Multi-Agent Motion Planning with Bézier Curve Optimization under Kinodynamic Constraints

Jingtian Yan[1], Jiaoyang Li[1]

***Abstract*—Multi-Agent Motion Planning (MAMP) is a problem that seeks collision-free dynamically-feasible trajectories for multiple moving agents in a known environment while minimizing their travel time. MAMP is closely related to the well-studied Multi-Agent Path-Finding (MAPF) problem. Recently, MAPF methods have achieved great success in finding collision-free paths for a substantial number of agents. However, those methods often overlook the kinodynamic constraints of the agents, assuming instantaneous movement, which limits their practicality and realism. In this paper, we present a three-level MAPF-based planner called PSB to address the challenges posed by MAMP. PSB fully considers the kinodynamic capability of the agents and produces solutions with smooth speed profiles. Empirically, we evaluate PSB within the domains of traffic intersection coordination for autonomous vehicles and obstacle-rich grid map navigation for mobile robots. PSB shows up to 49.79% improvements in solution cost compared to existing methods while achieving significant improvement in scalability.**

***Index Terms*—Multi-agent Motion Planning, Traffic Intersection Coordination, Obstacle-rich Grid Map Navigation**

I. Introduction

MULTI-AGENT Motion Planning (MAMP) is a problem that focuses on finding collision-free dynamically-feasible trajectories for multiple agents in a known environment while minimizing their travel time. MAMP has received significant attention in recent years, becoming a core challenge in various real-world applications, including traffic management [1], warehouse automation [2], and robotics [3]. MAMP is closely related to a well-studied problem called Multi-agent Path Finding (MAPF) [4], which plans collision-free paths in discrete timesteps for multiple agents on a given graph. MAPF methods show the advantage of finding collision-free paths for hundreds of agents with optimal [5] or sub-optimal [6] guarantee. However, the standard MAPF model does not consider the kinodynamic constraints of the agents. It assumes instantaneous movement and infinite acceleration capabilities, leading to discrete solutions that prescribe agents to move in synchronized discrete time steps and are thus not directly executable on their controllers.

To tackle this challenge, some methods [7]–[9] discretize the action space and search using the motion primitives (i.e., elementary actions that an agent can perform). Those methods usually search in high-dimensional state space to account for the kinodynamics of agents such as speed and acceleration. However, due to their discretized nature, their solutions do not fully capture the kinodynamic capacity of the agents, leading to limited choices of actions and, thus, poor performance in challenging scenarios. To address this issue, a three-level method PBS-SIPP-LP (PSL) [1] is introduced for the traffic intersection coordination problem, a special case of MAMP. PSL integrates search-based and optimization methods. Level 1 and Level 2 employ an extended MAPF planner to identify potential paths, which invokes Level 3 to optimize vehicle travel speeds through a Linear Programming (LP) model. However, PSL only considers speed constraints for agents, making a strong assumption that each agent moves through the intersection at a constant speed.

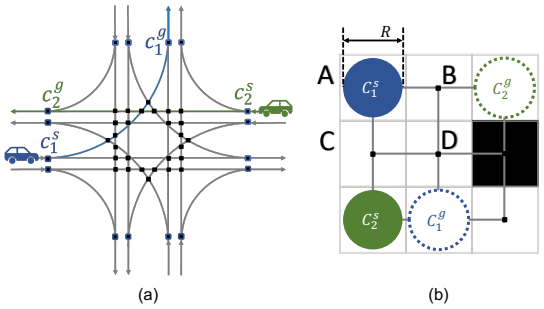

Fig. 1: Illustration of the intersection model and the grid model we use. (a) Traffic intersection coordination model. (b) Grid map navigation model. Collision points are marked as black dots. The black cells in (b) are static obstacles.

Taking inspiration from PSL, we introduce a three-level planner called PSB (**P**BS-**S**IPP-**B**ézier) to address the MAMP problem. Level 1 uses Priority-Based Search (PBS) [10] to resolve the collisions between agents. Level 2 employs an extended Safe Interval Path Planning (SIPP) [11] to search candidate paths for each agent given constraints from Level 1. Level 3 employs a Bézier-curve-based [12] planner to seek optimal speed profiles for agents based on the constraints from Level 2 and their kinodynamic constraints. To reduce the runtime, PSB utilizes a cache structure to reuse solutions from Level 3. It also incorporates a duplicate-detection mechanism and a time-window mechanism to reduce the runtime further when extending PSB to the grid map navigation scenario.

The following content outlines our work in this paper: (1) We propose PSB, a MAMP planner capable of finding feasible, high-quality trajectories for multiple agents by exploiting their

Manuscript received: Oct, 12, 2023; Revised Dec, 24, 2023; Accepted Jan, 22, 2024.

This paper was recommended for publication by Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers' comments. The research is supported by the National Science Foundation under grant number 2328671 and the CMU Manufacturing Futures Institute, made possible by the Richard King Mellon Foundation. *(Corresponding author: Jingtian Yan.)*

[1]J. Yan, and J. Li are with Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA (jingtianyan, jiaoyangli@cmu.edu).

Digital Object Identifier (DOI): see top of this page.

complete kinodynamic capability. PSB can produce speed profiles with any degree of smoothness. To our knowledge, this is the first MAPF-based work in MAMP that incorporates the full kinodynamic capability for the extensive group of agents. (2) We show that PSB can efficiently handle the intersection coordination problem for autonomous vehicles with kinodynamic constraints (shown in Fig. 1(a)). Compared to the previously best algorithm PSL [1], PSB shows up to 49.79% improvement in terms of solution cost. (3) We extend PSB to a more general domain: mobile robots navigation in obstacle-rich grid maps (shown in Fig. 1(b)). PSB outperforms the previously best algorithm SIPP-IP [7], achieving up to 27.12% improvement in solution cost and increased scalability to handle up to 220 agents, as opposed to the limit of 80 agents in SIPP-IP, while maintaining a comparable runtime.

II. Related Work

Multiple algorithms have been developed to tackle MAMP. Some methods extend single-agent motion planners for MAMP. In [13], a multi-agent RRT* derived from single-agent RRT* [14] is proposed, combining the state space of individual agents into a collective joint space for RRT* planning. However, planning within the joint state space of agents presents scalability challenges, as the dimension of the joint state space increases exponentially in the number of agents. Another category of methods uses control or optimization techniques to handle MAMP in real-world robotics. For instance, in [15], a vector field approach combined with model predictive control is used to steer agents in complex environments. In [3], a mixed-integer linear program is utilized to search for timed waypoints that fulfill signal temporal logic constraints, which encapsulate the kinodynamic constraints. These methods are effective in scenarios with few agents. However, they face scalability issues as the number of agents increases. For instance, the approach by [3] takes over 100 seconds to generate the solution for only a four-agent scenario.

Recently, MAPF methods have achieved significant progress in finding discrete collision-free paths for a large number of agents. Leading methods, such as Conflict-Based Search (CBS) [5], [16] and Prioritized-Based Search (PBS) [10], use single-agent solvers to plan paths for individual agents and address collisions among agents by introducing constraints to single-agent solvers. Some methods take advantage of the MAPF methods to solve the MAMP problem.

One category of such methods uses discrete paths from MAPF planners and smooths them to meet kinodynamic constraints. For instance, the method in [17] enforces agents to adhere to their designated discrete paths generated by the MAPF planner. Then, it considers the speed constraints and employs an LP solver to generate an executable plan. Similarly, in [18], agent trajectories, represented as Bézier curves, are optimized by accounting for high-order kinodynamic constraints using the paths from MAPF planners. These methods highly rely on the discrete paths from MAPF planners, and their open-loop nature can lead to failures if kinodynamically feasible trajectories cannot be derived from these paths.

Another category of methods extends MAPF methods to consider robot kinodynamics. For instance, CBS-MP [8] integrates probabilistic roadmaps with CBS to handle collisions and employs motion-primitive-based search for single-agent path planning. Cohen et al. [9] adapt CBS to continuous time and develop a bounded-suboptimal extension of SIPP for pathfinding for individual agents, where SIPP [11] is a variant of A^* that can find optimal paths in dynamic environments. Ali and Yakovlev [7] extend SIPP to account for the kinodynamics of agents by incorporating a wait interval projection mechanism, addressing the impractical assumption in SIPP of instantaneous stopping. However, since these methods are all graph-search-based, they discretize the action space for search. As a result, these methods consider a limited number of actions and thus fail to capture the full range of possible actions that agents could exhibit.

Recently, a three-level method called PSL [1] is introduced, combining MAPF with optimization methods. It uses PBS and SIPP to search for candidate paths and integrates an LP solver to optimize agent speeds along given paths. However, PSL makes a strong assumption that each agent moves through the intersection at a constant speed, which still fails to capture the full range of possible actions that agents could exhibit.

III. Problem Formulation

We define our MAMP problem by a graph $G = (V, E)$ and a set of M agents $\mathcal{A} = \{a_1, ..., a_M\}$. Vertices in V, referred to as *collision points*, represent potential collision locations for agents. Agents can move from collision points $c_i \in V$ to $c_j \in V$ along edge $(c_i, c_j) \in E$ while adhering to kinodynamic constraints as shown in Eq. 1, with the travel distance denoted as $d(c_i, c_j) \in \mathbb{R}^+$. Each agent a_i initiates its movement from a specified *start (collision point)* $c_i^s \in V$ with initial kinodynamic constraints as shown in Eq. 2 and moves toward a designated *goal (collision point)* $c_i^g \in V$.

Definition 1. *(Path). The path of agent a_i, if contains $m+1$ collision points, as $\phi_i = \{c_i^0 = c_i^s, c_i^1, ..., c_i^{m-1}, c_i^m = c_i^g\}$ where $(c_i^{j-1}, c_i^j) \in E, j = 1, \cdots, m$.*

Definition 2. *(Spatio-temporal profile). The spatio-temporal profile, denoted as $\ell_i(t) : \mathbb{R}^+ \to \mathbb{R}^+$, quantifies the distance traversed by a_i as a function of time t along a given path.*

Definition 3. *(Trajectory). The trajectory of a_i is the combination of a path and its associated spatio-temporal profile.*

The kinodynamic constraints limit the gradient of spatio-temporal profile with respect to time up to K-th order:

$$\underline{U_i^k} \leq d^k \ell_i(t)/dt^k \leq \overline{U_i^k}, \forall k \in 1, ..., K \tag{1}$$

$$d^k \ell_i(t)/dt^k|_{t=0} = \underline{U_i^k}, \forall k \in 1, ..., K \tag{2}$$

where $\underline{U_i^k}$ and $\overline{U_i^k}$ are constant values that define the constraints on the k-th order gradient. We use *arrival time* T_i to indicate the time needed for a_i to arrive at c_i^g. We denote $t_i(c)$ as the time when a_i reaches the collision point c, and $\tau_i(c)$ represents the duration that it occupies c. Therefore, a_i departs from c at time $t_i(c) + \tau_i(c)$. In the event that both a_i and a_j pass through the same collision point c, a *collision* occurs iff the time intervals $[t_i(c), t_i(c)+\tau_i(c))$ and $[t_j(c), t_j(c)+\tau_j(c))$ overlap.

1) Intersection Model: For the intersection coordination problem, we adopt the model from [19]. As shown in Fig. 1 (a), the intersection has a set of entry lanes Γ^- and a set of exit lanes Γ^+. Each agent $a_i \in \mathcal{A}$, with a length of L_i, has a traveling request from the entry lane $c_i^s \in \Gamma^-$ to the exit lane $c_i^g \in \Gamma^+$ with the *earliest start time* $e_i \in \mathbb{R}^+$ (i.e., the earliest time a_i can reach at the entry lane). To prevent agents from making sharp turns, they are constrained to follow the predefined path (gray lines in Fig. 1 (a)). We assume the agents can wait before the entry lane without any collision. After entering the intersection, the minimum speed of agents is strictly positive. If both a_i and a_j start from the same entry lane with $e_i < e_j$, then a *overtake* happens if $\exists c \in V$ that $t_i(c) \geq t_j(c)$. Our task is to generate trajectories for all agents so that no collisions or overtakes happen while minimizing the sum of their arrival time.

2) Grid Model: We adopt the grid model from classical MAPF problems [4] and represent G as a four-neighbor grid map. Fig. 1 (b) shows an example. There are three differences compared to the intersection model: (D1) In contrast to the intersection model where there is only one path for each agent to move from its start to goal, the grid model involves an extra spatial domain search where we need to plan the path for each agent. (D2) Agents are allowed to stop at any location, which imposes a minimum speed constraint of $\underline{U_i^1} = 0$. (D3) All agents start simultaneously and remain at their respective goals after they finish. It should be noted that this differs from the intersection model, where agents are only present in G while in the intersection and not before entry or after departure. Our primary objective is to plan collision-free trajectories for all agents while minimizing the sum of their arrival time.

IV. MAMP on Intersection

This section begins with a system overview of our proposed algorithm PSB, followed by a discussion of the specifics of our trajectory optimization formulation and method. After that, we delve into the techniques used to tackle runtime challenges.

A. System Overview

PSB consists of a three-level planner as illustrated in Fig. 2, with Level 1 and Level 2 extended from PSL [1]. Level 1 uses PBS [10] to resolve collisions among agents through priority ordering searching, where the trajectory of each agent is planned by Level 2. Given the priority orderings from Level 1, Level 2 uses an extended SIPP [11] to search for the optimal trajectory for an agent, where the spatio-temporal profile of the trajectory is optimized by Level 3. Given the path (together with some temporal constraints) from Level 2, Level 3 uses BCP (**B**ézier-**C**urve-based **P**lanner) to generate the optimal spatio-temporal profile.

1) PBS-based planner: We adopt PBS [10] to resolve collisions among agents. If two agents have a priority ordering, the agent with lower priority must avoid collisions with the agent with higher priority. The goal is to find a set of priority orderings for agents so that they do not collide with each other. To achieve this, we search a binary *Priority Tree* (PT) in a depth-first manner. Each PT node contains a set of priority

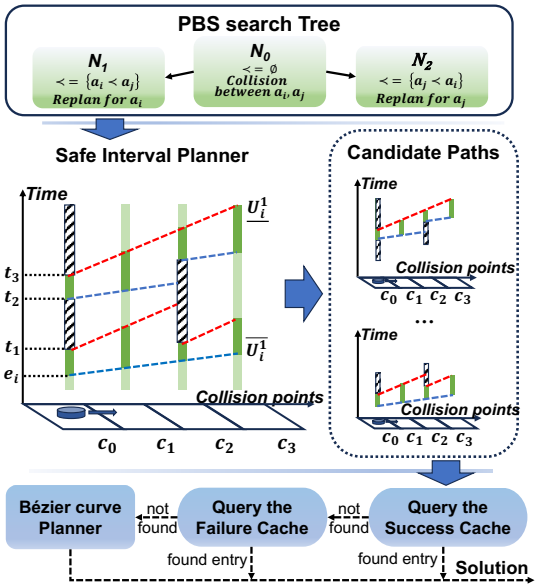

Fig. 2: Overview of PSB. The shadowed strips denote time intervals occupied by other agents, the green segments denote safe intervals. The dark green segments represent the intervals in the open list.

orderings and a set of trajectories consistent with the priority orderings. We initialize the root PT node by adding priority orderings for the agents starting from the same entry lane based on their earliest start time to prevent *overtake*. Then, Level 2 is called to plan trajectories for agents at each lane from the highest priority to the lowest priority. We expand PT nodes by searching for collisions between agents and generating child PT nodes with additional priority orderings to address the collisions. For instance, as shown in Fig. 2, if a collision between agents a_i and a_j is detected in node N_0, we resolve it by expanding N_0 into two child nodes N_1 and N_2. In N_1, a_i is given higher priority than a_j, while in N_2, the priority is reversed. Then, in each child node, we use Level 2 to replan trajectories for each agent based on the priority orderings in that node. If we find a PT node with collision-free trajectories, we terminate the search and return the trajectories.

2) Safe Interval Planner: Level 2 aims to find a trajectory for a given agent a_i that minimizes its arrival time while avoiding collisions with higher-priority agents. To achieve this, we run a modified SIPP on a safe interval graph, which associates each collision point with a set of safe intervals. A *safe interval* $[lb, ub)$ is a time duration that an agent can stay at a collision point without colliding with higher-priority agents. The modified SIPP plans a path (along with safe intervals) and calls Level 3 to specify the spatio-temporal profile considering kinodynamic constraints. We provide an example of this process in Fig. 2, with a detailed description available in [1]. While we use a 1-D example for demonstration, as later shown in the grid model, this search process is applicable in general graphs. Since there are two safe intervals $[e_i, t_1)$ and $[t_2, t_3)$ at the start c_0, we generate two SIPP nodes and insert them into the open list. In each iteration, we expand the node from the open list with the smallest f-value (= the lower bound of its safe interval plus the minimum arrival time required to reach

the goal from its collision point). In this example, we choose the node with safe interval $[e_i, t_1)$. During node expansion, to speed up the search process while guaranteeing completeness, we use relaxed kinodynamic constraints. Specifically, we assume that a_i occupies each collision point for only an instant of time while allowing adjustment of its speed within the range $[\underline{U_i^1}, \overline{U_i^1}]$ instantly. In our case, as a_i moves from c_i^0 to c_i^1, a new interval $[e_i + t_{min}, t_1 + t_{max})$ is generated at c_i^1, where $t_{min} = d(c_0, c_1)/\overline{U_i^1}$ and $t_{max} = d(c_0, c_1)/\underline{U_i^1}$ are the minimum and maximum time for this movement. We iterate the safe intervals at c_1 that overlap with this new interval and insert them into the open list. When a node reaches the goal, we backtrack to retrieve the full path and its associated safe intervals. Then, we call Level 3 to determine the kinodynamically feasible trajectory. If Level 3 finds a trajectory with a smaller arrival time than the best trajectory found so far, we update the current best trajectory. We terminate the search if no node in the open list has a smaller f-value than the arrival time of the current best trajectory. As shown in [1], Level 2 guarantees to return the optimal trajectory when Level 3 is optimal and complete.

3) Bézier-curve-based Planner: Level 3 aims to generate a kinodynamically feasible spatio-temporal profile $\ell_i(t)$ for a given path within given safe intervals while minimizing the arrival time. We use Bézier curve $B^{T_i}(t)$, where T_i is the *arrival time*, to represent $\ell_i(t)$ and reformulate this problem by finding the minimum T_i and control points P_i for $B^{T_i}(t)$ that satisfy the kinodynamic and safe-interval constraints. As shown in Sec. IV-C, Level 3 is complete and optimal.

B. Background on Bézier Curves

A Bézier curve [12] is a function parameterized by a set of control points. With a sufficiently large number of control points, it is able to approximate any continuous function $f(t)$ with $t \in [0, 1]$, making it an ideal choice for modeling $\ell(t)$ [18]. We use $B^T(t)$ to denote the Bézier curve that scales the interval for t from $[0, 1]$ to $[0, T]$:

$$B^T(t) = \sum_{r=0}^{n} p_r B_{r,n}^T(t),\ t \in [0, T], \tag{3}$$

where $B_{r,n}^T(t) = \binom{n}{r}(\frac{t}{T})^r(\frac{T-t}{T})^{n-r}$ is called a Bernstein basis polynomial, and $P = \{p_0, ..., p_n\}$ are $n+1$ control points.

As proven in [12], the Bézier curve has three properties that can help find kinodynamically feasible trajectories efficiently: (P1) $B^T(t)$ is bounded by the convex hull of its control points P for $t \in [0, T]$. (P2) The curve always starts at the first control point ($B^T(0) = p_0$) and ends at the last control point ($B^T(T) = p_n$). (P3) The derivative of the n-degree Bézier curve is another Bézier curve with degree $n-1$:

$$\frac{d^k B^T(t)}{dt^k} = \sum_{r=0}^{n-k} p_r^k B_{r,n-k}^T(t),\ t \in [0, T] \tag{4}$$

where $p_r^k = \frac{n!}{(n-k)!T^k}\sum_{j=0}^{k}(-1)^j\binom{k}{j}p_{n-j}$ is the r-th control point for the k-th gradient of $B^T(t)$.

By properties (P1) and (P3), we can map the kinodynamic constraints discussed in Eq. 1 to constraints on control points:

$$\underline{U_i^k} \le p_r^k \le \overline{U_i^k},\ \forall r \in \{0, 1..., n-k\}. \tag{5}$$

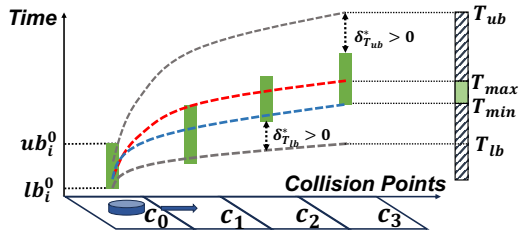

Fig. 3: Illustration of the search for optimal arrival time. The agent moves along the path from c_0 to c_3 with associated safe intervals represented by green segments. The optimal spatio-temporal profiles found by Eq. 6-11 for the four different arrival time T_x shown on the right are represented by four dashed curves. Since the two gray ones do not intersect with all green segments, they are infeasible, and $\delta^*_{T_x}$ indicates how far away they are from feasible profiles. As the figure shows, minimizing $\delta^*_{T_x}$ can bring these profiles closer to feasible ones.

C. Bézier-curve-based Planner (BCP)

Given a path $\phi_i = \{c_i^0, \cdots, c_i^m\}$ from Level 2, along with its associated safe intervals $S_i = \{[lb_i^0, ub_i^0), \cdots, [lb_i^m, ub_i^m)\}$, BCP aims to find a spatio-temporal profile $l_i(t)$, which we approximate by $B^{T_i}(t)$, for path ϕ_i within the safe intervals in S_i that satisfies the kinodynamic constraints defined in Eq. 1-2 and minimizes the arrival time T_i. Inspired by [18], we introduce a method to determine $B^{T_i}(t)$ for a given arrival time T_i and then a method to find the optimal arrival time T_i^*. In the rest of this section, we omit subscript i for simplicity.

1) Find Bézier curve given arrival time: Given arrival time T, our task is to determine the control points P for $B^T(t)$ that meet all constraints, which we formulate as an LP problem:

$$\min_{P, \delta_T} \delta_T \tag{6}$$

$$s.t.\ \delta_T \ge 0 \tag{7}$$

$$p_0 = d^0, \quad p_n = d^m, \quad p_0^k = \underline{U^k},\ \forall k \in \{1, ..., K\} \tag{8}$$

$$B^T(lb^j) \le d^j + \delta_T,\ \forall j \in \{1, ..., m\} \tag{9}$$

$$B^T(ub^j - L/\omega) > d^j + L - \delta_T, \forall j \in \{1, ..., m\} \tag{10}$$

$$\underline{U^k} - \delta_T \le p_r^k \le \overline{U^k} + \delta_T, \quad \forall k \in \{1, ..., K\}, \quad \forall r \in \{0, 1, ..., n-k\} \tag{11}$$

where $d^j = \sum_{r=1}^{r=j} d(c^{r-1}, c^r)$ is the travel distance from start to collision point c^j. Eq. 6-7 minimize δ_T, a non-negative slack variable used to relax the safe-interval constraints and the kinodynamic constraints. $B^T(t)$ exists iff the optimal value for δ_T is zero. Eq. 8 requires the agent to initiate from its entry lane with its initial kinodynamic constraints and terminate at its exit lane. Given the challenge of finding a closed-form inverse function for $d^j = B^T(t)$, Eq. 9-10 ensure the agent visits each collision point within the safe interval by preventing arrivals before lb^j (travel distance at $t = lb^j$ must be no larger than d^j) or departures after ub^j (travel distance at $t = ub^j$ must be larger than $d^j + L$). Here, ω is a constant derived from wave speed [20] (we set it to 7 in our experiment). Eq. 11 enforces the kinodynamic constraints from Eq. 5.

2) Search for optimal arrival time: To find the minimum arrival time T^*, BCP employs a binary search. The LP model defined by Eq. 6-11 can be viewed as a function that maps

Algorithm 1 Bézier-curve-based Planner (BCP)

Input: Path ϕ and associated safe intervals S
Output: The ϵ-optimal T^* and Bézier curve $B^{T^*}(t)$
Function *IsSolutionExist*(T_{lb}, T_{ub})
  return $\nabla\delta^*_{T_{lb}} \leq 0$ *and* $\nabla\delta^*_{T_{ub}} \geq 0$
1: $[T_{lb}, T_{ub}) \leftarrow [lb^m, ub^m)$
2: **if** $ub^m = \infty$ **then** $T_{ub} \leftarrow$ objective value of Eq. 12
3: **if** !*IsSolutionExist*(T_{lb}, T_{ub}) **then** **return** *'no solution'*
4: **while** $T_{ub} - T_{lb} > \epsilon$ **do**
5: $T_{mid} \leftarrow (T_{lb} + T_{ub})/2$
6: **if** $\delta^*_{T_{mid}} = 0$ **then** $T_{ub} \leftarrow T_{mid}$, $T^* \leftarrow T_{mid}$
7: **else**
8: **if** $\nabla\delta^*_{T_{mid}} > 0$ **then** $T_{ub} \leftarrow T_{mid}$
9: **else** $T_{lb} \leftarrow T_{mid}$
10: **if** $T^* = null$ **then** **return** *'no solution'*
11: **else** **return** T^* and $B^{T^*}(t)$

an arrival time T to the optimal slack value δ^*_T. We denote the gradient of this function at T by $\nabla\delta^*_T$, which, empirically, we approximate by $(\delta^*_{T+\Delta} - \delta^*_T)/\Delta$ with Δ being a sufficient small positive number. We borrow the following lemma from Theorem 2 of [18].

Lemma 1. *$B^T(t)$ exists iff a single time range $[T_{min}, T_{max}]$ exists such that $\delta^*_T = 0$ for $T \in [T_{min}, T_{max}]$ and $\delta^*_T > 0$ for $T \notin [T_{min}, T_{max}]$, making T_{min} the optimal travel time T^*. Moreover, $\nabla\delta^*_T < 0$ for $T \in [0, T_{min})$, $\nabla\delta^*_T = 0$ for $T \in [T_{min}, T_{max}]$, and $\nabla\delta^*_T > 0$ for $T \in (T_{max}, \infty)$.*

Thus, in an iteration of our binary search with $T^* \in [T_{lb}, T_{ub}]$, we can decide the search direction by evaluating the gradient of δ^*_T at the midpoint $(T_{lb} + T_{ub})/2$. The initial range $[T_{lb}, T_{ub}]$ for T^* is set to the safe interval at the exit lane $[lb^m, ub^m)$. However, due to the possibility of the agent waiting indefinitely at its entry lane, ub^m can be infinite. In that case, we compute T_{ub} by the following LP model:

$$\begin{aligned} \min_{t} \quad & t + (d^m + L)/\underline{U} \\ s.t. \quad & lb_j \leq t + d^j/\underline{U}, \quad \forall j \in \{1, ..., m\} \end{aligned} \tag{12}$$

Lemma 2. *If the latest arrival time at the exit lane is unbounded ($ub^m = \infty$), then there exists a valid $B^T(t)$ with T being the objective value of Eq. 12.*

Proof. Since the minimum speed of agents is strictly positive, any safe interval with a finite upper bound restricts subsequent safe intervals along the path to also have finite upper bounds. Therefore, $ub^m = \infty$ implies $ub^j = \infty$ for $j = 0, \cdots, m$. As a result, the agent can wait at its entry lane for sufficient time and then maintain a continuous movement at $\underline{U}$ throughout the intersection. Eq. 12 computes the minimum arrival time for such a solution. □

3) Pseudocode for BCP: As shown in Algorithm 1, we initialize the range for T^* by the last safe interval $[lb^m, ub^m)$ [Line 1]. In case of $ub^m = \infty$, the upper bound T_{ub} is computed by Eq. 12 [Line 2]. Then, we determine if $\exists T \in [T_{lb}, T_{ub})$ such that $B^T(t)$ exists. According to Lemma 1, $B^T(t)$ exists when $[T_{lb}, T_{ub})$ overlaps with $[T_{min}, T_{max}]$, which is true if $\nabla\delta^*_{T_{lb}} \leq 0$ and $\nabla\delta^*_{T_{ub}} \geq 0$ (as shown in function *IsSolutionExist*). If $B^T(t)$ does not exist, BCP returns *'no solution'* [Line 3]. Otherwise, we proceed with the binary search to find T^* within range $[T_{lb}, T_{ub})$, and this search continues until the range is smaller than a predefined threshold ϵ (we use $\epsilon = 0.1$ in our experiments) [Line 4]. In each iteration, we average T_{lb} and T_{ub} to get T_{mid} and solve Eq. 6-11 at T_{mid} [Line 5]. If $\delta^*_{T_{mid}} = 0$, it indicates that $T_{mid} \in [T_{min}, T_{max}]$. Consequently, we use T_{mid} as the new upper bound for T^* [Line 6]. On the other hand, if $\delta^*_{T_{mid}} \neq 0$, we calculate its gradient. If $\nabla\delta^*_{T_{mid}} > 0$, it implies that $T_{mid} > T_{max}$, and we thus use T_{mid} as the new T_{ub} [Line 8]. Conversely, it can be concluded that $T_{mid} < T_{min}$, and we thus use T_{mid} as the new T_{lb} [Line 9].

Theorem 1 (Completeness and optimality of BCP)**.** *Given a small enough ϵ and a large enough number of control points, BCP finds the spatio-temporal profile $B^{T^*}(t)$ with minimum arrival time T^* if one exists and returns failure otherwise.*

Proof. With sufficient control points, the Bézier curve $B^T(t)$ is able to approximate any continuous function in $[0, T]$. Thus, Eq. 6-11 always find P^* given T^*, if a feasible spatio-temporal profile exists. At the same time, with Lemma 2, the upper bound for T^* is always finite. So after a finite number of iterations, as shown in Theorem 3 of [18], the binary search in BCP can find the ϵ-optimal T^*. □

Theorem 2 (Completeness and suboptimality of PSL)**.** *PSB guarantees to find a (sub-optimal) solution in finite time.*

Proof. PSL [1] guarantees to find a (sub-optimal) solution in finite time. The primary distinction between our PSB and PSL lies in Level 3. As Theorem 1 demonstrates that Level 3 of PSB is both complete and optimal, we can reuse the proof for PSL without modifications to show that our PSB also guarantees to find a (sub-optimal) solution in finite time. □

D. Caching BCP Results

As BCP encounters recurrent LP solving, calling BCP frequently can lead to a significant increase in runtime. To address this issue, we implement two types of caches, namely *SuccessCache* and *FailureCache*, for each agent to utilize the results of previous BCP calls to expedite the search process.

While, in the intersection model, there exists only one path that moves each agent from its start to goal, we consider the general case where each agent can have multiple paths so that the caching mechanisms can be later applied to the grid model with no changes. We say two paths for the same agent are *pseudo-identical* iff they contain the same number of collision points and the travel distances between corresponding collision points are the same. Since the LP model in Eq. 6-11 uses the distances between collision points instead of the specific location of each collision point, we can relax the requirements for reusing results from both caches from the path being identical to pseudo-identical.

SuccessCache maps paths with associated safe intervals to optimal spatio-temporal profiles. During the search process, if a path is pseudo-identical to the path of an entry from

SuccessCache, and their associated safe intervals are identical, then we can reuse the spatio-temporal profile from that entry.

In contrast, *FailureCache* stores paths with associated safe intervals for which no spatio-temporal profiles exist. Notably, if no spatio-temporal profiles exist for a path with associated safe intervals, reducing the ranges of some safe intervals will still result in the non-existence of spatio-temporal profiles. Thus, during the search process, if a path is pseudo-identical to the path of an entry from *FailureCache*, and each of its safe intervals is either identical to or a subset of the corresponding safe interval from the entry, we can infer that no spatio-temporal profiles exist.

As shown in Fig. 2, during the search process of Level 2, after obtaining a path ϕ_i with associated safe intervals S_i through backtracking, we first check *SuccessCache*. If an entry is found in the cache, we return the stored spatio-temporal profile as the optimal profile for the (ϕ_i, S_i) pair. If no entry is found in *SuccessCache*, we then check *FailureCache*. If an entry is found in *FailureCache*, we declare 'no solution' for the (ϕ_i, S_i) pair. Otherwise, we call BCP to find spatio-temporal profiles. If BCP returns a valid profile, then the (ϕ_i, S_i) pair along with the obtained profile is stored in *SuccessCache*. Otherwise, the (ϕ_i, S_i) pair is inserted into *FailureCache*.

V. MAMP on Grid Model

In this section, we extend PSB to address MAMP on the grid model. Given the three differences between the two models outlined in Sec. III-2, we make the following changes to PSB. Level 1 is applied to the grid model with no changes. However, the root PT node, in this case, contains no priority orderings because, as per Difference (D3), no two agents share the same start. While Level 2 can also be applied to the grid model with minor changes to handle Difference (D3), direct adoption results in poor scalability, as Difference (D1) causes SIPP to expand many duplicate SIPP nodes that reach the same collision point through different paths. To overcome this, we introduce a duplicate detection mechanism to prevent the expansion of duplicate nodes and a time window mechanism to expedite the planning process. For Level 3, given that Lemma 2 relies on the assumption of positive minimum speeds, which is no longer true due to Difference (D2), we incorporate a new method to initialize the upper bound. Additionally, due to Difference (D3), it is important to note that PSB is incomplete for the grid model.

1) Upper bound estimation for optimal arrival time: In cases where ub^m is infinite, we employ an exponential increment approach to establish the initial value for T_{ub}. We begin by setting T_{ub} to T_{lb} and then evaluate the gradient of $\delta^*_{T_{ub}}$. If the gradient is non-negative, then, as per Lemma 1, $T^* \in [T_{lb}, T_{ub}]$, in which case we find a valid upper bound and thus conclude the search process. Otherwise, we double T_{ub} and evaluate its gradient again. However, if T_{ub} reaches infinity (represented by the large value of 4000 in our experiment), we declare 'no solution' for the corresponding SIPP node.

2) Duplicate detection: When extending Level 2 to the grid model, we may generate "duplicate" nodes that reach the same collision point through different paths. Since the input of Level 3 contains the entire path and associated safe intervals from the root node to the current goal node, we cannot naively prune such "duplicate" nodes. Instead, we always add generated nodes to the open list even if nodes with identical collision points have been previously added. However, this method causes an exponential rise in the number of expanded nodes. Therefore, we develop a cleverer duplicate-detection method. Recall our reasoning on pseudo-identical paths for caching: if two goal nodes have pseudo-identical paths with exactly the same safe intervals, then their results from Level 3 should be identical. For example, in Fig. 1 (b), if we consider the movement of agent 1, expanding the SIPP node at A generates two nodes at B and C. Expanding these nodes further results in two "duplicate" nodes at D. While the two nodes correspond to two different paths from A to D, if their safe intervals are identical, they lead to the same LP models in Level 3. Thus, we can retain only one of the nodes without losing completeness. Specifically, when we generate a node at collision point c, we prune it if we have previously generated a node that is also at c and corresponds to a pseudo-identical path with exactly the same safe intervals.

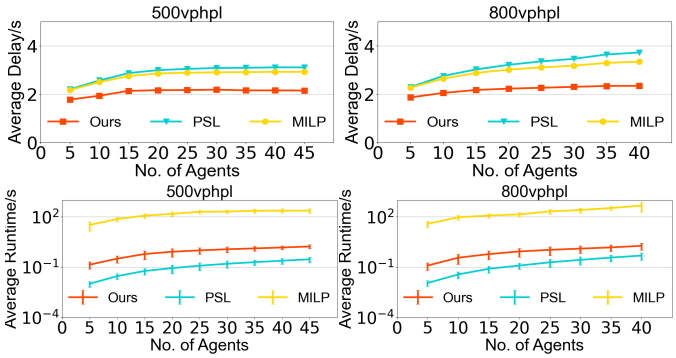

Fig. 4: Delay and average runtime (in log scale) for the intersection model. Error bars show the standard deviations.

3) Windowed PSB: While the duplicate detection mechanism helps mitigate some duplicates, the SIPP search in Level 2 can still be time-consuming. We thus employ a rolling-horizon strategy [21], [22]. In each iteration, we only focus on the collisions that occur within the current time window of size t_W. Once PSB finds collision-free trajectories within the time window, we shift the window by a predetermined replan window $t_s < t_W$. (In our experiment, we set $t_s = 4, t_W = 6$.) By recurrently replanning, we progressively advance the time window and compute the full trajectories for all agents.

VI. Empirical Evaluation

Both our PSB and baseline methods are implemented in C++ and utilize CPLEX to solve the programming models.[1] All experiments are conducted using a single core on an Ubuntu 20.04 machine equipped with an AMD 3990x processor and 188 GB of memory.

[1]The source code for our method and baselines is publicly available at https://github.com/JingtianYan/PSB-RAL.

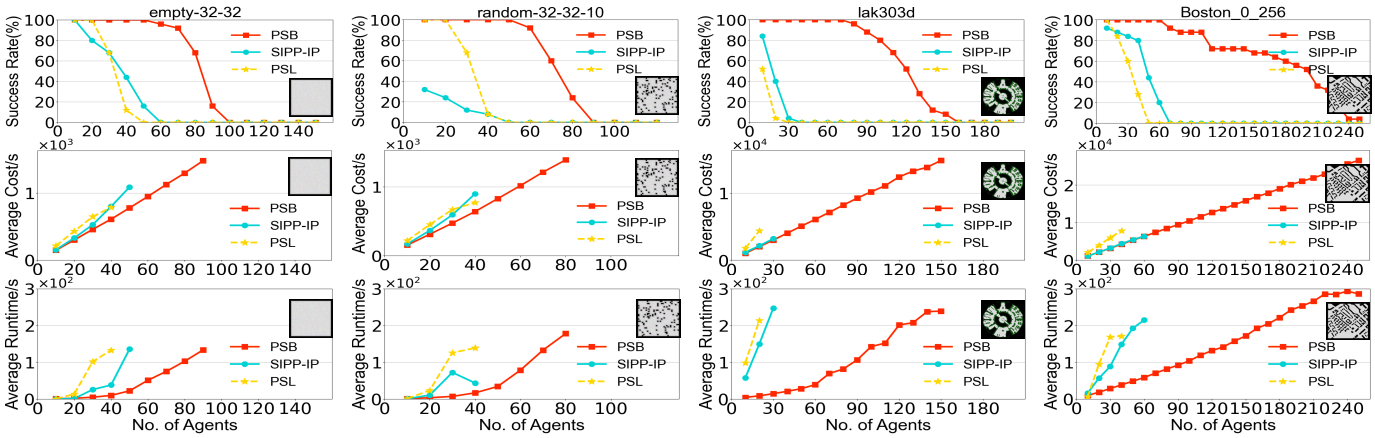

Fig. 5: Success rate, solution cost, and runtime of PSB, PSL, and SIPP-IP across all maps for the grid model. The solution cost and runtime are averaged only over scenarios where the planner successfully generates a solution. Note that PSL uses a relaxed agent model.

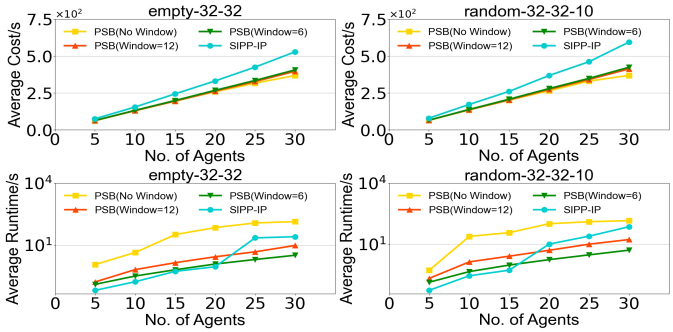

Fig. 6: Runtime and cost of PSB with various window sizes.

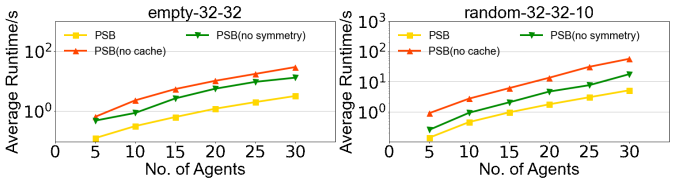

Fig. 7: Runtime of PSB for ablation study.

A. MAMP on Intersection

1) Baseline Methods: We include two baseline methods. The first baseline is PSL [1], the previous algorithm that our PSB builds on. The second baseline is MILP [19], which solves the problem with mixed-integer linear programming. It is important to note that both methods assume that agents travel across the intersection at constant speeds.

2) Simulation Setup: We utilize a simulation setup identical to the one used by both baseline methods [1], [19]. Graph G is depicted in Fig. 1. The lane width is $3.66\ m$, the right-turn radius is $1.83\ m$, and the left-turn radius is $9.14\ m$. The length of each agent is $L_i = 5\ m$. The entry lane for each agent is generated uniformly at random. An agent at an entry lane has an 80% probability of going straight and a 20% probability of turning left or right. For simplicity, we only consider the speed and acceleration constraints. We set a minimum speed of $\underline{U_i^1} = 3\ m/s$ for all agents. The maximum speed for agents turning left is $\overline{U_i^1} = 5\ m/s$, while that for other agents is $\overline{U_i^1} = 15\ m/s$. As for acceleration, for all agents, we set $\underline{U_i^2} = -2\ m/s^2$ and $\overline{U_i^2} = 5\ m/s^2$. We randomly sample travel requests from two demands, namely 500 vphpl (i.e., 500 vehicles per hour per lane) and 800 vphpl, and report results averaged over 25 instances from each demand and each number of agents.

3) Comparison: While our objective is to minimize the sum of the arrival time for all agents, their arrival time depends on their earliest start time, which varies from instance to instance. Thus, we instead report the *average delay*, a popular metric used in transportation that measures the average difference between the arrival time and the *earliest arrival time* (i.e., the earliest time an agent can reach its goal, defined by $e_i + d(c_i^s, c_i^g)/\overline{U_i^1}$) among agents. As shown in Fig. 4, while PSB runs slower than PSL, PSB shows an improvement of up to 41.15% for 500 vphpl demands and 49.79% for 800 vphpl demands compared to PSL in terms of average delay. Although MILP is optimal under constant speed assumption [19], it still produces worse solutions than PSL. In terms of runtime, MILP is significantly slower than both PSL and PSB. At the same time, the difference between the average delays of PSL and MILP is much smaller than that between PSB and PSL. These improvements arise from the fact that the baseline methods assume the constant speed during intersection traversal, while our planner considers the full kinodynamic capabilities of agents. We also notice that, in the 500 vphpl scenario, the average-delay curves of all three methods level off as the number of agents increases. In contrast, in the 800 vphpl scenario, only the delay curve of PSB levels off, suggesting that PSB can find a stable solution for higher demand scenarios.

B. MAMP on Grid Model

1) Baseline Methods: In this experiment, we compare PSB with PSL and a straightforward extension of SIPP-IP [7]. SIPP-IP is a state-of-the-art single-agent path planner designed to accommodate kinodynamic constraints, making it a suitable representation of motion-primitive-based methods. The same kinodynamic motion primitives as [7] are used during the evaluation. To adapt SIPP-IP for multi-agent scenarios, we replaced Level 2 and Level 3 in PSB with the SIPP-IP.

2) Simulation Setup: We evaluate PSB, PSL, and SIPP-IP on four four-neighbor grid maps from the MAPF benchmark [4], namely `empty` (size: 32×32), `random` (size: 32×32), `lak303d` (size: 194×194), and `Boston` (size: 256×256). For each map, we conducted experiments with a progressive increment in the number of agents, using an average of 25 random instances from the benchmark set. The agents are modeled as disks with a diameter of $0.99\ cell$. All agents have identical kinodynamic constraints, which, similar to the intersection model, encompass only speed and acceleration constraints. The speed is bounded by the range of $[0, 2]\ cell/s$, while the acceleration is confined to $[-0.5, 0.5]\ cell/s^2$. To be noticed, since PSL assumes agents move at a constant speed, we have relaxed the acceleration constraints on starts and goals for PSL, allowing agents to adjust to the desired speed immediately.

3) Comparison: We evaluate solution quality using the sum of the arrival time of all agents. As shown in Fig. 5, PSB shows better solution quality than both PSL and SIPP-IP while achieving comparable or better runtime performance. PSB also outperforms them across all four maps in terms of *success rate*, which represents the ratio of instances solved within $300s$ over all instances. This can be attributed to two factors. Firstly, the longer runtime of PSL and SIPP-IP in scenarios with more agents makes it easier to hit the preset cutoff time. Secondly, the solution from baseline methods is limited in action choices, leading to failures in solving certain cases. For instance, when c_i^g is adjacent to the c_i^s, SIPP-IP encounters difficulties in finding an appropriate solution (The accelerating primitive defined in [7] takes 4 cells).

4) Ablation study: We first evaluate the influence of window size t_W by comparing our method under different window sizes: $t_W = 6$, $t_W = 12$, and $t_W = \infty$. As shown in Fig. 6, we observe marginal improvement in solution quality as we increase t_W (less than 5% from $t_W = 6$ to $t_W = \infty$), while causing a significant rise in runtime. Furthermore, as shown in Fig. 7, PSB exhibits a runtime improvement of up to 88.58% compared to PSB without the cache mechanism and 76.71% improvement over PSB without the duplicate detection mechanism. Importantly, these two mechanisms do not affect the solution quality of the algorithms.

VII. Conclusion

This paper introduces PSB, a three-level planner designed to tackle MAMP with kinodynamic constraints. PSB produces smooth solutions by effectively utilizing the full kinodynamic capacity of agents, addressing a limitation often faced by existing methods. We apply PSB to two domains: traffic intersection coordination for autonomous vehicles and obstacle-rich grid map navigation for mobile robots. In both domains, PSB outperforms the baseline methods with up to 49.79% improvement in terms of solution quality, while achieving better success rates and comparable runtime.

References

[1] J. Li, T. A. Hoang, E. Lin, H. L. Vu, and S. Koenig, "Intersection coordination with priority-based search for autonomous vehicles," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 11 578–11 585.

[2] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Persistent and robust execution of MAPF schedules in warehouses," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1125–1131, 2019.

[3] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3451–3458, 2022.

[4] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Barták, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, 2019, pp. 151–159.

[5] J. Li, W. Ruml, and S. Koenig, "EECBS: A bounded-suboptimal search for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 14, 2021, pp. 12 353–12 362.

[6] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, vol. 301, p. 103574, 2021.

[7] Z. A. Ali and K. Yakovlev, "Safe interval path planning with kinodynamic constraints," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 12 330–12 337.

[8] I. Solis, J. Motes, R. Sandström, and N. M. Amato, "Representation-optimal multi-robot motion planning using conflict-based search," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4608–4615, 2021.

[9] L. Cohen, T. Uras, T. K. S. Kumar, and S. Koenig, "Optimal and bounded-suboptimal multi-agent motion planning," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, 2019, pp. 44–51.

[10] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 7643–7650.

[11] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2011, pp. 5628–5635.

[12] G. G. Lorentz, *Bernstein polynomials*. American Mathematical Society, 2012.

[13] M. Čáp, P. Novák, J. Vokrínek, and M. Pěchouček, "Multi-agent RRT: sampling-based cooperative pathfinding," in *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, 2013, pp. 1263–1264.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] R. Hegde and D. Panagou, "Multi-agent motion planning and coordination in polygonal environments using vector fields and model predictive control," in *Proceedings of the IEEE European Control Conference*, 2016, pp. 1856–1861.

[16] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[17] W. Hönig, T. K. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Multi-agent path finding with kinematic constraints," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 26, 2016, pp. 477–485.

[18] H. Zhang, N. Tiruviluamala, S. Koenig, and T. S. Kumar, "Temporal reasoning with kinodynamic networks," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 31, 2021, pp. 415–425.

[19] M. W. Levin and D. Rey, "Conflict-point formulation of intersection control for autonomous vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 85, pp. 528–547, 2017.

[20] G. F. Newell, "A simplified theory of kinematic waves in highway traffic, part i: General theory," *Transportation Research Part B: Methodological*, vol. 27, no. 4, pp. 281–287, 1993.

[21] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[22] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 272–11 281.