Cooperative Task and Motion Planning for Multi-Arm Assembly Systems

Jingkai Chen[1], Jiaoyang Li[2,*], Yijiang Huang[1,*], Caelan Garrett[1,4], Dawei Sun[3],
Chuchu Fan[1], Andreas Hofmann[1], Caitlin Mueller[1], Sven Koenig[2], Brian C. Williams[1]

Abstract—**Planning for a team of robots to complete varying assembly tasks in dynamic environments is one of the key technical challenges for developing the next generation of manufacturing automation. In this paper, we present a cooperative task and motion planning framework that jointly plans safe, high-quality assembly plans for multiple robot arms to assemble complex spatial structures. We demonstrate our planning system on several challenging domains including Lego bricks, bars, plates, and irregular-shaped blocks with up to three robots with grippers or suction plates for assembling up to 23 objects. We show that, with our framework, assembly plans can be automatically planned for various complex structures and potentially applied to real-world manufacturing.**

I. Introduction

In the past decades, the number of robots deployed for repetitive assembly tasks in factories has been significantly increased. However, most robotic assembly automation is manually designed and usually takes months to accommodate new production needs. In the coming generation, robotic assembly automation is expected to be more adaptable to flexible assembly requirements. This calls for automating robotic assembly automation with robotic task and motion planning with respect to varying machine setups and products.

As planning single robot manipulators towards a desired goal configuration have been studied for decades and can be solved reasonably fast, there are other three open challenges in task and motion planning for generating assembly plans: (1) *planning over a long horizon:* assembly requirements in manufacturing typically include a large number of different tasks (e.g., assembling, holding, and welding) over a long horizon, and these tasks are often tightly coupled with precedence constraints and concurrency requirements given the product properties, which imposes significant computational challenges; (2) *cooperative planning for heterogeneous robot teams:* deploying multiple robots with different tools in manufacturing is critical to increasing assembly productivity and assembling complex products via cooperation. Planning with such a cooperative robot team significantly increases the problem dimension; (3) *optimizing assembly time:* optimizing makespan (i.e., total assembly time) is critical for industrial applications. Finding high-quality solutions that are optimal or near-optimal for multi-task, multi-robot assembly planning problems requires thorough consideration over all the aspects mentioned above.

Consider the example in Fig. 1. Two robot arms R1 and R2 must detach and attach the bricks to move them from their start configurations to their goal configurations (see top left). The assembly task plan is given in the bottom left of Fig. 1. In particular, after the blue brick (B) is assembled in its goal configuration, one robot needs to hold B in order for the other robot to attach the red brick (R) (see Steps 4 and 5 in bottom right of Fig. 1). One needs to assign these tasks (see the assignment in the top right of Fig. 1) and plan paths that adhere to the assignments while avoiding three types of collisions (see the bottom middle of Fig. 1). The bottom right sketches a plan with six critical moves.

We propose a cooperative task and motion planning framework to plan safe, high-quality assembly plans for multi-task multi-robot assembly problems, which attempts to address the above challenges. The framework structure is given in Fig. 1, this planning system takes as input the robot and assembly structure setup along with descriptions of the possible robot operating modalities, called mode graphs, and goal descriptions, called assembly task plans. The planning system has three phases: (1) generate a multi-modal roadmap for each robot that describes its feasible movements and interactions with objects, and annotate colliding situations among different roadmaps; (2) optimally assign assembly tasks to robots based on their roadmaps by using Mixed Integer-Linear Programs (MILPs), in which we temporarily ignore inter-robot collisions but respect critical manipulable object collisions; (3) deploy a priority-based, decoupled multi-agent search on the collision-annotated roadmaps to efficiently generate collision-free robot trajectories that fulfill the assembly task plan and task assignments, which is biased towards plans with shorter makespan.

To demonstrate our planning system's ability to automatically generate high-quality assembly solutions for complex assembly structures with multiple robot arms, we test on several challenging simulated assembly domains involving Lego bricks, bars, plates, and irregular-shaped blocks with grippers or suction plates as end-effectors for up to 23 objects.

II. Related Work

Early attempts in Multi-Modal Motion Planning (MMMP) solve multi-step manipulation problems by planning across the configuration spaces of manipulation modes [1], [2]. Recent

This work was supported by Kawasaki Heavy Industry, Ltd (KHI) under grant number 030118-00001. This article solely reflects the opinions and conclusions of its authors and not KHI or any other Kawasaki entity.

[1]Massachusetts Institute of Technology; [2]University of Southern California; [3]University of Illinois Urbana-Champaign; [4]NVIDIA; * indicates equal contributions; Email: jkchen@csail.mit.edu

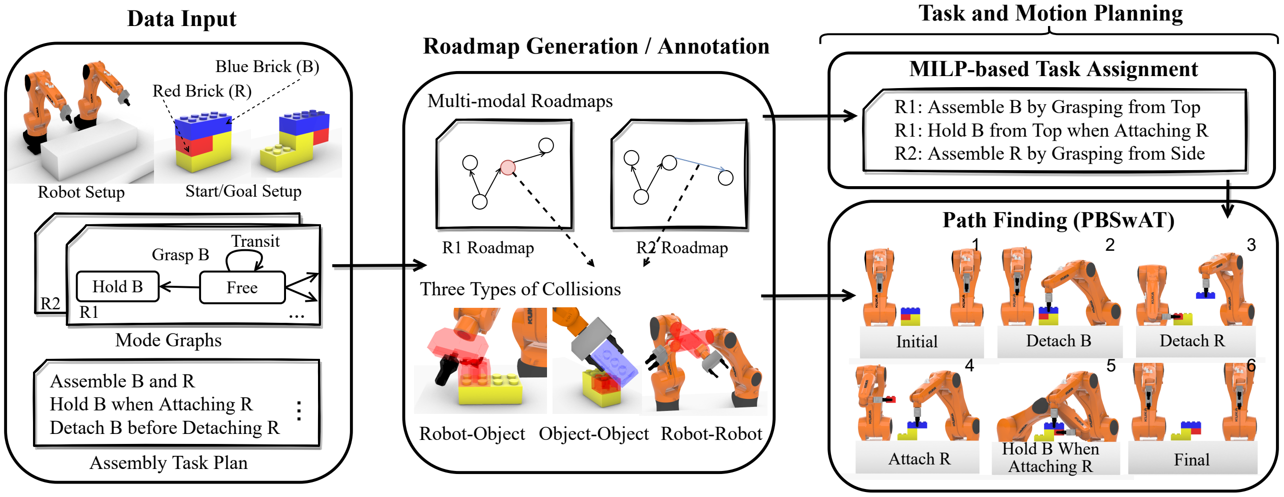


Fig. 1: Phases of the proposed multi-robot task and motion planning algorithm on a two-robot Lego assembly problem.

works in Task and Motion Planning (TAMP) further bridge symbolic reasoning about actions that achieve desired goals and geometric reasoning in search of a collision-free robotic motions [3]. Some work also extends theses frameworks to plan vehicle teams for multiple goals expressed as formal temporal logic [4]. Although capable of modeling and solving multi-arm assembly problems [5], general MMMP and TAMP methods that do not take advantage of the factored nature of multi-robot systems are inefficient to plan for robot teams. From these works, we mainly borrow two ideas: (1) search solutions on the sampled multi-modal roadmaps that describe the robot movements similar to MMMP; and (2) search a plan skeleton candidate and instantiate it to feasible, collision-free paths similar to TAMP.

To tackle the challenges in the multi-arm assembly setting, several other planning systems are proposed under the MMMP or TAMP framework to plan robot arms for assembling furniture [6], [7], construction architecture [8], or LEGO bricks [9], or for general rearrangement problems [10]. In particular, by using an effective factoring approach that plans limited-horizon sub-problems in sequence, [8] is capable of planning feasible solutions for up to 12 robot arms with mobile bases to assemble 100 parts in a relatively open workspace. In contrast, our work is contextualized in planning for fixed-based robots working in a crowded workspace, in which finding collision-free paths is very challenging. In this setting, we also leverage factoring but in a different spirit: while the tasks are optimally assigned in a centralized way, our approach searches collision-free paths in a decoupled manner. In the similar setting of crowded-space assembly planning, [9] plans for three fixed-base robot arms to assemble up to 32 bricks, but their output plans are not guaranteed to be collision-free and thus not safe.

Multi-Agent Path Finding (MAPF) aims at navigating a group of agents, such as vehicles or drones, to reach specified goals without colliding. MAPF algorithms operate on a graph where each agent occupies exactly one vertex and can only move to adjacent vertices at each discretized time step. Recent work in this field significantly improved the scalability of the algorithms [11] and generalized grid-world planning to incorporate task assignment [12], [13]. Conflict-based search originates from MAPF and has been successfully applied to perform multi-arm motion planning [14]. Inspired by the Priority-Based Search (PBS) approach [15], our planning system explores task priorities to generate collision-free paths. In addition to being able to plan for multiple tasks, our system solves a path finding problem on general roadmaps with annotated collisions and continuous traversal time compared to classical MAPF.

III. Problem Formulation

A multi-robot assembly planning problem is defined as the problem of planning configuration trajectories for a team of robots $A = \{a_1, .., a_N\}$ from their start configurations to manipulate a set of objects $O = \{o_1, ..o_M\}$ such that (1) the robots assemble all the objects with respect to the assembly requirements and finish at their goal configurations, and (2) robots, objects, and obstacles do not collide. We do not assume homogeneous robot teams, and each robot is described by an URDF published by its manufacturer, which includes the information such as shapes for collision checking, joint ranges (i.e., configuration space), and maximum joint velocities. We consider first-order dynamics: each configuration trajectory is a time-stamped path $(t_0, c_0), .., (t_K, c_K)$, in which (t_k, c_k) is a time-configuration pair, and $(c_k - c_{k-1})/(t_k - t_{k-1})$ does not exceed the maximum joint velocities. We use makespan (i.e., plan execution time) as our optimization criteria.

In our example in Fig. 1, B and R must be detached from the base and then attached at a new location. During the process, B needs to be held while R is being attached. To describe such assembly requirements, we use mode graphs to qualitatively describe the procedures of the robots manipulating the objects. We use an assembly task plan that leverages this mode graph representation to describe the assembly requirements.

A mode graph for a robot $a_n \in A$ is a directed graph $\langle \Sigma_n, \mathcal{T}_n \rangle$: nodes Σ_n are a set of modes that specify qualitative relations between a_n and objects, each of which represents a set of robot configurations and the manipulated object states,

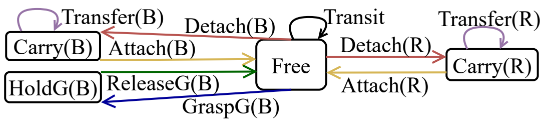


Fig. 2: Mode graph.

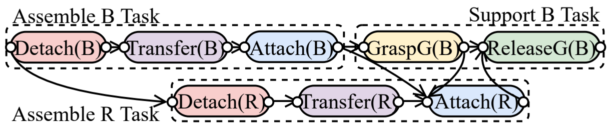


Fig. 3: Assembly task plan.

and directed edges $\mathcal{T}_n$ between nodes are motion primitives, each of which represents a set of moving or manipulating trajectories. A robot trajectory along with the manipulated object states can be qualitatively described as a time-stamped, interleaving sequence of modes and primitives. Fig. 2 shows the mode graph of a robot manipulating B and R in our example: the modes classify the states as free hand (Free), carry an object (Carry), and hold an object at its goal to be stable (HoldG); the primitives classify the trajectories as move with free hand (Transit), transfer an object (Transfer), detach an object from its base or attach it at its target (Detach, Attach), and grasp or release an object at its goal (GraspG, ReleaseG).

An assembly task plan is defined as $\langle \mathrm{T}, \mathrm{P} \rangle$: T is a set of tasks, where each task $\mathrm{T}_p = (\tau_{p,1}, .., \tau_{p,K_p}) \in \mathrm{T}$ is a sequence of motion primitives that must be assigned to and achieved by a unique robot, and P is a set of precedence constraints, and each constraint $(a, b) \in P$, where a and b are in the form of $(\tau, \vdash)$ or $(\tau, \dashv)$, represents the start ($\vdash$) or end ($\dashv$) times of τ. Note that these precedence constraints only specify necessary partial orderings instead of a total ordering. Fig. 3 shows an example assembly task plan. There are three tasks: the first two are assembly tasks with primitives [Detach(B), Transfer(B), Attach(B)] and [Detach(R), Transfer(R), Attach(R)], and the other one is a support task [GraspG(B), ReleaseG(B)]. There are three types of precedence constraints in the example: (1) the precedence constraints between subsequent modes or primitives in the same task (e.g. Detach(B) precedes Transfer(B)); (2) Detach(B) precedes Detach(R), Attach(B) precedes Attach(R), and Attach(B) precedes GraspG(B) given the assembly structure; (3) Attach(R) precedes Grasp(B) and succeeds ReleaseG(B) given the support requirement.

In this paper, we assume: (1) each object can only be manipulated by one robot at each time, and thus lifting or handover among robots are not supported; (2) the assembly process is monotonic and thus objects are not put at intermediate places for regrasping; (3) the start and goal poses of objects as well as the start and goal configurations of robots are known.

IV. Roadmap Generation

Given a robot a_n and its mode graph $(\Sigma_n, \mathcal{T}_n)$, we generate a multi-modal roadmap $G_n = (\mathrm{V}_n, \mathrm{E}_n)$ to describe its feasible movements and interaction with the objects. Vertices V_n are a set of configurations, and each vertex $v \in \mathrm{V}_n$ is labeled with a mode $v.\sigma$ and, when relevant, a grasp pose $v.\mathrm{p}$ that describes the relative transformation between the end-effector and the manipulated object. $v.\mathrm{p} = \emptyset$ if $v.\sigma =$ Free. Directed edges E_n are a set of configuration trajectories, and each edge $e \in \mathrm{E}_n$ from vertex $e.\mathrm{s}$ to vertex $e.\mathrm{e}$ is labelled with a minimal traversal time $e.\mathrm{w}$ of this configuration trajectory given the maximum joint velocities, a primitive $e.\tau$ and a grasp pose $e.\mathrm{p}$. Let $e.\mathrm{p} = \emptyset$ if $e.\tau =$ Transit. These vertices and edges are collision-free with the robot itself and static obstacles. Fig. 4 shows an example multi-modal roadmap.

We sample the multi-modal roadmap similar to the general MMMP sampling method in [2], which iteratively samples in the mode configuration spaces and their intersections. In our mode graph, the primitives can be classified as mode-changing primitives (e.g., Detach, Attach, GraspG, ReleaseG) and mode-preserving primitives (e.g., Transit, Transfer). As the transition space in assembly problems is structured, we take a task-aware sampling method. The edges and vertices are sampled as follows: (1) we first use manipulation skill simulators to sample a diverse set of trajectories as edges called *mode-changing edges* (i.e., the red, yellow, blue, and green edges in Fig. 4) for mode-changing motion primitives, and their starts and ends are *milestones* of the corresponding modes (e.g., the solid-line circles); (2) then we sample edges and vertices for the mode-preserving primitives and the pointed modes, which also connect the previously sampled milestones. These edges and vertices compose a *single-mode roadmap* of a mode-preserving primitive and its pointed mode (e.g., the black lines and the dashed circles). We have two ways to sample such single-mode roadmaps: (2a) we generate single-mode roadmaps for the other primitives and modes (e.g., Transit-Free roadmap and Transfer-Carry roadmap) by using roadmap spanners [16], which are connected to the milestones; (2b) we can also generate a single-mode roadmap by using RRT-connect [17] to find paths (i.e., an interleaving sequence of edges and vertices) between milestones as *highways*. Vertices in (2a) and (2b) can be connected together via *connection edges* to enhance connectivity. Because roadmap vertices also differ in grasp poses, a single-mode roadmap can have multiple disconnected parts under different relative grasp poses. For example, grasping B from the top or side results in two disjoint components in Carry(B)), which is reflected in Fig.4.

In a multi-modal roadmap, mode-changing edges (step 1) and highways (step 2b) are enough to capture the fastest paths for a robot to complete tasks while the spanned vertices and edges (step 2a) can serve as alternatives when the highways are blocked by other robots during a time window. Thus, to reduce the roadmap size, we sample and add these edge differently for task assignment and path finding. In task assignment, since we only consider inter-robot collisions of mode-changing edges, we use a multi-modal roadmap only consisting of mode-changing edges and highways. Then, when the tasks are assigned, we use a multi-modal roadmap consisting of all the spanned vertices and edges and the highways that are related to the assigned tasks, which are connected via corresponding connection edges. As each object has a unique Carry-Transfer roadmap, the numbers of spanned vertices and edges increase

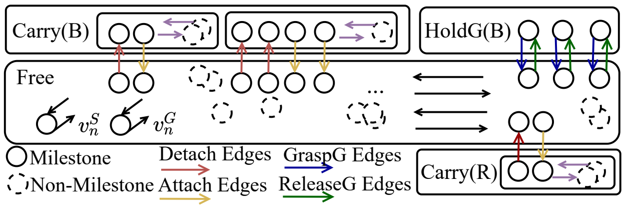


Fig. 4: Multi-modal roadmap (most edges in single-mode roadmaps such as Transit-Free and Transfer-Carry are omitted).

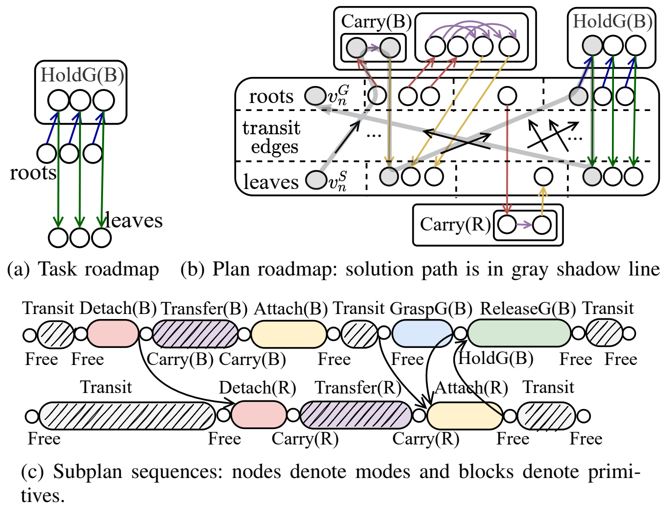
linearly with the object number. Thus, to further reduce the generation time, we cache the arm configurations and collision checking information in the Transit-Free roadmap and reuse them for spanning other single-mode roadmaps in the same multi-modal roadmap. As a result, a large portion of spanned components share the same arm configurations.

A. *Collision Annotation*

Although a robot and its manipulated object are guaranteed to be collision-free with the obstacles when traversing through its roadmap, they also need to avoid colliding with other robots and objects to be safe. We adopt the idea of annotated collisions Π in [18] to characterize pairwise collisions between robot and robot, robot and object, or object and object in our assembly problem. Each annotated collision $\pi \in \Pi$ is a pair of conditions, where each condition denotes an area swept by a robot or an object in the workspace, and the annotated collision implies that the two areas overlap. Specifically, we have two conditions types: (1) edge condition (a_n, e) and vertex condition (a_n, v) represent the swept area of robot a_n and its manipulated object when traversing edge $e \in \mathrm{E}_n$ or waiting at vertex $v \in \mathrm{V}_n$, respectively; (2) object condition $(o_m, \bot)$ or $(o_m, \top)$ represent the area occupied by object o_m being at its start ($\bot$) or goal position ($\top$), respectively. With all the roadmaps, we collect the conditions for all the robots and objects. Then, we do pairwise collision checks between them to record the colliding pairs as annotated collisions. We say there is a collision when a pair of annotated conditions both hold true at the same time. As a large portion of vertices and edges in roadmaps share the same arm configurations, they sweep the same area and the collisions between them and others are only checked once.

V. Task Assignment

The task assignment module is to generate an optimal plan in which each robot takes a sequence of tasks such that all assembly tasks are assigned with respect to the assembly requirements and roadmap connectivity. The optimizing criteria is minimizing the makespan. The task assignment problem at this stage is a relaxation of the original problem since we ignore potential collisions when robots traverse through non-milestones. Fully collision-free paths will be generated by our multi-task multi-agent path finding algorithm by refining a set of partially ordered subplans extracted from our task assignment solution (Section VI). This problem can be treated as an extension of Vehicle Routing Problems with Time Windows with exclusion constraints [19] and formulated as a Mixed Integer-Linear Program (MILP) [12].



(a) Task roadmap (b) Plan roadmap: solution path is in gray shadow line

(c) Subplan sequences: nodes denote modes and blocks denote primitives.

Fig. 5: Task roadmap, plan roadmap, and subplan sequences.

1) Task Roadmap: For every task $\mathrm{T}_p = (\tau_{p,1}, .., \tau_{p,K_p}) \in \mathrm{T}$ and every robot a_n, we extract *task roadmap* $\mathrm{G}^{\mathrm{T}}_{n,p}$, an acyclic directed graph that represents all the paths of a_n to complete task T_p. We construct $\mathrm{G}^{\mathrm{T}}_{n,p}$ by (1) adding all the milestones and mode-changing edges introduced by primitives T_p to G_n; and (2) in each mode, adding a mode-preserving edge from every milestone that ends a mode-changing edge to every milestone that starts a mode-changing edge with the minimum traversal time as the weight except for mode Free. The added edge is an abstraction of the highways between them and thus has the same primitive. A task roadmap example is in Fig.5(a).

2) Plan Roadmap: Then, given all the tasks T, we construct a *plan roadmap* $\mathrm{G}^{\mathrm{T}}_n$ for every robot a_n to represent all the paths of a_n to complete tasks T. To construct $\mathrm{G}^{\mathrm{T}}_n$, we first compose all the task roadmaps $\otimes_p \mathrm{G}^{\mathrm{T}}_{n,p}$. We find the root vertices of all the task roadmaps along with the robot goal vertex as the plan roots, and the leaf vertices along with the robot start vertex as the plan leaves. Roots and leafs have zero in-degree and out-degree, respectively. Then, we add an edge from every leaf to every root with its minimal traversal time except the vertices belonging to the same task roadmap. The nodes of this plan roadmap are not necessarily the subset of the multi-modal roadmap since some tasks are required to execute more than once. An example of plan roadmap is given in Fig.5(b).

3) MILP Encoding: For every $a_n \in A$ and every edge $e \in \mathrm{G}^{\mathrm{T}}_n.\mathrm{E}$, we have a binary variable $A[e]$ to indicate that a_n traverses edge e and a non-negative real variable $t_n[v]$ to denote the times of a_n arriving $v \in \mathrm{G}^{\mathrm{T}}_n.\mathrm{V}$. For every primitive $\tau_{p,k} \in \mathrm{T}_p$, we use real variables $t_{\mathrm{T}}[\tau_{p,k}, \vdash]$ and $t_{\mathrm{T}}[\tau_{p,k}, \dashv]$ to denote the start and end times of $\tau_{p,k}$, respectively. By slightly abusing the notation, we also use $t_{\mathrm{T}}[e.\mathrm{s}]$ and $t_{\mathrm{T}}[e.\mathrm{e}]$ to denote the same variables as $t_{\mathrm{T}}[\tau_{p,k}, \vdash]$ and $t_{\mathrm{T}}[\tau_{p,k}, \dashv]$ for $e \in \mathrm{G}^{\mathrm{T}}_n.\mathrm{E}$ if e is labeled with $\tau_{p,k}$. We use real variable t to denote the total makespan. For a vertex $v \in \mathrm{G}^{\mathrm{T}}_n$, we denote its incoming edges in $\mathrm{G}^{\mathrm{T}}_n$ as $\mathrm{IN}(v)$ and the outgoing edges as $\mathrm{OUT}(v)$. The implication logic in the MILP model is compiled to linear constraints by using the big-M method [20].

minimize t

$$\textstyle\sum_{e\in\text{IN}(v)}A[e]=\sum_{e\in\text{OUT}(v)}A[e], \qquad \forall v\in\cup_n\text{G}^{\text{T}}_n.\text{V}/\{v^{\text{S}}_n,v^{\text{G}}_n\} \quad (1)$$

$$\textstyle\sum_{e\in\text{OUT}(v)}A[v^{\text{S}}_n]=1,\sum_{e\in\text{IN}(v)}A[v^{\text{G}}_n]=1, \qquad \forall n\in(1..N) \quad (2)$$

$$A[e]\rightarrow(t_n[e.\text{e}]-t_n[e.\text{s}]\geq e.\text{w}), \qquad \forall e\in\cup_n\text{G}^{\text{T}}_n.\text{E} \quad (3)$$

$$\textstyle\sum_{e\in\text{E}}A[e]=1, \qquad \forall\text{T}_{k,p}\in\text{T where E}=\cup_n G^{\text{T}}_{n,p,k}.\text{E} \quad (4)$$

$$A[e]\rightarrow(t_{\text{T}}[e.\text{s}]=t[e.\text{s}]))\wedge(t_{\text{T}}[e.\text{e}]=t[e.\text{e}])), \qquad \forall e\in\text{G}^{\text{T}}_n.\text{E} \quad (5)$$

$$t_{\text{T}}[a]\leq t_{\text{T}}[b], \qquad \forall(a,b)\in\text{P and }(t\geq t_n[\cdot]\wedge t\geq t_{\text{T}}[\cdot]) \quad (6)$$

$$(A[e]\wedge A[e'])\rightarrow(t_n[e.\text{e}]<t_{n'}[e'.\text{s}])\vee(t_{n'}[e'.\text{e}]<t_n[e.\text{s}])),$$
$$\forall((a_n,e),(a_{n'},e'))\in\Pi\text{ and }e\in G_n,e'\in G_{n'} \quad (7)$$

$$A[e]\rightarrow(t_n[e.\text{s}]>t_{\text{T}}[\tau,\vdash]),\ \forall((a_n,e),(o,\perp))\in\Pi,\forall\tau\text{ detaches }o \quad (8)$$

$$A[e]\rightarrow(t_n[e.\text{e}]<t_{\text{T}}[\tau,\dashv]),\ \forall((a_n,e),(o,\top))\in\Pi,\forall\tau\text{ attaches }o \quad (9)$$

While constraints (1-2) ensure every robot a_n traverse through a valid path in its plan roadmap G^{T}_n from its start v^{S}_n to goal v^{G}_n, (3) enforces the arrival time of the vertices on this path to respect the traversal time. Then, (4) constrains each task to be assigned to exactly one robot, and (5) links the start and end times of each task primitive with the arrival times of its assigned robot. (6) enforces these start and end times to satisfy the precedence constraints P provided in the task plan, and makespan t is the upper bound of all the time variables. Thus, constraints (1-6) ensure all the tasks are taken by exactly one robot while optimizing the total execution time.

We also add constraints (7-9) to prevent robots from colliding with objects or each other when taking mode-changing edges, which are often the manipulation action and more likely lead to deadends given the intricate assembly structure. Constraint (7) enforces the mode-changing edges that collide with each other not to happen concurrently, and (8) guarantees such edges that collide with objects at starts or goals are not taken before detaching or after attaching objects, respectively.

4) Extracting Subplan Sequences: Given a MILP solution of each robot as the gray shadow path in Fig.5(b), we extract a *subplan sequence* $\gamma_n=\{g_{n,k}\}_k$ for this robot as shown in Fig. 5(c). Each subplan is a mode or primitive associated with the start and end vertices as assigned in the MILP solution. A subplan $g_{n,k}$ is *planned* if a time-stamped path from its start vertex to its end vertex on roadmap G_n is provided. The subplans can be classified as three types: (1) a mode subplan has identical start and end vertices and thus its path is a single vertex; (2) implicitly, the path of a mode-changing primitive subplan can only traverse its corresponding edge in the MILP solution; and (3) the other primitive subplans need to plan longer paths such as Transit and Transfer primitives as sketched in Fig. 5(c). We extract such subplan sequences for all robots to obtain $\Gamma=\{\gamma_n\}_n$ and add precedence constraints $\mathcal{P}$ between the subplan end times according to the assembly task plan, which together are the goal description of our collision-free pathfinding algorithm introduced in Section VI. Note that this goal description does not specify exact arrival times.

VI. Path Finding with Assigned Tasks

We now introduce our Priority-Based Search algorithm for multi-robot pathfinding with Assigned Tasks, denoted as

Algorithm 1: PBSwAT

```
1  $Root = (paths, C, \prec) \leftarrow (\emptyset, \emptyset, \mathcal{P})$;
2  $S \leftarrow \{Root\}$;
3  while $(N \leftarrow S.pop()) \neq \emptyset$ do
4      while $\mathcal{C}_N = \emptyset$ do
5          if $g_i \leftarrow NextUnplannedSubplan(\prec_N)$ is false then
6              return $N.paths$;
7          if $paths \leftarrow PlanPaths(N, g_i)$ is false then
8              go to Line 3;
9          foreach $p_i \in paths$ do $N.paths[g_i] \leftarrow p_i$;
10         $\mathcal{C}_N \leftarrow UpdateCollisions(N, paths)$;
11     $g_i, g_j \leftarrow$ two subplans involved in the 1st collision in $\mathcal{C}_N$;
12     foreach $(g_k, g_l) \in \{(g_i, g_j), (g_j, g_i)\}$ do
13         $N' \leftarrow (N.paths, \mathcal{C}_N, \prec_N \cup \{g_k \prec g_l\})$ ;
14         if UpdateNode$(N', g_k)$ is true then $S.insert(N')$;
15 return false;
```

PBSwAT, that plans paths for every robot a_n to fulfill its subplan sequence γ_n on its multi-modal roadmap G_n such that the precedence constraints $\mathcal{P}$ are satisfied and the paths are collision-free given annotated collision Π. The idea of PBSwAT is to divide a multi-robot problem into single-robot sub-problems and explore the priorities of planning sub-problems as proposed in Priority-Based Search (PBS) [15]. PBS is a two-level algorithm that plans collision-free paths for multiple robots from their starts to goals in grid graphs (i.e., classical MAPF). As its high level explores the priorities between robots in a Priority Tree (PT) such that lower-priority robots should avoid colliding with higher-priority robots, its low level uses A* to plan single-robot paths optimally in discretized timesteps by reserving the paths of higher-priority robots as moving obstacles. While PBSwAT is also a two-level algorithm similar to PBS, its high level explores the priorities between subplans instead of robots and calls its low level to plan paths for subplans lazily instead of calling it to plan all the paths at the beginning, and its low level plans paths for multiple successive same-robot subplans at once instead of one subplan at a time. For simplicity, in this section, we use g_i, $i=1,\ldots,\sum_{a_n\in A}|\gamma_n|$, instead of $g_{n,k}$ to denote a subplan, where subplans of different robots have different index values i, and p_i to denote its corresponding path.

The high level of PBSwAT (Algorithm 1) performs a depth-first search on the PT. It starts with the root PT node that contains an empty set of paths, an empty set of collisions, and the initial priority orderings $\prec$, which are initialized with respect to the precedence constraints $\mathcal{P}$ to enforce the subplans that must end later to have lower priorities (Line 1). The precedence constraints between subsequent same-robot subplans are trivially included. Then, a stack S is initialized with the root node (Line 2). When expanding PT node N (Line 3), it first plans paths for unplanned subplans one at a time with respect to the priority orderings $\prec_N$ (Lines 4 to 10), i.e., *NextUnplannedSubplan* always returns the unplanned subplan that does not have any unplanned higher-priority subplans (Line 5), until (1) some collisions are found (Line 4), (2) all paths are planned, in which case we return the paths (Line 6), or (3) no paths exist, in which case we prune N (Line 8). *PlanPaths*(N, g_i) returns a set of paths because it plans a path

Algorithm 2: UpdateNode (PT node N, subplan g_i)

```
1  $R \leftarrow \{g_i\}$;            // sort subplans to replan in in order of $\prec_N$
2  while $(g_j \leftarrow R.pop()) \neq \emptyset$ do
3      if $livelock$ occurs then return $false$;
4      if $paths \leftarrow$ PlanPaths$(N, g_j)$ is $false$ then return $false$;
5      foreach $p_k \in paths$ do $N.paths[g_k] \leftarrow p_k$;
6      $R \leftarrow R \cup \{g_l \mid (g_k \prec_N g_l) \wedge (g_k, g_l) \in \mathcal{C}_N, p_k \in paths\} \cup \{g_l \mid$
         $g_k$ precedes $g_l) \wedge (N.paths[g_l].T < p_k.T), p_k \in paths\}$;
7      $R \leftarrow R \setminus \{g_k \mid p_k \in paths\}$;
8  return $true$;
```

for g_i and, if necessary, replans paths for the previous same-robot subplans of g_i. More details of these subroutines will be introduced below. Last, PBSwAT resolves a collision in $\mathcal{C}_N$ in the same way as PBS and replans the paths in each generated child node by calling *UpdateNode* (Lines 11 to 14).

UpdateNode (Algorithm 2) iteratively updates the paths of all the affected lower-priority subplans until all planned paths in $N.paths$: (1) satisfy the precedence constraints, (2) do not collide with any objects, and (3) any two planned paths that have priorities in between are collision-free. It first constructs a priority queue R to store all the subplans to replan, in which subplans are sorted according to $\prec_N$ (Line 1). It then repeatedly calls *PlanPaths* to replan until no more subplans need to be replanned (Line 2), a live lock occurs (Line 3), or a failure is reported by *PlanPaths* (Line 4). A live lock refers to a condition where updating a set of subplans triggers replanning of each other in a loop and leads to infinite replanning. In each iteration, when *PlanPaths* replans paths successfully, PBSwAT updatea $N.paths$ accordingly (Line 5), adds the lower-priority subplans that either violate the precedence constraints due to the updated times of the replanned subplans or collide with the updated paths to R (Line 6), and deletes the subplans that have been replanned in this iteration from R (Line 7).

PlanPaths(N, g_i) plans an optimal path for g_i that (1) avoids collisions with the objects and the paths of higher-priority subplans; and (2) ends after the end time of any subplan that must end earlier than g_i. In the case that the subsequent same-robot subplans of g_i already have paths, *PlanPaths* replans their paths accordingly to avoid disjoining the paths of two subsequent subplans over time. Moreover, if there does not exist a path for g_i or its subsequent subplans, *PlanPaths* backtracks and replans the previous subplan of g_i. This backtracking procedure is repeated until *PlanPaths* successfully finds paths for all the subplans that it has to plan, in which case it returns *true*, or no more previous subplan exists, in which case it returns *false*. In each backtracking iteration with newly added subplan g_j, *PlanPaths* calls recursive SIPP (rSIPP) with g_j, and rSIPP will plan paths for g_j and all its subsequent same-robot subplans that already have paths.

Safe Interval Path Planning (SIPP) [21] is a variant of A* that finds an optimal path with minimal total traversal time that avoids given moving obstacles. We adapt it to rSIPP (Algorithm 3) so that it finds a set of paths for successive subplans that avoid the moving obstacles, i.e., the objects and the higher-priority subplans, and satisfies the precedence constraints. It takes input a Reservation Table (RT) as in SIPP that reserves the time intervals at each vertex that are occupied by the moving obstacles. The unreserved time intervals are called safe intervals. rSIPP searches in the resulting vertex-safe-interval graph to find an optimal path that (1) visits each vertex within a safe interval, (2) does not collide with any moving obstacles when it traverses an edge, and (3) ends no earlier than $T_{\min}$, where $T_{\min}$ is the minimum allowable time to finish this subplan with respect to all the subplans that should precede it (Line 1). Its search procedure is the same as that of SIPP except for the goal test. When rSIPP finds a goal node (Line 4), it extracts the path p from n (Lines 5 to 7), checks whether g_i is the last subplan to replan (Lines 8 to 9), and terminates if so; otherwise, it plans for the subsequent subplan g_j to ensure a path starting from the end of p exists (Lines 10 to 14). Specifically, it checks each reachable safe interval at the end vertex with respect to the RT for g_j, calls rSIPP for each of them, and, if succeed, returns the found paths together with p.

Algorithm 3: rSIPP(PT node N,subplan g_i, time t_0, RT rt_i)

```
1  $T_{\min} \leftarrow \max\{N.paths[g_j].T \mid g_j \in \Gamma$ should precede $g_i\}$;
2  generate root node at $g_i.start$ at time $t_0$ and insert it to $Q$;
3  while $(n \leftarrow Q.pop()) \neq \emptyset$ do
4      if $n.v = g_i.goal \wedge n.I.ub > T_{\min}$ then
5          $p \leftarrow$ extract the path from $n$;
6          if $p.T < T_{\min}$ then
7              Add a wait action till time $T_{\min}$ to the end of $p$;
8          $g_j \leftarrow$ the subsequent subplan of $g_i$;
9          if $g_j$ doesn't exist or $N.paths[g_j] = \emptyset$ then return $\{p\}$;
10         $rt_j \leftarrow$ ReservationTable$(N, g_j)$;
11         foreach $[lb, ub) \in rt_j$.SafeIntervals$[n.v]$ do
12             if $([lb', ub') \leftarrow [lb, ub) \cap [p.T, n.I.ub)) \neq \emptyset$
                 $\wedge (paths \leftarrow rSIPP(N, g_j, lb', rt_j)) \neq \emptyset$ then
13                 Add a wait action till time $lb$ to the end of $p$;
14                 return $\{p\} \cup paths$;
15     expand node $n$ w.r.t. $rt_i$ and insert its child nodes to $Q$;
16 return $\emptyset$;
```

VII. Theoretical Properties

The overall approach is algorithmically incomplete and thus trivially suboptimal as other multi-arm assembly approaches. There are mainly three reasons: (1) the sampling-based method can only achieve asymptotical completeness and optimality similar to the original PRM method; (2) the task are assigned while ignoring some collisions and thus may not have corresponding collision-free paths; (3) PBSwAT is incomplete similar to the original PBS algorithm.

We highlight several designs in our system towards mitigating the incompleteness and suboptimality: (1) we use a combination of sampling methods to generate roadmaps and thus it has higher chance to include shorter paths while being sparse; (2) when assigning tasks, the collisions between robots and collisions between robots and objects when taking mode-changing edges are considered as in MILP constraint 7-9, since these collisions are most likely to prevent the algorithm from finding collision-free paths. Then, this relaxed task assignment problem is solved optimally to provide high-quality subplan sequences; (3) PBSwAT can explore all possible priority orderings and is biased towards high-quality solutions with shorter makespan. Meanwhile, instead of only planning each subplan optimally, rSIPP also tries to optimize the paths back and forth in an non-myopic way.

VIII. Simulation Results

In our implementation of roadmap generation and collision annotation, we leverage PyBullet Planning to generate single-mode roadmaps, check collision, and simulate skill trajectories[1]. We ran the MILP encodings for task assignment by using Gurobi [22]. The PBSwAT algorithm is implemented in C++. We execute most our implementation on a 3.40GHZ 8-Core Intel Core i7-6700 CPU with 36GB RAM and leverage 100 CPUs each with 8G memory on Amazon Web Service (AWS) to parallel compute annotated collisions. In all the domains, we use Kuka KR-6-R900 arms with grippers or suction plates.

The skill samplers searches 24 different grasps for Lego bricks and 8 for other objects to generate collision-free manipulation trajectories. We generate 500 vertices for the Free-and-Transit roadmap by using the k-nearest PRM with $k = 10$. This generation takes roughly 40 minutes in all the domains. The maximum sample number of RRT-connect is set to 3000. The highway vertices are connected to its 20 nearest spanned vertices via connection edges. The maximum edge duration is 0.1s, and longer edges are interpolated to sequences of edges. The robot joint resolution for collision checking is set to 0.01π.

We test our examples on domains with different features as shown in Fig. 6: (a) Lego Bridge: two robots with grippers need to detach 17 Lego bricks from the right and assemble them as a bridge, during which necessary holding actions are provided to keep Lego bricks stable; (2) Puzzle Vault: three robots with suction plates assemble vault with 14 irregular-shaped blocks; (3) Truss Boat: three robots with grippers assemble a bridge with 16 bars in a narrow space; (4) Card House: two robots with a gripper and a suction plate, respectively, cooperate to assemble a construction with 23 plates together, in which our planner should be aware of different robot abilities such as only the robot with the suction plate can place plates at the bottom. In the last three domains, we assume (1) objects are teleported to the top of the construction, which abstracts the operations of some cranes or conveyors delivering the objects, for robots to pick; (2) objects are glued to the structure when they are assembled. All the domains are borrowed from the real-world designs and the assembly task plans are extracted given the design structure. The simulations are provided in the supplementary materials.

The runtime results and statistics of each step (Section IV, V, VI) are reported in TABLE I. For roadmap generation and annotation, we only report the generation time of the roadmaps that are used by PBSwAT and skip the statistics for the roadmap used by task assignment. The latter roadmap of each robot is of a much smaller size and takes less than 20 minutes to generate and annotate. We report the average number of sampled vertices and edges for highways and connection edges over all the robots (#V, #E). We do not include the size of the whole multi-modal roadmap in this section. Because the Transfer-Carry roadmaps share a large portion of same arm configurations as the previously sampled Transit-Free roadmaps, and the highways and connection edges are more representative to the problem sizes of different domains. We also report the runtime to generate this multi-modal roadmap (t_{map}), and the runtime to annotate all the necessary collision pairs (t_{anno}). They also include the time to generate and annotate the shared configurations in Transit-Free roadmaps. As we can see in the table, the time to generate roadmaps is around 100 minutes for each robot and the annotation time can be around 1000 hours. Although we leverage CPU clusters for parallel computation and reduce the total runtime to a couple of hours, this time can be significantly reduced by using (1) lazily roadmap generation and annotation methods [23], [24]; or (2) Voxel-based collision checking by using GPUs [25]. We consider implementing these techniques into our planning systems as important future work.

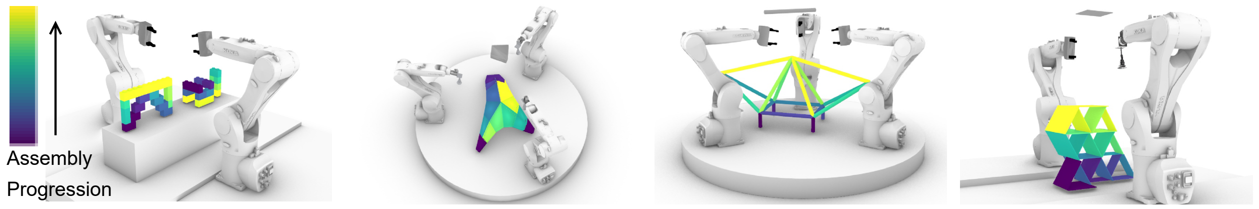
In the MILP-based task assignment, we report the numbers of binary variables (#B), continuous variables (#X), and constraints (#C) along with the runtimes to find the first solution t_T, find the optimal solution $t_T^\dagger$, and exhaust the solution space t_T^*. As we can see in the table, all the MILP encodings feature a relatively small number of continuous variables and can have up to eighteen thousand binary variables. #B is dominated by the number of mode-changing edges and the potential collision between these edges. Thus, we can see Truss Boat has the largest #B since its mode-changing edges are very easy to collide with each other or the objects. Although these MILPs are of huge size, Gurobi can find a feasible solution in two minutes and then an optimal one in a couple of minutes for all the domains, which is due to the relatively small number of continuous variables in all the domains.

In the PBSwAT column, we show the number of all the subplans (#g), the expanded nodes (#N), the calls to rSIPP (#rSIPP), the runtime to find the solutions (t_P), and the ratio of the used annotated collisions over all the annotated collisions (η). As we can see, all the domains can be solved in two minutes with up to 44 node expansions. We also observe that, in crowded domains such as Puzzle Vault and Truss Boat, in which robots are easy to collide, PBSwAT needs to expand more nodes and add more priorities, and η can be up to 22%. However, in the setups where robots are front to front and do not collide often, η can be low as 3%. As we can see, our task assignment and path finding algorithms can quickly find high-quality solutions in a couple of minutes, the overall runtime is dominated by roadmap generation and collision annotation.

IX. Conclusion and Future Work

In this paper, we present a cooperative task and motion planner that jointly plans safe, high-quality assembly plans for multiple robot arms to assemble complex structures. We demonstrate its capabilities in several challenging domains. The future work includes (1) planning more advanced co-operative behaviors such as lifting or handovers; (2) developing a multi-arm executive while considering higher-order dynamics such as accelerations to robustly execute the planned configuration trajectories; (3) improving the implementation of

[1]https://pypi.org/project/pybullet-planning/



(a) Lego Bridge (b) Puzzle Vault (c) Truss Boat (d) Card House

Fig. 6: Problem instances. (a) Lego Bridge: two robots with grippers assemble 17 Lego bricks; (b) Puzzle Vault: three robots with suction plates assemble 14 irregular-shaped blocks; (c) Truss Boat: three robots with grippers assemble 16 bars; (d) Card House: two robots with gripper and suction plates assemble 23 plates.

Domain	Roadmap Generation & Annotation				MILP-based Task Assignment				PBSwAT					T(s)
	#V	#E	t_{map}(min)	t_{anno}(hr)	#B	#X	#C	$t_T/t_T^{\dagger}/t_T^{*}$(s)	#g	#N	#rSIPP	t_P(s)	η	
(a)	8450	13454	87	741	2985	83	95795	40/44/65	166	6	214	66	0.04	85
(b)	4973	8642	103	1289	1148	57	2853	3/10/11	129	29	278	32	0.22	65
(c)	4049	8688	86	1035	18857	88	51224	90/175/496	153	44	221	122	0.18	66
(d)	8188	12818	112	861	4253	141	13157	71/105/821	190	11	221	17	0.03	98

TABLE I: Simulation Results. #V/#E: average vertex/edge number of highways and connections; t_{map}: average roadmap generation time; t_{anno}: roadmap annotation time; #B,#X,#C: number of binary variables, continuous variables, and constraints in the MILP; t_T,$t_T^{\dagger}$,t_T^{*}: runtimes to find a feasible MILP solution, find an optimal solution, and exhaust the solution space; #g: number of subplans; #N: number of expanded nodes; #rSIPP: calls to rSIPP; t_P: PBSwAT runtime; η: ratio of the used annotated collisions; T: solution makespan.

roadmap generation and annotation by using GPU-based collision checking and lazy roadmap generation; and (4) adjusting task assignments online in PBSwAT.

References

[1] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[2] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *The International Journal of Robotics Research*, vol. 29, no. 7, pp. 897–915, 2010.

[3] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated Task and Motion Planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, 2021.

[4] C. K. Verginis and D. V. Dimarogonas, "Motion and cooperative transportation planning for multi-agent systems under temporal logic formulas," *arXiv preprint arXiv:1803.01579*, 2018.

[5] M. Toussaint and M. Lopes, "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017, pp. 4044–4051.

[6] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus, "Ikeabot: An autonomous multi-robot coordinated furniture assembly system," in *2013 IEEE International conference on robotics and automation*. IEEE, 2013, pp. 855–862.

[7] M. Dogar, A. Spielberg, S. Baker, and D. Rus, "Multi-robot grasp planning for sequential assembly operations," *Autonomous Robots*, vol. 43, no. 3, pp. 649–664, 2019.

[8] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, "Long-Horizon Multi-Robot Rearrangement Planning for Construction Assembly," *arXiv preprint arXiv:2106.02489*, 2021.

[9] L. Nägele, A. Hoffmann, A. Schierl, and W. Reif, "Legobot: Automated planning for coordinated multi-robot assembly of lego structures," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Systems*, 2020.

[10] R. Shome and K. E. Bekris, "Synchronized multi-arm rearrangement guided by mode graphs with capacity constraints," in *Algorithmic Foundations of Robotics XIV*, 2021, pp. 243–260.

[11] A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek, "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *Tenth Annual Symposium on Combinatorial Search*, 2017.

[12] K. Brown, O. Peltzer, M. A. Sehr, M. Schwager, and M. J. Kochenderfer, "Optimal sequential task assignment and path finding for multi-agent robotic assembly planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 441–447.

[13] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2018.

[14] 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.

[15] 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, no. 01, 2019, pp. 7643–7650.

[16] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[17] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[18] W. Hönig, J. A. Preiss, T. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 856–869, 2018.

[19] G. Laporte and I. H. Osman, "Routing problems: A bibliography," *Annals of operations research*, vol. 61, no. 1, pp. 227–262, 1995.

[20] I. Griva, S. G. Nash, and A. Sofer, *Linear and nonlinear optimization*. Siam, 2009, vol. 108.

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

[22] I. Gurobi Optimization, "Gurobi optimizer reference manual," *URL http://www. gurobi. com*, 2021.

[23] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 521–528.

[24] S. Li and J. A. Shah, "Safe and efficient high dimensional motion planning in space-time with time parameterized prediction," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5012–5018.

[25] O. S. Lawlor and L. V. Kalée, "A voxel-based parallel collision detection algorithm," in *Proceedings of the 16th international conference on Supercomputing*, 2002, pp. 285–293.