Prioritized Safe Interval Path Planning for Multi-Agent Pathfinding With Continuous Time on 2D Roadmaps

Kazumi Kasaura, Mai Nishimura, and Ryo Yonetani

***Abstract*—We address a challenging multi-agent pathfinding (MAPF) problem for hundreds of agents moving on a 2D roadmap with continuous time. Despite its known potential for producing better solutions compared to typical grid and discrete-time cases, few approaches have been established to solve this problem due to the intractability of collision checks on a large scale. In this work, we propose Prioritized Safe-Interval Path Planning with Continuous-Time Conflicts (PSIPP/CTC) that extends a scalable prioritized planning algorithm to work on the 2D roadmap and continuous-time setup by alleviating intensive collision checks. Our approach involves a novel concept named Continuous-Time Conflict (CTC), which describes a pair among vertices and edges associated with continuous-time intervals within which collisions can happen between agents. We pre-compute CTCs using geometric neighbor-search and sweeping techniques and annotate roadmaps with the CTCs just once before planning starts. Doing so allows us to efficiently enumerate collision-free time intervals for all vertices and edges and find each agent's path with continuous time in prioritized planning. Extensive experimental evaluations demonstrate that PSIPP/CTC significantly outperforms existing methods in terms of planning success rate and runtime while maintaining an acceptable solution quality. As a proof of concept, we also confirmed the effectiveness of the proposed approach on a physics simulation with differential wheeled robots.**

***Index Terms*—Path planning for multiple mobile robots or agents, collision avoidance, computational geometry.**

I. Introduction

MULTI-AGENT pathfinding (MAPF) is the problem of finding collision-free paths for a team of agents from their start to goal in a given graph. MAPF has long been a classic challenge in the AI field [1], and many powerful approaches to solve it have been proposed in recent years [2], [3]. In this work, we addresss a challenging class of MAPF problems 1) *for hundreds of agents* 2) moving *on a 2D roadmap* that approximates a 2D environment in a non-grid fashion, 3) *with continuous time* such that the agents can start or stop moving at any time. Planning collision-free paths for such a large number of agents is critical for practical applications including large-scale warehouses [4], [5] and swarm robotics [6], [7]. On the other hand, planning on non-grid roadmaps with continuous time is known to lead to better solutions [8].

Nevertheless, solving the three challenges above all at the same time, namely *scalable MAPF on 2D roadmaps with continuous time*, is still technically intractable. Unlike classical MAPF on grid maps (e.g., [9], [10]) where agents can collide with each other only in the same vertices or edges, non-grid 2D roadmaps involve collisions across different vertices and edges that are in close proximity [11]. This makes collision checks computationally more intensive, and having to take a continuous time setup into account makes it further complicated as we need to check where *and when* collisions can happen. The most relevant approach in this regard is CCBS [8], which extends conflict-based search (CBS) [10] to work on 2D roadmaps and the continuous time setup by using Safe-Interval Path Planning (SIPP) [12]. However, this approach is not scalable due to the inherent limitation of CBS that requires computational time increasing very rapidly with the number of agents.

A promising direction to enable MAPF on a large scale is to adopt prioritized planning [13], a fast non-complete MAPF algorithm that determines a set of agent paths one by one in sequence. We are interested in extending the prioritized planning to work with a continuous time setup, particularly by performing SIPP to find each agent's path. This is significantly challenging as SIPP requires all possible 'safe' (i.e., collision-free) intervals to be enumerated for each vertex and edge for each agent, which is computationally too expensive. A recent work has proposed pre-computing *conflicts*, pairs among vertices and edges that can cause collisions between agents, so that planning can be done without expensive collision checks [14]. This idea has been applicable only for a discrete-time setup and motivates our key question: *"how can we enable such conflict-annotated roadmaps for a continuous-time setup?"*

To this end, we propose a novel concept named *Continuous-Time Conflict (CTC)*, the conflicts associated with time intervals that cause collisions among agents along a continuous timeline (Fig. 1(a)). CTCs can be precomputed efficiently using geometric neighbor search and sweeping algorithms, and can be used to annotate 2D roadmaps with where and when collisions can happen. Then we present a concrete algorithm called *Prioritized Safe-Interval Path Planning with Continuous-Time Conflicts (PSIPP/CTC)*, which leverages the annotated roadmaps to efficiently enumerate all possible safe intervals and find a collision-free path for each agent via SIPP (Fig. 1(b)). Note that the annotation with CTCs is required only once for each roadmap and reusable for multiple problem instances as long as the roadmap remains unchanged.

Manuscript received 23 February 2022; accepted 22 June 2022. Date of publication 30 June 2022; date of current version 4 August 2022. This letter was recommended for publication by Associate Editor A. Dutta and Editor M. Ani Hsieh upon evaluation of the reviewers' comments. *(Corresponding author: Kazumi Kasaura.)*

The authors are with the OMRON SINIC X Corporation, Tokyo 113-0033, Japan (e-mail: kazumi.kasaura@sinicx.com; mai.nishimura@sinicx.com; ryo.yonetani@sinicx.com).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3187265, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3187265

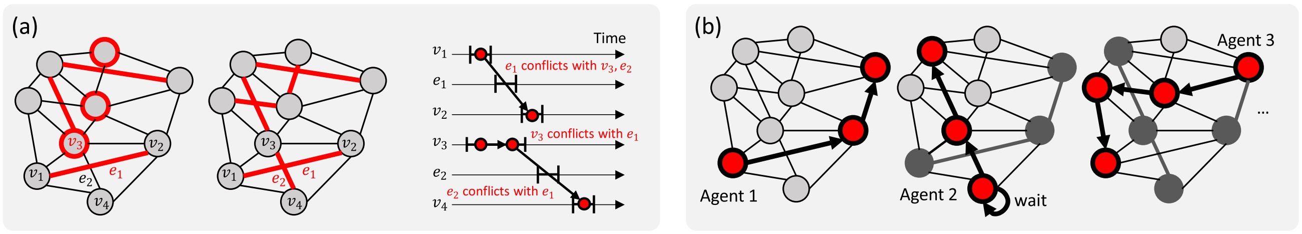


Fig. 1. Proposed approach. (a) We annotate a given 2D roadmap with *Continuos-Time Conflicts (CTCs)* that describe pairs among vertices and edges associated with temporal intervals within which collisions between agents can happen. (b) The annotated roadmap is then used by *Prioritized Safe-Interval Path Planning with CTCs* (PSIPP/CTC) that efficiently performs SIPP using CTCs to find collision-free paths for each agent during prioritized planning.

We evaluate the effectiveness of the proposed PSIPP/CTC on a variety of challenging MAPF problems. Our extensive experimental results demonstrate that, thanks to the annotation with CTCs, our approach is several orders-of-magnitude faster than CCBS [8] as well as a straight-forward extension of prioritized planning to our problem setup without CTCs, while maintaining a high success rate and acceptable solution quality. Under certain conditions, the proposed approach can even allow us to find collision-free paths for thousands of agents in just 30 seconds. As a proof-of-concept, we also confirmed the applicability of our method on a wheeled robot simulation with differential drive kinematics, which is widely adopted to real-world robots in indoor environments. We show that, without essential changes, our approach can produce feasible solutions for hundreds of robots while taking into account their acceleration, deceleration, and rotation.

II. Related Work

Multi-agent pathfinding (MAPF) has a long history in the AI and robotics fields [1]. Popular approaches include, but are not limited to, Conflict-Based Search (CBS) [10], A*-based algorithms (e.g., M* [15]), Increasing Cost Tree Search (ICTS) [16], and prioritized planning [13] (e.g., HCA* [9]). However, most of these classical works have addressed relatively simplified problems where the edges of a graph, typically a grid map given a priori, have a uniform cost and collisions happen only when agents are in the same vertices or edges. This limits their application to scenarios where such assumptions hold, such as planning in simple warehouse [5].

On the other hand, planning on a non-grid 2D roadmap has been a research topic of interest mainly for single-agent cases with the motivation of how to better approximate continuous state-spaces on the plane. To this end, diverse types of space representation have been proposed, i.e., trapezoidal map, visibility graph [17], and pathnet graph [18]. Constrained Delaunay Triangulation (CDT) [19] can produce roadmaps constrained with the obstacles while reducing the size of the adjacency graph for searching paths to be much smaller than that of grid-based representations. Aside from these geometric representations, Probabilistic Roadmaps (PRM) [20] that iteratively samples state points from continuous spaces is widely used for path planning in high-dimensional spaces [21]. In addition to these works for single-agent planning, a few recent studies have explored how to construct effective roadmaps for MAPF with discrete times [22]–[24], which is orthogonal to our focus on how planning can be done efficiently on a given roadmap.

Studies on MAPF for non-grid roadmap graphs and continuous time setups have been relatively limited. Some existing work has addressed this challenging problem by extending CBS [8], [25]–[27] and ICTS [11]. A recent work particularly relevant to our approach is CCBS [25], which leverages SIPP to plan on non-grid roadmaps with continuous time. However, all of these extensions require computational time that rapidly increases with the number of agents. Other work incorporated SIPP with prioritized planning [28], [29] to improve the scalability for the continuous-time setup, which nevertheless only assumed planning on a gridmap. Another apporach has addressed the scalability issue for non-grid graphs by enumerating all possible collisions before the planning [14], but this has only worked for the case of discrete time. In summary, there are currently no scalable approaches for MAPF with continuous time on 2D roadmaps.

III. Preliminaries

A. MAPF With Continuous Time on 2D Roadmaps

Our focus in this work is a problem of MAPF where agents move with continuous time on 2D roadmaps, i.e., an arbitrary graph consisting of vertices associated with 2D points. Such roadmaps can be constructed by, for example, running roadmap construction methods such as PRM [20] or CDT [19] on 2D environments. Following [11], [25], [27], we also assume that agents have a body modeled by a circular shape with fixed and identical radii, leave and arrive at vertices at any time in a continuous timeline, move along edges at the same constant unit speed, and stay at their own goal vertices once arrived. Two agents are regarded as being collided if their bodies are physically overlapped. Formally, let us represent a problem instance by a tuple (G, r, A), where $G = (\mathcal{V}, \mathcal{E})$ is a roadmap modeled by a directed graph with a set of vertices $\mathcal{V}$ and edges $\mathcal{E}$, r is the radius of agents, and $A = [(s_1, g_1), \ldots, (s_n, g_n)]$ is the task information for n agents represented by their start and goal $s_i, g_i \in \mathcal{V}$. Each vertex $v \in \mathcal{V}$ corresponds to a 2D location in the Euclidean state-space $\mathbb{R}^2$ and each edge $e = (v, u) \in \mathcal{E}$ connects two vertices $v, u \in \mathcal{V}$ between which there are no obstacles. We also describe a set of edges starting from a particular vertex v by $\mathcal{E}_{(v,\cdot)} \subset \mathcal{E}$, those arriving at a vertex u by $\mathcal{E}_{(\cdot,u)} \subset \mathcal{E}$, and the length of edge e by $|e|$.

On a given roadmap, agents take two types of *actions: move* along edges and *wait* on vertices. We represent a moving action along edge $e = (v, u)$ from time $t \in \mathbb{R}$ by $\mathrm{move}(e, t)$. Completing this action by moving at a constant unit speed requires the time of $|e|$. The waiting action at a vertex v from time t

until $t + \Delta t$ is given by $\text{wait}(v, t, \Delta t)$. As a result, a possible action at a vertex v from time t is: $\text{action}(v,t) \in \{\text{move}(e,t) \mid e \in \mathcal{E}_{(v,\cdot)}\} \cup \{\text{wait}(v,t,\Delta t) \mid \Delta t \in \mathbb{R}_+\}$.

The solution to an MAPF problem with n agents is a set of n collision-free *paths*, $[P_1, \ldots, P_n]$, where $P_i = [\text{action}(s_i, 0), \ldots, \text{wait}(g_i, T_i, \infty)]$ is a sequence of actions starting from s_i to the goal g_i. The variable T_i is the time it takes for the agent to arrive at the goal, which is also referred to as the *cost* of the path. Then, the quality of solutions is measured by the *sum-of-costs (SoC)* defined by $\sum_i T_i$. Our objective is to find a solution with the smallest possible SoC.

B. Solving MAPF Problems

We build our MAPF algorithm on top of the prioritized planning technique [13], which is non-optimal and incomplete but efficient and scalable to the number of agents. In the prioritized planning, agents are first assigned with a unique 'priority' determined in some way. Then, collision-free paths are searched for each agent sequentially on the basis of the assigned priority, where the paths of agents with higher priorities are regarded as dynamic obstacles for the remaining agents with lower priorities.

Any pathfinding algorithm can be used to determine each agent path, as long as it has the ability to handle dynamic objects that move, in our case, with continuous time. In this work, we adopt a popular approach called Safe Interval Path Planning (SIPP) [12]. In SIPP, roadmaps are converted into a *safe interval graph*, whose vertices (resp. edges) are a *safe interval*, i.e., a time interval within which agents can stay at a certain vertex (resp. edge) in the original roadmap without colliding with moving obstacles. Collision-free paths are then searched on the safe-interval graph such that agents are always in the safe intervals. The output of SIPP is a sequence of move or wait actions introduced in Section III-A.

In principle, constructing safe interval graphs requires *all possible collisions with all moving obstacles (agents in our case)* to be enumerated. Naively doing so results in the computational complexity of $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ for each action of each agent, making it computationally intensive to perform SIPP for prioritized planning.

IV. Continuous-Time Conflict Annotations

The key novelty of this work is the concept of *Continuous-Time Conflicts (CTCs)*, i.e., pairs among vertices and edges associated with time intervals causing collisions between agents in a continuous timeline. Roadmaps annotated with CTCs can then be used to efficiently construct and update a safe interval graph used by SIPP to find individual agent paths in the prioritized planning. In this section, we first formalize collisions in Section IV-A and CTCs in Section IV-B. Then, Section IV-C presents our technique to annotate roadmaps with CTCs using geometric neighbor-search and sweeping algorithms.

A. Collisions in Continuous Time

We consider two types of collisions between agents: *vertex-edge collisions (VEC)* and *edge-edge collisions (EEC)*. They are validated by the following predicates:

- $\text{VEC}(v, e, t_1, t_2) \in \{\text{True}, \text{False}\}$ indicating if an agent staying at a vertex $v \in \mathcal{V}$ at time $t_1 \in \mathbb{R}$ will collide with another agent starting to move along an edge $e \in \mathcal{E}$ at time $t_2 \in \mathbb{R}$, and
- $\text{EEC}(e_1, e_2, t_1, t_2) \in \{\text{True}, \text{False}\}$ checking if an agent starting moving along the edge $e_1 \in \mathcal{E}$ at time $t_1 \in \mathbb{R}$ will collide with another agent moving along the edge $e_2 \in \mathcal{E}$ from time $t_2 \in \mathbb{R}$.

Unlike the vertex-edge and edge-edge constraints in [14], these predicates explicitly take into account the continuous time setup[1]. Importantly, VECs and EECs are invariant to temporal translations:

$$\text{VEC}(v, e, t_1, t_2) \Leftrightarrow \quad \text{VEC}(v, e, t_1 + \tau, t_2 + \tau), \tag{1}$$

$$\text{EEC}(e_1, e_2, t_1, t_2) \Leftrightarrow \quad \text{EEC}(e_1, e_2, t_1 + \tau, t_2 + \tau). \tag{2}$$

This property further allows us to introduce *collision intervals*, the time intervals that contain collisions as follows:

$$I_{\text{VEC}}(v, e) := \{t \mid \text{VEC}(v, e, 0, t)\}, \tag{3}$$

$$I_{\text{EEC}}(e_1, e_2) := \{t \mid \text{EEC}(e_1, e_2, 0, t)\}. \tag{4}$$

Here, evaluating collisions VEC and EEC as well as calculating collision intervals $I_{\text{VEC}}, I_{\text{EEC}}$ can be done algebraically with the method presented in [30]; in other words, they can be evaluated in constant time.

B. Continuous-Time Conflicts (CTCs)

We shall call vertex v and edge e (resp. two edges e_1 and e_2) form a continuous-time conflict (CTC) if the time interval defined by Eq. (3) (resp. Eq (4)) is not empty. Let us define a *vertex-edge CTC* by a tuple $(v, e, I) \in \mathcal{V} \times \mathcal{E} \times \text{Int}$ and *edge-edge CTC* by $(e_1, e_2, I) \in \mathcal{E} \times \mathcal{E} \times \text{Int}$, where Int is a set of all possible non-empty intervals with positive lengths on real numbers. Moreover, let us describe a set of all possible vertex-edge CTCs by $\mathcal{C}_{\text{VEC}}$ and that of edge-edge CTCs by $\mathcal{C}_{\text{EEC}}$.

Crucially, we can derive the following relationships between VEC, EEC and $\mathcal{C}_{\text{VEC}}, \mathcal{C}_{\text{EEC}}$ from the invariance to temporal translations of VEC and EEC:

$$\begin{aligned} &\text{VEC}(v, e, t_1, t_2) \\ &\iff \exists I \in \text{Int} : (v, e, I) \in \mathcal{C}_{\text{VEC}} \wedge t_2 - t_1 \in I, \end{aligned} \tag{5}$$

$$\begin{aligned} &\text{EEC}(e_1, e_2, t_1, t_2) \\ &\iff \exists I \in \text{Int} : (e_1, e_2, I) \in \mathcal{C}_{\text{EEC}} \wedge t_2 - t_1 \in I. \end{aligned} \tag{6}$$

In other words, when $\mathcal{C}_{\text{VEC}}$ and $\mathcal{C}_{\text{EEC}}$ are available, we can check if two agents in a vertex v and an edge e or edges e_1 and e_2 will collide by just evaluating the right-hand-side of the formulae above. This motivates us to annotate a given roadmap with all the CTCs, only once before the planning.

C. Annotating Roadmaps With CTCs

While annotating roadmaps with CTCs would make it efficient to search for a collision-free path for each agent in prioritized planning, naively doing such annotation just by checking all pairs of vertices and edges, as done in [14], is impractical for large graphs due to its time complexity of $\mathcal{O}((|V| + |E|)^2)$ in total. Our key insight to this end is that, for 2D roadmaps, the whole annotation process reduces to a pure geometric problem and can be solved efficiently by leveraging neighbor search and sweeping algorithms in the literature of computational geometry.

[1]Unlike [14], we do not consider the vertex-vertex constraint because in our definition it can be viewed as a part of the vertex-edge collisions.

Algorithm 1: Annotate Roadmaps with CTCs.

```
1: function ANNOTATEROADMAPSWITHCTCS($\mathcal{V}, \mathcal{E}, r$)
2:   $P_{\text{VEC}} \leftarrow \emptyset$, $P_{\text{EEC}} \leftarrow \emptyset$, $\mathcal{C}_{\text{VEC}} \leftarrow \emptyset$, $\mathcal{C}_{\text{EEC}} \leftarrow \emptyset$
3:   $P_A \leftarrow$ FIXEDRADIUSNEARNEIGHBOR ($\mathcal{V}, 2r$)
4:                 $\triangleright$ Returns a set of pairs of points within $2r$.
5:   for all $\{u, v\} \in P_A$ do
6:     for all $e \in \mathcal{E}_{(u,\cdot)} \cup \mathcal{E}_{(\cdot,u)}$ do
7:       $P_{\text{VEC}} \leftarrow P_{\text{VEC}} \cup \{(v, e)\}$
8:     end for
9:     for all $e \in \mathcal{E}_{(v,\cdot)} \cup \mathcal{E}_{(\cdot,v)}$ do
10:      $P_{\text{VEC}} \leftarrow P_{\text{VEC}} \cup \{(u, e)\}$
11:    end for
12: end for
13:  for all $e \in \mathcal{E}$ do
14:  for all $v \in \mathcal{V} \cap$ PerpendicularRectangle $(e, 2r)$ do
15:      $P_{\text{VEC}} \leftarrow P_{\text{VEC}} \cup \{(v, e)\}$
16:    end for
17:  end for
18:  for all $(v, e) \in P_{\text{VEC}}$ do
19:    $I \leftarrow I_{\text{VEC}}(v, e)$
20:    $\mathcal{C}_{\text{VEC}} \leftarrow \mathcal{C}_{\text{VEC}} \cup \{(v, e, I)\}$
21:  end for
22:  for all $(v, e) \in P_{\text{VEC}}$ do
23:    for all $e' \in \mathcal{E}_{(v,\cdot)} \cup \mathcal{E}_{(\cdot,v)}$ do
24:      $P_{\text{EEC}} \leftarrow P_{\text{EEC}} \cup \{\{e, e'\}\}$
25:    end for
26:  end for
27:  $P_C \leftarrow$ BENTLEYOTTMANN ($\mathcal{E}$)
28:                 $\triangleright$ Returns a set of crossing pairs of edges.
29:  $P_{\text{EEC}} \leftarrow P_{\text{EEC}} \cup P_C$
30:  for all $\{e, e'\} \in P_{\text{EEC}}$ do
31:    $I \leftarrow I_{\text{EEC}}(e, e')$
32:    $\mathcal{C}_{\text{EEC}} \leftarrow \mathcal{C}_{\text{EEC}} \cup \{(e, e,' I), (e,' e, -I)\}$
33:  end for
34:  return $\mathcal{C}_{\text{VEC}}, \mathcal{C}_{\text{EEC}}$
35: end function
```

The overall pseudocode for the proposed annotation algorithm is summarized in Algorithm 1, which involves the enumeration of $\mathcal{C}_{\text{VEC}}$ and $\mathcal{C}_{\text{EEC}}$ shown below.

1) Enumerating $\mathcal{C}_{VEC}$: Recall that we denote the agent size by r. To enumerate $\mathcal{C}_{\text{VEC}}$, it is enough for each vertex v to retrieve a subset of edges $\mathcal{E}' \subset \mathcal{E}$ with the distance smaller than $2r$ because their collision interval can be calculated in constant time. Such an edge should satisfy the condition that 1) at least one of its endpoints is within $2r$ of the vertex v or 2) there exists a perpendicular line from v to the edge that is smaller than $2r$. Checking the first condition can be done efficiently by the fixed-radius near neighbor (FRNN) algorithm in the time complexity of $\mathcal{O}(|V| + M_1)$, where M_1 is the number of pairs to enumerate[2] [17]. Furthermore, we observe that a set of points satisfying the second condition against a certain edge form a rectangle (PerpendicularRectangle$(e, 2r)$ in the algorithm). Therefore, for every edge in the roadmap, we retrieve vertices found within that rectangle and add them with the edge to $\mathcal{C}_{\text{VEC}}$.

[2]While M_1 is at most $\Theta(|V|^2)$ and M_2 is at most $\Theta(|E|^2)$ for concentrated roadmaps, $M_1 \ll |V|^2$ and $M_2 \ll |E|^2$ for practical roadmaps such as PRM and CDT, where the vertices and the edges are scattered.

Algorithm 2: PSIPP with Continuous-Time Conflicts.

```
1: function PSIPP/CTC($G = [\mathcal{V}, \mathcal{E}], r, A, \mathcal{C}_{\text{VEC}}, \mathcal{C}_{\text{EEC}}$)
2:   Sort $A$ based on the predefined priority
3:   for all $v \in \mathcal{V}$ do
4:     $I_v \leftarrow (-\infty, +\infty)$
5:   end for
6:   for all $e \in \mathcal{E}$ do
7:     $I_e \leftarrow (-\infty, +\infty)$
8:   end for
9:   $\mathcal{I} \leftarrow ((I_v)_{v \in \mathcal{V}}, (I_e)_{e \in \mathcal{E}})$
10:  for all $(s_i, g_i) \in A$ do
11:    $P_i \leftarrow$ SIPP $(s_i, g_i, \mathcal{I})$
12:    if $P_i =$ failure then
13:      return failure
14:    end if
15:    for all wait$(v, t, \Delta t)$ in $P_i$ do
16:      for all $(e, \tau_0, \tau_1) \in \mathcal{E}_{\text{VEC}}(v)$ do
17:        $I_e \leftarrow I_e \setminus [t + \tau_0, t + \Delta t + \tau_1]$
18:      end for
19:    end for
20:    for all move$(e, t)$ in $P_i$ do
21:      for all $(v, \tau_0, \tau_1) \in \mathcal{V}_{\text{VEC}}(e)$ do
22:        $I_v \leftarrow I_v \setminus [t + \tau_0, t + \tau_1]$
23:      end for
24:      for all $(e', \tau_0, \tau_1) \in \mathcal{E}_{\text{EEC}}(e)$ do
25:        $I_{e'} \leftarrow I_{e'} \setminus [t + \tau_0, t + \tau_1]$
26:      end for
27:    end for
28:  end for
29:  return $\mathcal{P} = [P_1, \ldots, P_n]$
30: end function
```

To do this, we use a spatial partition technique [31] with a uniform grid having the cell size of $2r$.

2) Enumerating $\mathcal{C}_{EEC}$: As for pairs of conflicting edges, they should satisfy the condition that 1) one of the endpoints of one of the edges is within $2r$ of the other edge or 2) the two edges cross. The first condition is exactly what we check to enumerate $\mathcal{C}_{\text{VEC}}$ and has already been solved. Further, all crossing pairs of edges can be swept efficiently by using the Bentley-Ottmann algorithm with the time complexity of $\mathcal{O}((|\mathcal{E}| + M_2) \log |\mathcal{E}|)$, where M_2 is the number of pairs to enumerate[2] [32].

V. Prioritized SIPP With CTCs

Algorithm 2 presents the proposed Prioritized Safe-Interval Path Planning with Continuous-Time Conflicts (PSIPP/CTC). Here we introduce the individual safe intervals for every vertex and edge, i.e., $(I_v)_{v \in \mathcal{V}}, (I_e)_{e \in \mathcal{E}}$, which are shared across all the agents, used in SIPP to produce each agent path, and updated incrementally on the basis of each SIPP result.

Specifically, let us define the following functions that return sets of vertices and edges with intervals within which they conflict with a query vertex or edge:

$$\mathcal{V}_{\text{VEC}}(e) := \{(v, \tau_0, \tau_1) \mid v \in \mathcal{V}, I \in \text{Int}, (v, e, I) \in \mathcal{C}_{\text{VEC}}\},$$

$$\mathcal{E}_{\text{VEC}}(v) := \{(e, \tau_0, \tau_1) \mid e \in \mathcal{E}, I \in \text{Int}, (v, e, I) \in \mathcal{C}_{\text{VEC}}\},$$

$$\mathcal{E}_{\text{EEC}}(e) := \{(e,' \tau_0, \tau_1) \mid e' \in \mathcal{E}, I \in \text{Int}, (e, e,' I) \in \mathcal{C}_{\text{EEC}}\},$$

Algorithm 3: getSuccessors for SIPP.

```
1:  function GETSUCCESSORS($\mathcal{I}, v, (t_0, t_1), t$)
2:    $Succ \leftarrow \emptyset$
3:    for all $e = (v, u) \in E_{(v,\cdot)}$ do
4:      $\mathcal{J}_e \leftarrow \{J \in \mathrm{comp}(I_e) \mid (t, t_1) \cap J \neq \emptyset\}$
5:      for all $(\tau_0, \tau_1) \in \mathcal{J}_e$ do
6:        $(t'_0, t'_1) \leftarrow (t, t_1) \cap (\tau_0, \tau_1)$
7:        $(s_0, s_1) \leftarrow (t'_0 + |e|, t'_1 + |e|)$
8:        $\mathcal{J}_u \leftarrow \{J \in \mathrm{comp}(I_u) \mid (s_0, s_1) \cap J \neq \emptyset\}$
9:        for all $(\tau'_0, \tau'_1) \in \mathcal{J}_u$ do
10:         $(s'_0, s'_1) \leftarrow (s_0, s_1) \cap (\tau'_0, \tau'_1)$
11:         $Succ \leftarrow Succ \cup \{(u, (\tau'_0, \tau'_1), s'_0)\}$
12:       end for
13:     end for
14:   end for
15:   return $Succ$
16: end function
```

where τ_0, τ_1 are the endpoints of intervals I. These functions are used to retrieve vertices and edges in L16, L21, and L24 to update safe intervals in the subsequent lines. Each of these updates can be implemented as an operation on the set of intervals managed by a balanced binary tree, whose amortized time is logarithmic to the size of the set. Accordingly, the time complexity for updating safe intervals throughout the algorithm is log-linear to the number of collisions between paths and the vertices or edges of a given roadmap.

In addition, we extend the original SIPP by using Algorithm 3 to obtain neighbor nodes in A* search. Here, a pair of a vertex v and its safe interval (t_0, t_1) represents a vertex of the safe interval graph and t is the time to reach this vertex, where $t_0 \leq t \leq t_1$. Also, let $\mathrm{comp}(I)$ be the set of the connected components of I, which is, in this case, a set of intervals. It is not necessary for the algorithm to calculate the successors $\mathcal{J}_e$ and $\mathcal{J}_u$ by brute force because the range of $\mathcal{J}_e$ for each $e \in \mathcal{E}$ and that of $\mathcal{J}_u$ for each $u \in \mathcal{V}$ shift monotonously. This makes the time complexity of SIPP log-linear to the number of visited vertices of the safe interval graph.

VI. Evaluation

In this section, we first evaluate the three key aspects of the proposed PSIPP/CTC: (1) overall performance comparisons with several different roadmap generation methods as well as the density of the generated roadmaps; (2) applicability for various environments with different obstacle layouts; and (3) scalability for a large number of agents. Further, we demonstrate using a physical simulation that, without any essential change of our algorithm, our method can easily be adopted to wheeled robots with differential drive kinematics.

A. Problem Setup

a) Environments: Figure 2 shows the environments used in the experiments: **den** (`den520 d`), **room** (`room-64-64-16`), **random** (`random-64-64-10`), **maze** (`maze-128-128-10`), **warehouse** (`warehouse-20-40-10-2-2`), **Berlin** (`Berlin_1_256`), and **empty** (i.e., no obstacles). All the environments but **empty** are from the MAPF benchmark [1], which is a popular choice in prior work [8], [10], [16], [25], [27], [33]. As done in [8], [25], we use **den** as a baseline environment to evaluate the general performance and scalability of the proposed approach. As a preprocessing, we simplify each map by the Douglas-Peucker algorithm in the Boost C++ Library with the max distance of 1.0.

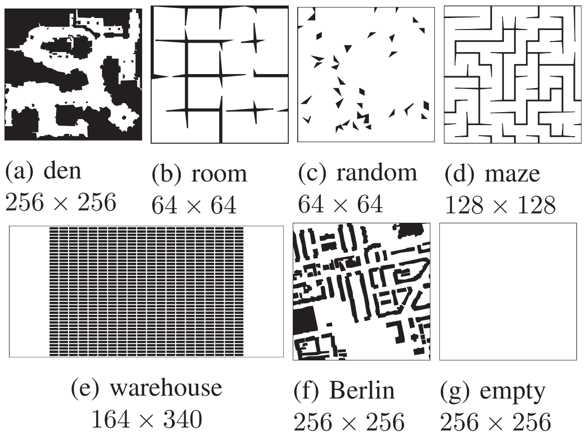


Fig. 2. Environment maps used in the experiments.

b) Roadmaps: We constructed roadmaps by connecting vertices sampled from free space bidirectionally using a variant of PRM [20] called k-sPRM [34] with $k = 15$ [35] or CDT [19]. For k-sPRM, we connect each vertex with its nearest neighbors if the connecting edges are fully inside the free space. For CDT, in addition to the sampled vertices, we also take vertices on the boundaries of the free space. These vertices are then used to construct the constrained Delaunay triangulation while ensuring that segments on the boundaries are always treated as edges of the triangles. Overall, roadmaps generated by k-sPRM have many edges as well as edge-edge CTCs in open spaces, while there are many edges around border regions and few edge-edge CTCs with CDT. The use of k-sPRM roadmaps often leads to better solutions (i.e., lower SoCs), as they have many edges. On the other hand, CDT roadmaps are advantageous for connectingregions in narrow par0s.

B. Experimental Setup

a) Methods: We compare PSIPP/CTC with CCBS [25] as it is the most relevant approach to our problem setting with continuous-time and 2D roadmaps. We used the official implementation of CCBS available online[3]. We also evaluate a degraded version of our approach referred to as PSIPP, which computes $\mathcal{V}_{\mathrm{VEC}}(e)$, $\mathcal{E}_{\mathrm{VEC}}(v)$, and $\mathcal{E}_{\mathrm{EEC}}(e)$ by checking all vertices and edges. For PSIPP/CTC and PSIPP, we determined the priority of agents in a simple FIFO fashion, i.e., agents with younger indices got higher priorities.

b) Evaluation scheme: Agent radius r was set to 0.5 through all experiments. For each combination of environments and roadmap construction methods, we evaluated the aforementioned MAPF methods as follows with a density parameter N, which is also the upper bound of the number of agents. 1) We first sampled N random pairs of vertices from the free space of environment as start and goal positions of agents, i.e., $(s_1, g_1), \ldots, (s_N, g_N)$, while ensuring that no two starts and no

[3]`github.com/PathPlanning/Continuous-CBS`

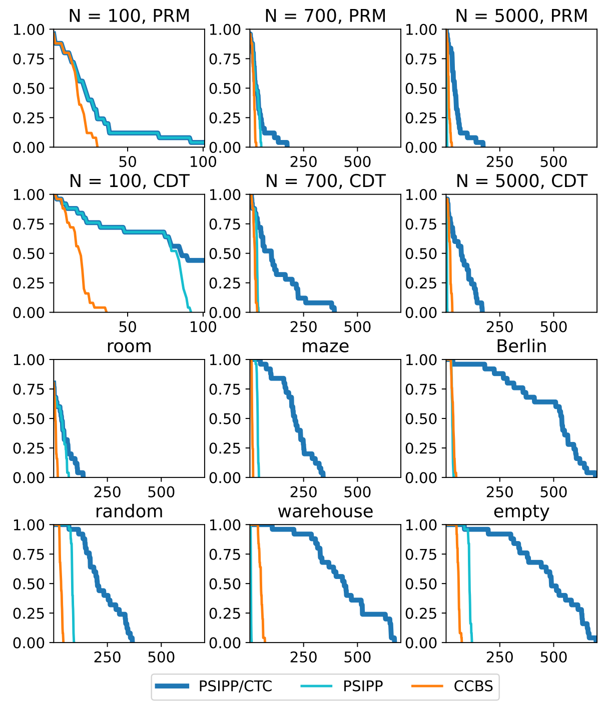


Fig. 3. Success rate for Result 1 and 2. Horizontal axis: number of agents; vertical axis: success rate.

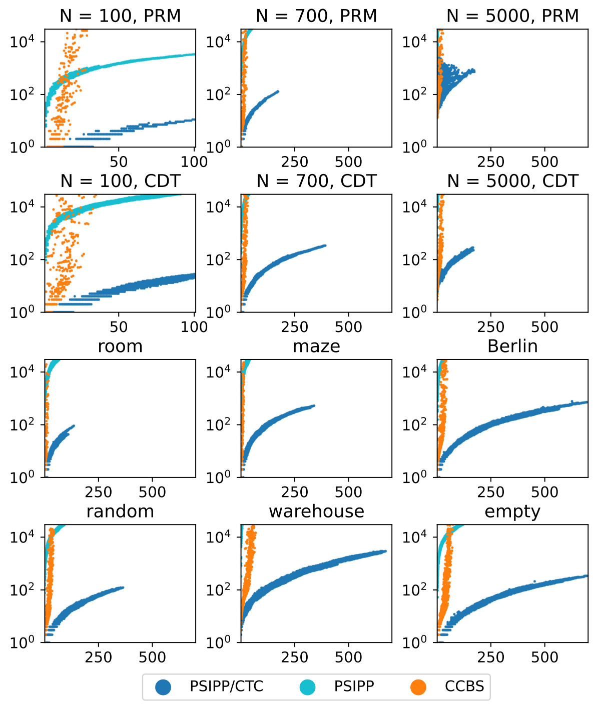


Fig. 4. Scatter plots of runtime for Result 1 and 2. Horizontal axis: number of agents; vertical axis: runtime (ms).

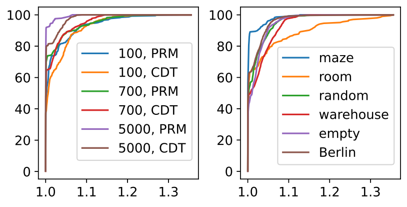


Fig. 5. Cumulative distributions for the ratio of SoCs. Horizontal axis: ratio of SoC; vertical axis: percentile.

two goals are closer than $2r$; 2) Then, we used these 2 N vertices (and some additional vertices on the boundaries for CDT) to construct a roadmap. 3) We annotated the constructed roadmap with CTCs. 4) On the roadmap, we solved tasks starting from $A_1 = [(s_1, g_1)]$ (i.e., single agent), $A_2 = [(s_1, g_1), (s_2, g_2)]$, and gradually increased n for $A_n = [(s_1, g_1), \ldots, (s_n, g_n)]$ until the planning resulted in failure or exceeded the time limit of 30 seconds following [8]. We repeated these three steps 25 times with different random seeds. Once all evaluations were completed, we calculated the success rate for each roadmap construction method and each number of agents. We also measured the runtime for some of the tasks that each method successfully solved. In addition, for tasks at which both PSIPP/CTC and CCBS succeeded, we calculated the ratio of SoCs between them to determine how close the solutions by PSIPP/CTC are to the optimal one provided by CCBS. All the methods are implemented in C++ and evaluated with Intel Core i9-9900 K CPU and 32 GB RAM.

C. Result 1: Performance Comparisons With Baselines

We first investigated the performance of each algorithm with different roadmaps with different construction methods (PRM and CDT) and densities with three different N: sparse ($N = 100$), dense ($N = 700$), and very dense ($N = 5000$) similar to [8]. The upper half of Figure 3 show the success rates for the **den** environment. Overall, PSIPP/CTC succeeded with a higher probability than CCBS and PSIPP, especially for the cases with a larger number of agents. Although PSIPP essentially adopts the same planning algorithm as PSIPP/CTC, the absence of annotations with CTCs led to planning timeout for dense and very dense roadmaps, limiting its success rate. We also confirmed that success rates with the CDT roadmaps were consistently higher than those with k-sPRM. One possible reason is that the **den** environment contains some narrow regions within which k-sPRMs often fail to connect edges. Another minor finding is that increasing the density did not necessarily contribute to high success rates for CDT, as shown in the results with $N = 700$ and $N = 5000$.

The upper half of Figure 4 shows the semi-log scatter plots of runtime in milliseconds with respect to the numbers of agents. PSIPP/CTC was several orders-of-magnitude faster than CCBS and PSIPP, thanks to the efficiency of prioritized planning combined with the CTCs annotated already before the planning (see Section VI-F for the runtime required for the roadmap annotation.) We also note that the runtimes of PSIPP/CTC and PSIPP have less variance compared to those of CCBS. This is because these approaches are built on top of the prioritized planning that requires all safe intervals to be enumerated, whereas CCBS is based on CBS that requires different runtimes depending on how many collisions are detected during planning. Nevertheless, we found that the runtime of PSIPP/CTC had higher variance for very-dense PRM, possibly due to the high non-uniformity of the local density of roadmaps. Figure 5 shows the cumulative distributions for the ratio of SoCs between PSIPP/CTC and

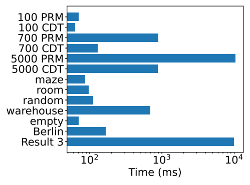
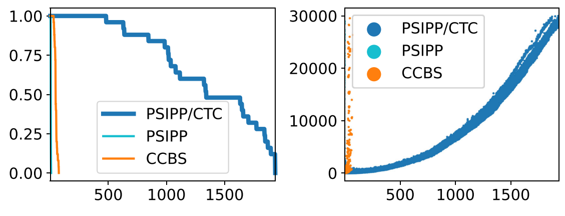


Fig. 6. Average of runtimes (ms) for roadmap annotation.



(a) Success rate (b) Runtime

Fig. 7. Result 3: (a) success rate and (b) runtime (ms).

CCBS. More rapid growth of percentile means that our method provided effective solutions for more problem instances. We can see that 1) denser roadmaps led to better ratios, and 2) the use of k-sPRM with $N = 5000$ resulted in the almost optimal ratio (precisely, less than 1.001) for more than 90% of the tasks.

D. Result 2: Effect of Environments

Next, we evaluated how the performances of our approach and the baselines vary for diverse environments with different obstacle layouts. We used CDT roadmaps with $N = 700$ here as they performed well in the previous experiment. The lower half of Figs. 3 and 4 respectively show success rates and runtime results. We confirmed that PSIPP/CTC consistently outperformed the other methods in both of these metrics on all the environments. Even so, there were some differences in performance depending on the environment. For example, **warehouse**, **Berlin**, and **empty** were relatively easy environments to solve due to their large map sizes and the absence of narrow passages. For smaller maps such as **maze** and **random**, the success rates were rather limited. The **room** environment was the most difficult to solve as it involves many narrow regions.

Figure 5 shows the cumulative distributions for the ratios of SoCs. PSIPP/CTC performed the best in the **maze** environment, while demonstrating rather limited performances in **room**. We speculate that path costs become high in **room** when the order of agents (i.e., priority given in a FIFO fashion) to go through the narrow parts is not optimal.

E. Result 3: Scalability to the Number of Agents

We also explored the scalability of the proposed method to the number of agents. Specifically, we constructed very dense roadmaps with $N = 5000$ by k-sPRM on the top of **empty** space, such that the planning failures were mostly due to the timeout (after 30 seconds). Figure 7 shows the success rate and runtime. The proposed PSIPP/CTC could solve problem instances with even up to 2,000 agents in 30 seconds, which is several orders-of-magnitude larger than CCBS that could solve problems with only less than 100 agents. However, note that the runtime grows non-linearly with respect to the number of agents. This is because the size of safe interval graphs increases with the number of agents, which affects the runtime in a log-linear fashion (as described in Section V).

F. Runtime for Roadmap Annotation

Figure 6 reports the average runtime required for annotating roadmaps with CTCs in each experiment. As long as the roadmaps were not extremely dense, the required time was acceptable (less than 1 s). Even for the most dense roadmap with $N = 5000$, the runtime for the annotation was no longer than 10 seconds. Crucially, once annotated with CTCs, roadmaps can be reused anytime as long as the obstacle layouts remain unchanged. Therefore, our approach is particularly advantageous for multi-query setups where multiple tasks are solved in the same environment.

G. Evaluation on the Wheeled Robot Simulation

To further validate the applicability of the proposed approach for practical scenarios, we investigate if the solutions are executable by wheeled robots via a simulation. Our simulator is built using PyBullet [36], and simulates a team of wheeled robots moving with the differential drive kinematics [37] that is one of the most popular mechanisms for many real robots working in indoor environments [38].

The proposed approach can allow for the gaps arising from rotations, acceleration, and deceleration of differential drive systems with a slight modification. Specifically, we (a) constrain robots to wait at the current vertex within a specified time depending on their rotation angles and (b) approximate their trajectories with acceleration and deceleration by a ramp function consisting of “awaiting” and “moving at constant speed” actions. As a result, we confirmed that a team of 300 simulated robots successfully followed the solution paths while avoiding collisions. Further details are available in the supplementary video. Note that it would be possible to further optimize robot trajectories via trajectory optimization (e.g., [39]), which we leave for future work.

H. Limitations and Possible Extensions

Finally, we discuss some limitations and possible extensions of the proposed approach. The time to annotate roadmaps with CTCs becomes non-negligible when the number of agents is smaller or the number of edges in a roadmap becomes larger (e.g., the fully connected grids considered in [28]), thus limiting its effectiveness. Moreover, as it is built on top of prioritized planning, the proposed PSIPP/CTC is non-optimal and non-complete and sometimes results in inefficient solutions or planning failures especially when the environment involves narrow regions. Possible extensions to mitigate this issue include an any-time approach [40], a re-ordering strategy [41], and priority-based search [42]. Adopting the revised prioritized planning [43] would further enhance the proposed approach with a guarantee on completeness under some assumptions as in [43].

Another interesting direction is an extension of the proposed approach to higher-dimensional roadmaps, non-straight edges, or agents with diverse shapes and kinematics. Efficient techniques to annotate roadmaps with CTCs for such challenging scenarios remain an open question.

VII. Conclusion

In this work, we proposed a scalable approach to MAPF with continuous time on 2D roadmaps. The key technical novelty is the concept of CTCs that can describe pairs among vertices and edges associated with continuous time intervals within which agents can collide. By annotating the roadmaps with the CTCs precomputed before the planning, our approach can enable the prioritized planning with SIPP to find collision-free paths along a continuous timeline for a large number of agents. The effectiveness of the proposed approach was confirmed with an extensive evaluation including a wheeled robot simulation. We believe that our work opens new avenues of applications of MAPF, including large-scale warehouse management and factory automation as well as swarm robotics.

References

[1] R. Stern et al., "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proc. Annu. Symp. Combinatorial Search*, 2019, pp. 151–158.

[2] A. Felner et al., "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *Proc. Int. Symp. Combinatorial Search*, 2017, vol. 8, no. 1, pp. 29–37.

[3] R. Stern, "Multi-agent path finding—An overview," in *Artif. Intell.* Cham, Switzerland: Springer, 2019, pp. 96–115.

[4] E. Ackerman, "Amazon uses 800 robots to run this warehouse," *IEEE Spectr.*, 2019. [Online]. Available: https://spectrum.ieee.org/amazon-introduces-two-new-warehouse-robots

[5] 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 *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2020, pp. 1898–1900.

[6] R. Ramaithitima, M. Whitzer, S. Bhattacharya, and V. Kumar, "Automated creation of topological maps in unknown environments using a swarm of resource-constrained robots," *IEEE Robot. Automat. Lett.*, vol. 1, no. 2, pp. 746–753, Jul. 2016.

[7] B. Araki, J. Strang, S. Pohorecky, C. Qiu, T. Naegeli, and D. Rus, "Multi-robot path planning for a swarm of robots that can both fly and drive," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 5575–5582.

[8] A. Andreychuk, K. Yakovlev, E. Boyarski, and R. Stern, "Improving continuous-time conflict based search," in *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 13, pp. 11220–11227.

[9] D. Silver, "Cooperative pathfinding," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2005, vol. 1, no. 1, pp. 117–122.

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

[11] T. T. Walker, N. R. Sturtevant, and A. Felner, "Extended increasing cost tree search for non-unit cost domains," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 534–540.

[12] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 5628–5635.

[13] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 1, pp. 477–521, 1987.

[14] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 856–869, Aug. 2018.

[15] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artif. Intell.*, vol. 219, pp. 1–24, 2015.

[16] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 195, pp. 470–495, 2013.

[17] D. M. Mount. (2011)., "CMSC 754 computational geometry," Univ. Maryland, College Park, MD, USA, 2002, pp. 1–122.

[18] C. S. Mata and J. S. Mitchell, "A new algorithm for computing shortest paths in weighted planar subdivisions," in *Proc. Annu. Symp. Comput. Geometry*, 1997, pp. 264–273.

[19] M. Kallmann, "Path planning in triangulations," in *Proc. IJCAI Workshop Reasoning, Representation, Learn. Comput. Games*, 2005, pp. 49–54.

[20] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[21] S. R. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Proc. Int. Symp. Robot. Res.*, Berlin, Heidelberg, 2005, pp. 36–54.

[22] F. F. Arias, B. Ichter, A. Faust, and N. M. Amato, "Avoidance critical probabilistic roadmaps for motion planning in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 10264–10270.

[23] K. Okumura, R. Yonetani, M. Nishimura, and A. Kanezaki, "CTRMs: Learning to construct cooperative timed roadmaps for multi-agent path planning in continuous spaces," in *Proc. Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2022, pp. 972–981.

[24] C. Henkel and M. Toussaint, "Optimized directed roadmap graph for multi-agent path finding using stochastic gradient descent," in *Proc. Annu. ACM Symp. Appl. Comput.*, 2020, pp. 776–783.

[25] A. Andreychuk, K. Yakovlev, P. Surynek, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," *Artif. Intell.*, vol. 305, 2022, Art. no. 103662.

[26] L. Cohen, T. Uras, T. S. Kumar, and S. Koenig, "Optimal and bounded-suboptimal multi-agent motion planning," in *Proc. Annu. Symp. Combinatorial Search*, 2019, pp. 44–51.

[27] P. Surynek, "Multi-agent path finding with continuous time and geometric agents viewed through satisfiability modulo theories (SMT)," in *Proc. Annu. Symp. Combinatorial Search*, 2019, vol. 10, no. 1, pp. 200–201.

[28] K. Yakovlev and A. Andreychuk, "Any-angle pathfinding for multiple agents based on SIPP algorithm," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2017, pp. 586–593.

[29] Z. A. Ali and K. Yakovlev, "Prioritized SIPP for multi-agent path finding with kinematic constraints," in *Int. Conf. Interactive Collaborative Robot.*, Cham, Switzerland: Springer, 2021, pp. 1–13.

[30] T. T. Walker and N. R. Sturtevant, "Collision detection for agents in multi-agent pathfinding," 2019, *arXiv:1908.09707v3*.

[31] C. Ericson, *Real-Time Collision Detection*. Boca Raton, FL, USA: CRC, 2004.

[32] J. L. Bentley and T. A. Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Trans. Comput.*, vol. C-28, no. 9, pp. 643–647, Sep. 1979.

[33] P. Surynek, "Multi-agent path finding modulo theory with continuous movements and the sum of costs objective," in *Proc. German Conf. Artif. Intell.*, Cham, Switzerland: Springer, 2020, pp. 219–232.

[34] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[35] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic Foundations of Robotics V*. Berlin, Heidelberg: Springer, 2004, pp. 43–57.

[36] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016–2022. [Online]. Available: https://pybullet.org

[37] D. J. Balkcom and M. T. Mason, "Time optimal trajectories for bounded velocity differential drive vehicles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 199–217, 2002.

[38] K. Yakovlev, A. Andreychuk, and V. Vorobyev, "Prioritized multi-agent path finding for differential drive robots," in *Proc. IEEE Eur. Conf. Mobile Robots*, 2019, pp. 1–6.

[39] M. Turpin, K. Mohta, N. Michael, and V. Kumar, "Goal assignment and trajectory planning for large teams of interchangeable robots," *Auton. Robots*, vol. 37, no. 4, pp. 401–415, 2014.

[40] J. Li, Z. Chen, D. Harabor, P. Stuckey, and S. Koenig, "Anytime multi-agent path finding via large neighborhood search," in *Proc. Int. Joint Conf. Artif. Intell.*, 2021, pp. 4127–4135.

[41] A. Andreychuk and K. Yakovlev, "Two techniques that enhance the performance of multi-robot prioritized path planning," in *Proc. Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2018, pp. 2177–2179.

[42] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 01, pp. 7643–7650.

[43] M. Čáp, P. Novák, A. Kleiner, and M. Seleckỳ, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Trans. Automat. Sci. Eng.*, vol. 12, no. 3, pp. 835–849, Jul. 2015.