at point x with safe interval $[t_1, t_2)$ is marked as $x \in [t_1, t_2)$. The distance from every entry and exit lane to point c is 20m. The minimum and maximum speeds for both vehicles are 5m/s and 10m/s, respectively. The time that each vehicle $i \in \{1, 2\}$ spent at point $x \in \{\gamma_i^-, \gamma_i^+, c\}$ is computed by $\tau_i(x) = \frac{5}{u_i} + 0.5$.

at y [Lines 14 to 16]. Then, for each safe interval at y that overlaps with interval $[t_1, t_2)$ [Line 17], we generate a SIPP node with its safe interval being the intersection of the two intervals and insert it to OPEN [Line 18].

Level 3: LP

We use a Linear Programming (LP) model to determine if we can find a path with a constant speed such that the duration for which vehicle i occupies each point is within the corresponding safe interval. We denote the sequence of conflict points by $c_0 = \gamma_i^-, c_1, \cdots, c_{m-1}, c_m = \gamma_i^+$ and the safe interval for c_j $(j = 0, \cdots, m)$ by $[lb_j, ub_j)$. We build the following LP model to optimize two variables, namely the entry time $t = t_i(\gamma_i^-)$ and the inverse speed $v^{-1} = 1/u_i$.

$$\min_{t, v^{-1}} t + \sum_{j=1}^{m} d_i^j v^{-1} + l_i^m v^{-1} \quad (1)$$

$$\text{s.t.} \quad 1/\overline{U}_i \le v^{-1} \le 1/\underline{U}_i \quad (2)$$

$$lb_0 \le t < ub_0 - l_i^0 v^{-1} - l_i^0/w \quad (3)$$

$$lb_j \le t + \sum_{j'=1}^{j} d_i^{j'} v^{-1} < ub_j - l_i^j v^{-1} - l_i^j/w, \quad \forall\, 1 \le j \le m, \quad (4)$$

where $d_i^j = d_i(c_{j-1}, c_j)$ and $l_i^j = l_i(c_j)$. We have $t_i(c_j) = t + \sum_{j'=1}^{j} d_i^{j'} v^{-1}$ and $\tau_i(c_j) = l_i^j v^{-1} + l_i^j/w$. Thus, Equation (1) minimizes the exit time of vehicle i since it equals to $t_i(\gamma_i^+) + \tau_i(\gamma_i^+) - l_i(\gamma_i^+)/w$, where the last term is a constant in this situation. Inequality (2) ensures that the speed of vehicle i is within the speed limit. Inequalities (3) and (4) are the safe interval constraints at the first and the remaining conflict points, respectively, that ensure that vehicle i reaches each point c_j at or after lb_j and leaves it before ub_j.

Running Example

We show a running example in Figure 3 to illustrate how PSL works as a whole. We generate a root PT node N_0 at Level 1. Since no two vehicles share the same entry lanes, PT node N_0 contains an empty set of partial priority ordering $\prec_{N_0} = \emptyset$ and a path $N_0.p_i$ that enters the intersection at the earliest entry time and moves with the maximum speed for each vehicle i. We omit the steps in Level 2 and Level 3 to generate these paths. Since vehicles 1 and 2 collide at point c, we have $N_0.collisions = \{(1, 2)\}$. Then, we expand PT node N_0 and generate two child PT nodes N_1 and N_2 with an additional partial priority $1 \prec 2$ and $2 \prec 1$, respectively.

Let us focus on PT node N_2 first. Since vehicle 1 collides with vehicle 2 and has a lower priority than vehicle 2, we call Level 2 to replan its path. At Level 2, we first build a safe interval table that excludes all the time intervals used by vehicle 2. We then start the SIPP search by generating root SIPP node n_0 at point γ_1^- with interval $[0, +\infty)$. There are two safe intervals at point c in the safe interval table. We thus generate two child SIPP nodes n_1 and n_2 with intervals $[2.0, 2.5)$ and $[3.5, +\infty)$, respectively. We next expand n_1 as it has a smaller f-value than n_2. We generate n_3 at point γ_1^+ with interval $[4.0, 6.5)$. Since it is at the exit lane, we call Level 3 to generate a path. However, the corresponding LP model shown in the upper box in Level 3 does not have any solutions. We thus continue searching and expand n_2. We generate n_4 at exit lane γ_1^+ with safe interval $[5.5, +\infty)$ and call Level 3 again. This time, the corresponding LP model is solvable, and we find an optimal solution with $t = 1.5$s and $v^{-1} = 0.1$s/m. Now, we terminate the SIPP search and return the optimal path $N_2.p_1$. We last set $N_2.collisions$ to be an empty set as the paths in N_2 are collision-free. We omit the details of generating PT node N_1 as it is analogical.

Next, we expand PT node N_1. Since the paths in N_1 are collision-free, we terminate and return its paths.

Theoretical Analysis

Lemma 1 (Completeness and Optimality of Level 3)**.** *Level 3 guarantees to find an optimal pair of an entry time and a speed (i.e., an entry time and a speed such that the vehicle visits each conflict point within the given safe interval and exits the intersection the earliest) if one exists and returns failure otherwise.*

We omit the proof for this lemma since it is trivial.

Lemma 2 (Completeness and Optimality of Level 2)**.** *Level 2 guarantees to find an optimal path (i.e., a path for vehicle i that does not collide with any vehicles that have higher priorities than vehicle i and exits the intersection the earliest) if one exists and returns failure otherwise.*

Proof. We first prove the completeness. $\mathcal{T}$ contains a finite number of safe intervals, so the search space of our modified

SIPP is finite, which indicates that Level 2 can terminate in finite time if no solution exists. If a solution exists, it is guaranteed to be found because we explore all reachable points during all reachable safe intervals. We then prove the optimality. Since the f-value of a SIPP node is provably a lower bound on the exit times of its corresponding paths, the smallest f-value of the SIPP nodes in OPEN, denoted as $f(n)$, is provably a lower bound on the exit times of the corresponding paths of the SIPP nodes in OPEN. Since p^* is the best path seen so far, when $f(n) \geq$ *exit_time*(p^*), no corresponding paths of the SIPP nodes in OPEN can have smaller exit times than p^*. That is, p^* is optimal. □

Level 2 is complete and optimal, so it can report failure in finite time for unsolvable instances. Nevertheless, the CPIC problem has special properties that ensure that the problem solved by Level 2 is always solvable (i.e., Level 2 never reports failure), as shown in Lemma 3.

Lemma 3 (Solvability of Level 2)**.** *The problem solved by Level 2 is always solvable. That is, there always exists a path for vehicle i that does not collide with the paths of the vehicles that have higher priorities than vehicle i.*

Proof. Let T be the largest exit time of the vehicles that have higher priority than vehicle i. The path for vehicle i that enters its entry lane at time $T + 1$ and moves to its exit lane with its largest speed does not collide with any vehicle that has higher priority than it. So the lemma holds. □

Theorem 1 (Completeness and Suboptimality of PSL)**.** *PSL is complete and suboptimal. That is, PSL can always find a solution within finite time, but the solution may not have the minimum sum of the exit times.*

Proof. We know from Lemmas 2 and 3 that Level 2 always returns a solution. So Level 1 can always successfully generate a root PT node and, when expanding a PT node, two child PT nodes. When it resolves a collision between two vehicles at PT node N, these two vehicles are guaranteed to be collision-free in all descendant PT nodes of N (otherwise, the paths in the descendant PT nodes are not consistent with their priority orderings). That is, along one branch, we need to resolve the collisions between any two vehicles at most once. Since the maximum number of pairs of vehicles that can collide is $|\mathcal{V}|(|\mathcal{V}| - 1)/2$ and Level 1 performs a depth-first search, it finds a PT node that contains collision-free paths within $|\mathcal{V}|(|\mathcal{V}| - 1)/2$ PT node expansions. Thus, PSL is complete. It is suboptimal because Level 1 uses a depth-first search and stops when finding the first solution. □

We have tested a version of PSL that uses best-first search, instead of depth-first search, in Level 1. But, it is still suboptimal because each vehicle optimizes its speed by minimizing the exit time, which might result in a non-minimum arrival time at some conflict point x along its path. This can delay the exit time of other vehicles that pass through x after this vehicle and thus lead to a suboptimal solution. Moreover, empirically, this best-first-search version runs substantially slower than the depth-first-search version with negligible improvements in solution quality in practice. We thus choose to use the depth-first-search version of PSL.

Lemma 4 (Search-Tree Size of Level 1)**.** *Level 1 generates at most $|\mathcal{V}|(|\mathcal{V}| - 1)$ PT nodes.*

This lemma can be derived from the proof of Thoerem 1.

Lemma 5 (Search-Tree Size of Level 2)**.** *Level 2 generates at most $2(D|\mathcal{V}|)^d$ SIPP nodes in each call, where D is the maximum number of adjacent conflict points a vehicle can reach from a given conflict point, and d is the maximum number of conflict points in a path from an entry to an exit lane.*

Proof. Since vehicles cannot move backward in an intersection and the conflict-point graph contains no circles, a vehicle can never visit the same point twice. Hence, in Level 2, each point can be reserved by at most $|\mathcal{V}| - 1$ times, one time per vehicle. That is, the number of safe intervals at each point in $\mathcal{T}$ is at most $|\mathcal{V}|$. Thus, when we expand a SIPP node during the SIPP search, we generate at most $D|\mathcal{V}|$ child SIPP nodes. Since the depth of the SIPP tree is bounded by d, the size of the SIPP tree is bounded by $2(D|\mathcal{V}|)^d$. □

Lemma 6 (Problem Size of Level 3)**.** *The LP model of Level 3 consists of 2 variables and at most $2(d + 1)$ constraints, where d is the number of conflict points in the path.*

We omit the proof for this lemma since it is trivial.

Theorem 2 (Time Complexity of PSL)**.** *The time complexity of PSL is $O(|\mathcal{V}|^{d+3} D^d \mathcal{F}(d))$,[2] where d is the largest number of conflict points in one path, and $\mathcal{F}(d)$ is the time complexity of the LP solver with d conflict points as input. Notably, this time complexity is polynomial in the number of vehicles.*

Proof. Since (1) PSL generates at most $O(|\mathcal{V}|^2)$ PT nodes, (2) each PT node replans paths for at most $|\mathcal{V}|$ vehicles, (3) each replan generates at most $2(D|\mathcal{V}|)^d$ SIPP nodes, and (4) each SIPP node calls the LP model at most once, the time complexity of PSL is $O(|\mathcal{V}|^2) \cdot |\mathcal{V}| \cdot 2(D|\mathcal{V}|)^d \cdot \mathcal{F}(d) \leq O(|\mathcal{V}|^{d+3} D^d \mathcal{F}(d))$. □

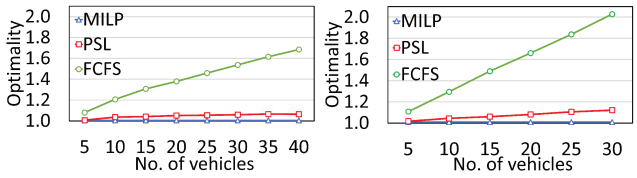
Empirical Evaluation

We implement both PSL and baseline algorithms (introduced below) in C++[3] and use C++ CPLEX to solve the programming models. We run experiments on a VMWare virtual machine with Intel I7-9850H 2.60Hz and 8GB RAM. CPLEX uses 12 cores, while the other codes use only 1 core.

Baseline Algorithms The survey paper (Zhong, Nejad, and Lee 2021) divides methods for solving the intersection coordination problems into three categories, namely system optimal, FCFS, and heuristics. We pick MILP proposed in (Levin and Rey 2017) as a representative system optimal method and FCFS introduced in Section 4.8 in (Levin and Rey 2017) as a representative FCFS method. We do not pick any heuristic methods because we are not aware of any heuristic methods that work for our problem setup.

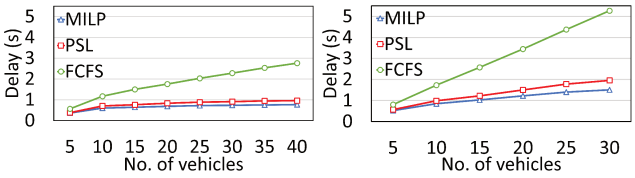
[2]This time complexity is in terms of the LP operations and ignores the other operations in the search such as collision checking, node insertion, etc. because they are usually cheaper than the LP operations and can be done in (near-)linear time.

[3]The code is available at https://github.com/theanhhoang/AIM.



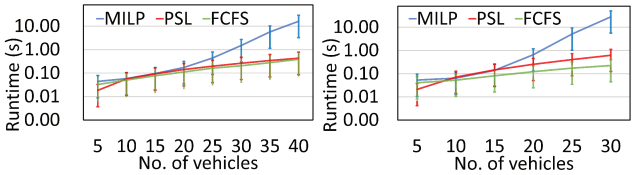
(a) 500vphpl (b) 800vphpl

Figure 4: Optimality with respect to total travel times.



(a) 500vphpl (b) 800vphpl

Figure 5: Average delay per vehicle.



(a) 500vphpl (b) 800vphpl

Figure 6: Runtime in log scales. The curves and error bars show the averages and the standard deviations, respectively.

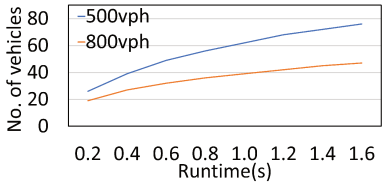


Figure 7: Scalibility of PSL.

Simulation Setup Figure 1 shows the intersection. The lane width λ is 3.66m (12ft), right-turn-radius μ_R is 1.83m (6ft), and left-turn-radius μ_L is 9.14m (30ft). The length of a vehicle is 5m. Its minimum and maximum speeds are $\underline{U}_i = 3$m/s and $\overline{U}_i = 15$m/s, respectively. We consider two demands, namely 500vphpl (i.e., 500 vehicles per hour per lane) and 800vphpl, which represent demands at normal and rush hours, respectively, and increasing numbers of vehicles (until MILP takes too long to find any solution). For instance, if the demand is 500vphpl and the number of vehicles is 40, then the distribution of the earliest entry times of the vehicles spread in $40/(8 \cdot 500/3600) = 36$s on average (since 500vphpl is equal to $8 \cdot 500/3600$ vehicles per second). We run 100 simulations for each setup. In each simulation, the entry lanes of the vehicles are generated uniformly at random. The possibility of a vehicle going straight is 80% and it turning left or right (according to the left or right lane) is 20%. We call each algorithm once to generate the paths for all vehicles in the beginning of each simulation.

Comparison in Optimality Our objective is the sum of the exit times, but its value relies on the earliest entry times of the vehicles. Therefore, we instead compare the total travel times, where the travel time of a vehicle i is the difference between its exit time $t_i(\gamma_i^+) + \tau_i(\gamma_i^+)$ and its earliest entry time e_i. We report the resulting optimality averaged over 100 simulations in Figure 4. As expected, the optimality of MILP is always 1. Although PSL is not optimal, its solution quality is no more than 10% worse than the optimal. In contrast, the solution of FCFS is poor; its solution quality is more than 100% worse than the optimal in the worst case.

Comparison in Delay We also evaluate the solution quality by looking at a popular metric used in transportation, namely the average delay per vehicle, where the delay of a vehicle is the difference between its exit time $t_i(\gamma_i^+)$ and its *earliest exit time* $e_i + d_i(\gamma_i^-, \gamma_i^+)/\overline{U}_i$, i.e., the exit time when it reaches its entry lane at the earliest entry time and travels through the intersection along the shortest path with its maximum speed. As Figure 5 shows, MILP always has the lowest average delay (because it is optimal), but the difference between the delay produced by MILP and the delay produced by PSL is minor. In the scenario of 500vphpl with 40 vehicles, PSL produces an average delay of 0.9s, which is only 0.2s larger than MILP. Similarly, in the scenario of 800vphpl with 30 vehicles, PSL produces an average delay of 2.0s, which is only 0.4s larger than MILP. Such small differences in delays are usually acceptable (or even can be neglected) in real traffic systems. However, the delay produced by FCFS is dramatically larger than that by MILP and PSL.

Comparison in Runtime Figure 6 compares the runtimes. MILP is more than 10 times slower than PSL and FCFS in many cases. The variation of the runtimes of MILP in both demand scenarios is also significant, indicating that MILP sometimes needs a dramatically larger runtime to find solutions. On the other hand, PSL and FCFS, even in the worst case, need only less than 1s to find solutions. These runtimes are well suited for real-time applications. To further demonstrate the scalability of PSL, we run PSL with more vehicles and report the results in Figure 7. Within 1s, PSL can solve problems with up to 62 vehicles in the 500vphpl scenario and 40 vehicles in the 800vphpl scenario. Within 1.6s, those numbers grow to 76 and 47 vehicles, respectively.

Conclusion

We proposed a MAPF-based algorithm PSL to coordinate autonomous vehicles at single no-signal intersections. PSL is complete, polynomial-time in the number of vehicles, and can coordinate dozens of vehicles in real-time. It runs substantially faster (more than 10 times faster in many cases) than the optimal method MILP without compromising the solution quality too much (no more than 10% worse than the optimal). It also finds significantly better solutions than the traditional FCFS strategy (e.g., FCFS can sometimes find solutions with quality more than 100% worse than optimal) while running similarly or slightly slower than FCFS.

Acknowledgments

The research at Monash University is part of a research project (Project No IH18.04.3) sponsored by the SPARC Hub (https://sparchub.org.au) at Department of Civil Eng, Monash University funded by the Australian Research Council (ARC) Industrial Transformation Research Hub (ITRH) Scheme (Project ID: IH180100010). The financial and in-kind support of Austroads and Monash University is gratefully acknowledged. Also, the financial support of ARC is highly acknowledged. Mr Ross Guppy from Austroads is profoundly thanked for his in-kind contributions to this project. The research at the University of Southern California is supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533 as well as a gift from Amazon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

Au, T.; Shahidi, N.; and Stone, P. 2011. Enforcing Liveness in Autonomous Traffic Management. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1317–1322.

Au, T.; and Stone, P. 2010. Motion Planning Algorithms for Autonomous Intersection Management. In *AAAI Workshop on Bridging the Gap Between Task and Motion Planning*.

Chen, J.; Li, J.; Fan, C.; and Williams, B. 2021. Scalable and Safe Multi-Agent Motion Planning with Nonlinear Dynamics and Bounded Disturbances. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 11237–11245.

Dresner, K.; and Stone, P. 2008. A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research*, 31: 591–656.

Dresner, K. M.; and Stone, P. 2004. Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 530–537.

Lee, J.; and Park, B. 2012. Development and Evaluation of A Cooperative Vehicle Intersection Control Algorithm under the Connected Vehicles Environment. *IEEE Transactions on Intelligent Transportation Systems*, 13(1): 81–90.

Levin, M. W.; Boyles, S. D.; and Patel, R. 2016. Paradoxes of Reservation-Based Intersection Controls in Traffic Networks. *Transportation Research Part A: Policy and Practice*, 90: 14–25.

Levin, M. W.; and Rey, D. 2017. Conflict-Point Formulation of Intersection Control for Autonomous Vehicles. *Transportation Research Part C: Emerging Technologies*, 85: 528–547.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021a. Pairwise Symmetry Reasoning for Multi-Agent Path Finding Search. *Artificial Intelligence*, 301: 103574.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: Bounded-Suboptimal Search for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 12353–12362.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021b. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 11272–11281.

Li, Z.; Chitturi, M. V.; Zheng, D.; Bill, A. R.; and Noyce, D. A. 2013. Modeling Reservation-Based Autonomous Intersection Control in VISSIM. *Transportation Research Record*, 2381(1): 81–90.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 7643–7650.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe Interval Path Planning for Dynamic Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 5628–5635.

Silver, D. 2005. Cooperative Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 117–122.

Skopkova, V.; Bartak, R.; and Svancara, J. 2020. What Does Multi-Agent Path-finding Tell Us About Intelligent Intersections. In *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, 250–257.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 151–159.

Yakovlev, K. S.; Andreychuk, A.; and Vorobyev, V. 2019. Prioritized Multi-agent Path Finding for Differential Drive Robots. In *Proceedings of the European Conference on Mobile Robots (ECMR)*, 1–6.

Zhong, Z.; Nejad, M. M.; and Lee, E. E. 2021. Autonomous and Semiautonomous Intersection Management: A Survey. *IEEE Intelligent Transportation Systems Magazine*, 13(2): 53–70.

Zohdy, I. H.; and Rakha, H. A. 2016. Intersection Management via Vehicle Connectivity: The Intersection Cooperative Adaptive Cruise Control System Concept. *Journal of Intelligent Transportation Systems*, 20(1): 17–32.