Multi-Robot Motion Planning with Dynamics via Coordinated Sampling-Based Expansion Guided by Multi-Agent Search

Duong Le and Erion Plaku

Abstract—**This paper combines sampling-based motion planning with multi-agent search to efficiently solve challenging multi-robot motion-planning problems with dynamics. This idea has shown promise in prior work which developed a *centralized* approach to expand a motion tree in the composite state space of all the robots along routes obtained by multi-agent search over a discrete abstraction. Still, the centralized expansion imposes a significant bottleneck due to the curse of dimensionality associated with the high-dimensional composite state space.**

To improve efficiency and scalability, we propose a *coordinated* expansion of the motion tree along routes obtained by the multi-agent search. We first develop a single-robot sampling-based approach to closely follow a given route σ_i. The salient aspect of the proposed coordinated expansion is to invoke the route follower one robot at a time, ensuring that robot i follows σ_i while avoiding not only the obstacles but also robots $1, \ldots, i-1$. In the next iteration, the motion tree could be expanded from another state along other routes. This enables the approach to progress rapidly and achieve significant speedups over a centralized approach.

Index Terms—**Motion and Path Planning; Nonholonomic Motion Planning; Multi-Robot Systems**

I. Introduction

MULTI-ROBOT systems provide a viable venue to increase productivity in exploration, inspection, transportation, and other applications. In these settings, robots are often required to move to specific locations while avoiding obstacles and other robots. This is challenging, especially in obstacle-rich environments, since each robot often has to avoid numerous obstacles and pass through narrow passages to reach its destination. Coordination also becomes necessary since it is often not sufficient to plan the motions of the robots independently as the plans of one robot can interfere with the plans of another robot. Moreover, robot motions are governed by the underlying dynamics which impose differential constraints on velocity, acceleration, curvature, and turning radius.

Related work has considered various settings for the multi-robot motion-planning problem. In a discrete setting, i.e., multi-agent pathfinding, each robot is a point with no dynamics that in one step can move to an adjacent vertex in a graph.

Manuscript received: September, 8, 2018; Revised December, 13, 2018; Accepted January, 30, 2019.

This paper was recommended for publication by Editor Nancy Amato upon evaluation of the Associate Editor and Reviewers' comments. This work is supported by NSF ACI 1440587

The authors are with the Department of Electrical Engineering and Computer Science, Catholic University of America, Washington DC, USA {05le,plaku}@cua.edu

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

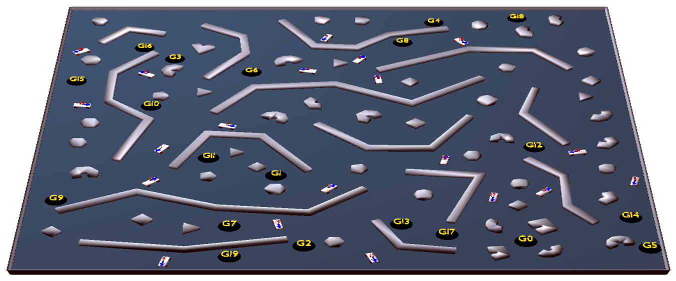

Fig. 1. The objective in multi-robot motion planning is to generate a dynamically-feasible trajectory for each robot i to reach goal $\mathcal{G}_i$ while avoiding collisions with the obstacles and the other robots. Visit goo.gl/ZXNqbd to see videos of solutions. Figure best viewed in color and on screen.

To solve this NP-hard problem [1], decoupled approaches often plan the paths for each agent one at a time, ensuring that agent i avoids conflicts with the plans of agents $1, \ldots, i-1$ [2]–[4]. Decoupled approaches are fast but suboptimal. To provide optimality, centralized approaches based on A* [5]–[7], increasing-cost tree [8], and conflict-based search [9] operate over the composite space of all the agents [10]. M* [11] maintains optimality but postpones conducting the search in the full composite space until necessary, using subdimensional expansion to resolve conflicts. Solvers have also used auctions [12], answer-set programming [13], or rules [14]–[17].

In a continuous setting, sampling-based planners are often used. To account for dynamics, `RRT` [18], `KPIECE` [19], `GUST` [20], and others expand a tree of collision-free and dynamically-feasible motions. `PRM` [21] cannot generally be used since each roadmap edge imposes two boundary value problems. Analytical solutions exist only in limited cases, while numerical ones are slow and leave gaps [22].

For multi-robot problems, sampling-based motion planners can be used in a decoupled or centralized framework [23]. Decoupled approaches plan the motions of the robots one at a time, treating robots $1, \ldots, i-1$ as moving obstacles when planning the motions of robot i. They are fast but not probabilistically complete since the motions planned for robots $1, \ldots, i-1$ could make it impossible for robot i to reach its goal. Centralized approaches treat the robots as one system and operate over the resulting composite state space. They are probabilistically complete, but do not scale well due to the high dimensionality of the composite state space.

To improve scalability, [24] developed a centralized approach that uses multi-agent search over a discrete abstraction to guide sampling-based motion planning. The core loop consist of (i) selecting a state $\langle s_1, \ldots, s_n \rangle$ from the motion tree, (ii) using multi-agent search over the discrete abstraction to compute routes $\langle \sigma_1, \ldots, \sigma_n \rangle$ for each robot to its goal, and (iii) expanding the motion tree in a *centralized* fashion from $\langle s_1, \ldots, s_n \rangle$ toward $\langle \sigma_1(1), \ldots, \sigma_n(1) \rangle$, where $\sigma_i(1)$ denotes the first target in σ_i. The centralized expansion involves applying a controller to $\langle s_1, \ldots, s_n \rangle$ for a number of steps until the targets are reached or a collision occurs.

Although [24] is faster than other approaches, the centralized expansion of the motion tree still imposes a significant bottleneck due to the curse of dimensionality associated with the high-dimensional composite state space. As the number of robots increases, it becomes significantly harder to generate valid trajectories from $\langle s_1, \ldots, s_n \rangle$ to $\langle \sigma_1(1), \ldots, \sigma_n(1) \rangle$, since the centralized expansion moves each robot toward its target one step at a time, stopping when a collision occurs.

Leveraging the idea of using multi-agent search to guide sampling-based motion planning, we develop an approach that utilizes a *coordinated* expansion of a motion tree $\mathcal{T}$ along the routes provided by the multi-agent search. Our approach maintains the abstraction and the overall structure of [24] but replaces the centralized motion-tree expansion with the proposed coordinated expansion. This significantly improves the efficiency and scalability.

To make this possible, we first develop a single-robot sampling-based approach $\textsc{follow}(s_i, \sigma_i, \langle \zeta_1, \ldots, \zeta_{i-1} \rangle)$, which generates a dynamically-feasible trajectory ζ_i that starts at s_i, closely follows σ_i, and avoids the obstacles as well as robots $1, \ldots, i-1$ as they move along $\zeta_1, \ldots, \zeta_{i-1}$. The idea is to cover σ_i with balls and expand a new motion tree (not $\mathcal{T}$) rooted at s_i from one ball to the next.

Using sampling-based motion planning for route following has also been studied in related work, e.g., expanding an RRT along a route obtained by an any-angle pathfinder [25] or using an A* expansion of RRT by defining a heuristic cost for each vertex [26], [27]. These approaches, however, were not developed for multi-robot motion planning, so they do not ensure that the route locations are reached in succession. In multi-robot problems, skipping or reaching the locations out of order could lead to robot-robot conflicts or even prevent the robots from reaching their goals. In contrast, FOLLOW ensures that the route locations are reached in succession.

During the core loop, given a state $\langle s_1, \ldots, s_n \rangle$ and routes $\langle \sigma_1, \ldots, \sigma_n \rangle$ obtained by the multi-agent search, the salient aspect of the proposed coordinated expansion is to invoke

$$\zeta_{\pi_i} \leftarrow \textsc{follow}(s_{\pi_i}, \sigma_{\pi_i}, \langle \zeta_{\pi_1}, \ldots, \zeta_{\pi_{i-1}} \rangle) \tag{1}$$

in succession where π denotes a random permutation of $\{1, \ldots, n\}$. In this way, motions planned for robot π_i closely follow σ_{π_i} and avoid collisions with the motions planned for robots $\pi_1, \ldots, \pi_{i-1}$. The motion tree $\mathcal{T}$ is expanded with $\langle \zeta_1, \ldots, \zeta_n \rangle$ as the branch from $\langle s_1, \ldots, s_n \rangle$.

We do not expect to solve the multi-robot motion-planning problem in one call to the coordinated expansion. In fact, many times FOLLOW will not generate a trajectory ζ_i that reaches the end of σ_i (which is the goal for robot i), but it could still make progress and reach $\sigma_i(k)$, where $k \geq 1$. To promote expansions from states close to the goal, the approach uses heuristic costs based on route lengths as obtained by the multi-agent search.

II. Problem Formulation

Each robot model $\mathcal{M}_i = \langle \mathcal{P}_i, \mathcal{S}_i, \mathcal{U}_i, f_i \rangle$ is defined by its geometric shape $\mathcal{P}_i$, state space $\mathcal{S}_i$, control space $\mathcal{U}_i$, and dynamics f_i expressed as a set of differential equations

$$\dot{s} = f_i(s, u), s \in \mathcal{S}_i, u \in \mathcal{U}_i. \tag{2}$$

As an example, the state of the vehicle model used in the experiments is defined as $\langle x, y, \theta, \psi, v \rangle$ in terms of the position (x, y), orientation θ, steering angle ψ, and velocity v. The vehicle is controlled by setting the acceleration *acc* and steering rate ω. The motion equations are defined as

$$\dot{x} = v\cos(\theta)\cos(\psi), \dot{y} = v\sin(\theta)\cos(\psi), \tag{3}$$

$$\dot{\theta} = v\sin(\psi)/L, \dot{v} = \mathit{acc}, \dot{\psi} = \omega, \tag{4}$$

where L is the distance from the back to the front wheels.

The new state $s_{\mathit{new}} \in \mathcal{S}_i$ obtained by applying a control $a \in \mathcal{U}_i$ to a state $s \in \mathcal{S}_i$ is computed by a function

$$s_{\mathit{new}} \leftarrow \textsc{simulate}(s, a, f_i, dt), \tag{5}$$

which numerically integrates f_i for one time step dt. Starting from $s \in \mathcal{S}_i$ and applying controls $\langle u_1, \ldots, u_{\ell-1} \rangle$ in succession gives rise to a dynamically-feasible trajectory $\zeta : \{1, \ldots, \ell\} \to \mathcal{S}_i$, where $\zeta(1) \leftarrow s$ and $\forall j \in \{2, \ldots, \ell\}$:

$$\zeta(j) \leftarrow \textsc{simulate}(\zeta(j-1), u_{j-1}, f_i, dt). \tag{6}$$

The multi-robot motion-planning problem is defined by

$$\langle \mathcal{W}, \mathcal{O}, \mathcal{G}_1, \ldots, \mathcal{G}_n, \mathcal{M}_1, \ldots, \mathcal{M}_n, s_1^{\mathit{init}}, \ldots, s_n^{\mathit{init}} \rangle. \tag{7}$$

The world $\mathcal{W}$ contains obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$ and goals $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$, which are all disjoint. The objective is to compute trajectories $\zeta_1, \ldots, \zeta_n$, one for each robot, so that $\zeta_i : \{1, \ldots, \ell\} \to \mathcal{S}_i$ starts at the initial state $s_i^{\mathit{init}} \in \mathcal{S}_i$, reaches $\mathcal{G}_i$, and avoids the obstacles and the other robots.

III. Framework

To facilitate presentation, we first summarize the framework developed in [24] for using multi-agent search to guide sampling-based motion planning. The next section describes our proposed approach, which significantly improves the efficiency and scalability by replacing the centralized motion-tree expansion with a coordinated expansion.

A. Discrete Abstraction and Multi-Agent Search

The framework first constructs a discrete abstraction which provides a simplified problem representation that ignores the robot dynamics. Specifically, drawing from PRM [21], a roadmap $\mathcal{R}_i = (V_{\mathcal{R}_i}, E_{\mathcal{R}_i}, \textsc{cost}_{\mathcal{R}_i})$ is built for each robot over its configuration space $\mathcal{C}_i$ to capture the connectivity of the environment by sampling collision-free configurations and connecting neighboring configurations with collision-free paths whenever possible, as shown in Fig. 2. Each robot configuration $c \in \mathcal{C}_i$ is defined by specifying the position and orientation, i.e., $c = \langle x, y, \theta \rangle$.

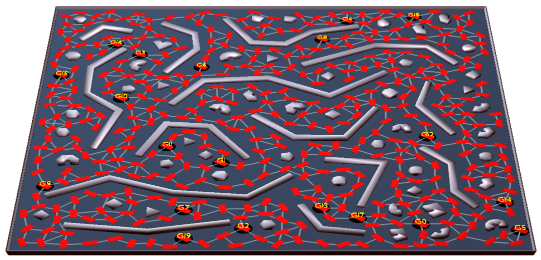

Fig. 2. An example of a roadmap.

The framework avoids explicitly constructing the roadmap over the composite configuration space $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$. Doing so would be impractical due to the high-dimensionality of $\mathcal{C}$. Instead, the framework combines the individual roadmaps $\mathcal{R}_1, \ldots, \mathcal{R}_n$ implicitly by computing the out-going edges on the fly, i.e., given $\langle c_1, \ldots, c_n \rangle \in V_{\mathcal{R}_1} \times \ldots \times V_{\mathcal{R}_n}$ compute

$$\textsc{OutEdges}(\langle c_1, \ldots, c_n \rangle) = \{\langle c'_1, \ldots, c'_n \rangle : \quad (8)$$

$$(c_1, c'_1) \in E_{\mathcal{R}_1}, \ldots, (c_n, c'_n) \in E_{\mathcal{R}_n}\}. \quad (9)$$

$\textsc{MultiAgentSearch}(\mathcal{R}_1, \ldots, \mathcal{R}_n, c_1, \ldots, c_n, \mathcal{G}_1, \ldots, \mathcal{G}_n)$ uses the roadmaps and the $\textsc{OutEdges}$ function to compute non-conflicting routes $\sigma_1, \ldots, \sigma_n$ for each robot such that σ_i is over $\mathcal{R}_i$, starts at c_i, and ends at a roadmap vertex associated with the goal $\mathcal{G}_i$. Note that roadmap $\mathcal{R}_i$ guarantees that robot $\mathcal{M}_i$ will not collide with the obstacles as it moves from one roadmap vertex to the next. Thus, the multi-agent search has to avoid only robot-robot conflicts.

B. Motion Tree and Partition into Equivalence Classes

A motion tree $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is rooted at the initial state $\langle s_1^{init}, \ldots, s_n^{init} \rangle$ and is incrementally expanded in the composite state space $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ by adding new vertices and edges, as shown in Fig. 3. Each vertex $v \in V_\mathcal{T}$ is associated with a collision-free composite state, denoted by $v.\langle s_1, \ldots, s_n \rangle$. Each edge $(v, v') \in E_\mathcal{T}$ is labeled with the controls $\langle u_1, \ldots, u_n \rangle$ applied to $v.\langle s_1, \ldots, s_n \rangle$ to generate $v'.\langle s_1, \ldots, s_n \rangle$, i.e., $\forall i \in \{1, \ldots, n\} : v'.s_i \leftarrow \textsc{simulate}(v.s_i, u_i, f_i, dt)$. Note that, as in other sampling-based planners, the time step dt discretizes the continuous motion from $v.\langle s_1, \ldots, s_n \rangle$ to $v'.\langle s_1, \ldots, s_n \rangle$.

The motion tree is expanded until a vertex v_{new} is added where each robot has reached its goal, i.e., $\forall i \in \{1, \ldots, n\} : v_{new}.s_i \in \mathcal{G}_i$. The path $\langle v_1, \ldots, v_\ell \rangle$ ($v_1 = v_{init}$ and $v_\ell = v_{new}$) from the root of $\mathcal{T}$ to v_{new} is retrieved and the solution trajectory for each robot i is set to $\zeta_i = \langle v_1.s_i, \ldots, v_\ell.s_i \rangle$.

The abstraction is used to partition the motion tree into equivalence classes. The partition allows the framework to incorporate multi-agent search to guide the motion-tree expansion. To construct the partition, a state $s \in \mathcal{S}_i$ is mapped to the nearest configuration in the roadmap $\mathcal{R}_i$, i.e.,.

$$\textsc{map}(\mathcal{R}_i, s) = \operatorname*{argmin}_{c \in V_{\mathcal{R}_i}} ||\textsc{position}(c) - \textsc{position}(s)||_2. \quad (10)$$

A composite state $\langle s_1, \ldots, s_n \rangle \in \mathcal{S}$ is mapped to $\langle c_1, \ldots, c_n \rangle$, where $c_i = \textsc{map}(\mathcal{R}_i, s_i)$. Inversely, the equivalence class $\Gamma_{\langle c_1, \ldots, c_n \rangle}$ groups together all the states in $\mathcal{T}$ that map to $\langle c_1, \ldots, c_n \rangle$, i.e.,

$$\Gamma_{\langle c_1, \ldots, c_n \rangle} = \{v : v \in V_\mathcal{T} \wedge \forall i : c_i = \textsc{map}(\mathcal{R}_i, v.s_i)\}. \quad (11)$$

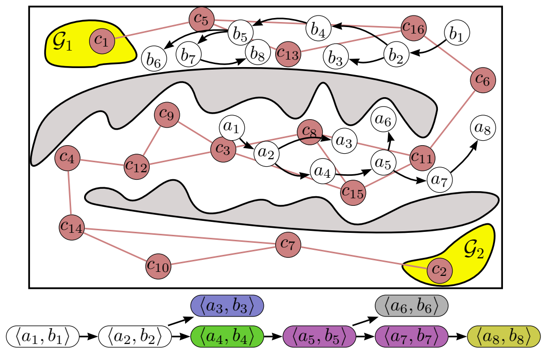

Fig. 3. Illustration of a roadmap, motion tree, and partition of the motion tree into equivalence classes. The roadmap configurations are labeled as $c_1, \ldots, c_{16}$. Assuming that robots 1 and 2 have the same shape, robot 2 can use the same roadmap, so $\mathcal{R}_2$ is set to $\mathcal{R}_1$. The composite roadmap $\mathcal{R}_1 \times \mathcal{R}_2$ is computed implicitly using the function $\textsc{OutEdges}$. As an example, $\textsc{OutEdges}(\langle c_{13}, c_4 \rangle) = \{\langle c_5, c_{12} \rangle, \langle c_5, c_{14} \rangle, \langle c_{16}, c_{12} \rangle, \langle c_{16}, c_{14} \rangle\}$. The graph at the bottom shows the parent-child relations in the motion tree with vertices labeled as $\langle a_1, b_1 \rangle, \ldots, \langle a_8, b_8 \rangle$. As an example, the edge $\langle a_1, b_1 \rangle \rightarrow \langle a_2, b_2 \rangle$ denotes the motion of robot 1 from a_1 to a_2 and of robot 2 from b_1 to b_2. The robot motions are illustrated with the curved arrows in the top figure. The motion tree is partitioned into these equivalence classes: $\Gamma_{\langle c_3, c_{16} \rangle} = \{\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle\}$, $\Gamma_{\langle c_8, c_{13} \rangle} = \{\langle a_3, b_3 \rangle\}$, $\Gamma_{\langle c_{15}, c_{13} \rangle} = \{\langle a_4, b_4 \rangle\}$, $\Gamma_{\langle c_{11}, c_5 \rangle} = \{\langle a_5, b_5 \rangle, \langle a_7, b_7 \rangle\}$, $\Gamma_{\langle c_{11}, c_1 \rangle} = \{\langle a_6, b_6 \rangle\}$, $\Gamma_{\langle c_6, c_{13} \rangle} = \{\langle a_8, b_8 \rangle\}$. Vertices that belong to the same equivalence class are shown with the same color.

Algorithm 1 Framework for using multi-agent search to guide sampling-based motion planning [24].

Input: $\langle \mathcal{W}, \mathcal{O}, \mathcal{G}_1, \ldots, \mathcal{G}_n, \mathcal{M}_1, \ldots, \mathcal{M}_n, s_1^{init}, \ldots, s_n^{init} \rangle$; time step dt; configuration spaces $\mathcal{C}_1, \ldots, \mathcal{C}_n$; runtime limit t_{max}
Output: collision-free and dynamically-feasible trajectories for each robot from the initial state to its goal; or $\perp$ if no solution is found

```
1: for $i = 1 \ldots n$ do
2:   $\mathcal{R}_i = (V_{\mathcal{R}_i}, E_{\mathcal{R}_i}, \textsc{cost}_{\mathcal{R}_i}) \leftarrow \textsc{roadmap}(\mathcal{C}_i, \mathcal{P}_i, s_i^{init}, \mathcal{G}_i)$
3: $\langle \mathcal{T}, \Gamma \rangle \leftarrow \textsc{InitializeMotionTreeAndEQC}(s_1^{init}, \ldots, s_n^{init})$
4: while $\textsc{time}() < t_{max}$ do
5:   $\Gamma_{\langle c_1, \ldots, c_n \rangle} \leftarrow \textsc{SelectEQC}(\Gamma)$
6:   $\langle \sigma_1, \ldots, \sigma_n \rangle \leftarrow \textsc{MultiAgentSearch}(\mathcal{R}_1, \ldots, \mathcal{R}_n,$
         $c_1, \ldots, c_n, \mathcal{G}_1, \ldots, \mathcal{G}_n)$
7:   $v \leftarrow \textsc{SelectMotionTreeVertexFromEQC}(\Gamma_{\langle c_1, \ldots, c_n \rangle})$
8:   $\langle \zeta_1, \ldots, \zeta_n \rangle \leftarrow \textsc{CentralizedExpansion}(\mathcal{T}, v, \sigma_1, \ldots, \sigma_n)$
9:   for $j = 1, \ldots, \min\{|\zeta_1|, \ldots, |\zeta_n|\}$ do
10:    $v_{new} \leftarrow \textsc{AddVertex}(\mathcal{T}, \langle \zeta_1(j), \ldots, \zeta_n(j) \rangle)$
11:    $\textsc{AddEdge}(\mathcal{T}, v, v_{new})$; $v \leftarrow v_{new}$
12:    if $\forall 1 \leq k \leq n : \zeta_k(j) \in \mathcal{G}_k$ then return $\textsc{trajs}(\mathcal{T}, v_{new})$
13:    $\textsc{UpdateEQC}(\Gamma, v_{new})$
14: return $\perp$
```

$\mathcal{T}$ is then partitioned into a set of equivalence classes

$$\Gamma = \bigcup_{v \in V_\mathcal{T}} \{\Gamma_{\langle \textsc{map}(\mathcal{R}_1, v.s_1), \ldots, \textsc{map}(\mathcal{R}_n, v.s_n) \rangle}\}. \quad (12)$$

Fig. 3 provides an illustration.

C. Using Multi-Agent Search to Guide the Motion-Tree Expansion

Pseudocode is shown in Alg. 1. After constructing the roadmaps $\mathcal{R}_1, \ldots, \mathcal{R}_n$ and rooting the motion tree $\mathcal{T}$ at the initial state, the framework proceeds iteratively until a solution is found or a runtime limit is reached. At each iteration, an equivalence class $\Gamma_{\langle c_1, \ldots, c_n \rangle}$ is selected from Γ (Alg. 1:5). Multi-agent search is then used to search the roadmaps and obtain non-conflicting routes $\langle \sigma_1, \ldots, \sigma_n \rangle$, where σ_i starts at c_i and reaches the goal $\mathcal{G}_i$ (Alg. 1:6). To save time, the routes are stored with $\Gamma_{\langle c_1,\ldots,c_n \rangle}$ and retrieved when the multi-agent search is invoked again for $\Gamma_{\langle c_1,\ldots,c_n \rangle}$. During the selection, priority is given to equivalence classes associated with short routes to promote rapid expansions toward the goal. After selecting $\Gamma_{\langle c_1,\ldots,c_n \rangle}$, the objective is to expand the motion tree from a vertex selected among those associated with $\Gamma_{\langle c_1,\ldots,c_n \rangle}$ (Alg. 1:7) along $\sigma_1, \ldots, \sigma_n$.

A centralized approach (Alg. 1:8) was used in [24] to expand $\mathcal{T}$ from $v.s_1, \ldots, s_n$ toward $\langle \sigma_1(1), \ldots, \sigma_n(1) \rangle$, where $\sigma_i(1)$ denotes the first configuration in σ_i. The centralized expansion applies a controller to $\langle s_1, \ldots, s_n \rangle$ for a number of steps until the targets are reached or a collision occurs. The intermediate states generated during the expansion are added to $\mathcal{T}$ (Alg. 1:10-11). The equivalence classes are also updated accordingly (Alg. 1:13).

IV. Method

Our approach maintains the discrete abstraction and the overall structure of the framework developed in [24] but replaces the centralized motion-tree expansion (Alg. 1:8) with the proposed coordinated expansion.

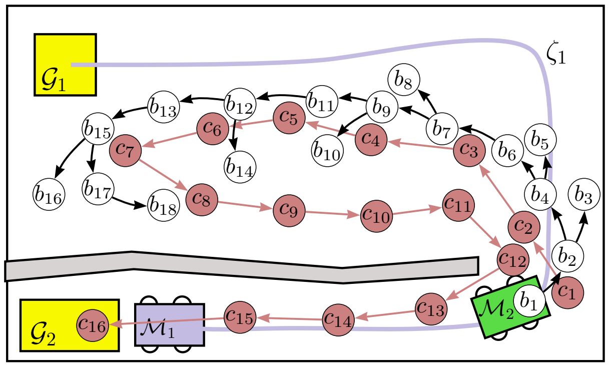
$\mathcal{G}_1$
ζ_1
$\mathcal{G}_2$
$\mathcal{M}_1$
$\mathcal{M}_2$
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15} b_{16} b_{17} b_{18}
c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8 c_9 c_{10} c_{11} c_{12} c_{13} c_{14} c_{15} c_{16}

Fig. 4. Illustration of FOLLOW. Given the trajectory ζ_1 of robot $\mathcal{M}_1$ and the route $\sigma_2 = \langle c_1, \ldots, c_{16} \rangle$ for robot $\mathcal{M}_2$, FOLLOW expands a motion tree Λ along σ_2, seeking to reach $c_1, \ldots, c_{16}$ in succession. Not maintaining the order could prevent the robot from reaching its goal, e.g., skipping $c_1, \ldots, c_{12}$ and focusing only on $c_{13}, \ldots, c_{16}$ could prevent $\mathcal{M}_2$ from reaching $\mathcal{G}_2$ since ζ_1 blocks the way. By maintaining the order, $\mathcal{M}_2$ gets out and only enters the narrow passage when $\mathcal{M}_1$ leaves. FOLLOW returns the trajectory $\zeta_2 = \langle b_1, b_2, b_4, b_6, b_7, b_9, b_{11}, b_{12}, b_{13}, b_{15}, b_{17}, b_{18} \rangle$ where $c_1, \ldots, c_8$ are reached (within d_{reach}) by $b_2, b_4, b_7, b_9, b_{11}, b_{12}, b_{15}, b_{18}$, respectively. Since a single call to FOLLOW is unlikely to reach the goal, the figure shows only a partial expansion along σ_2. FOLLOW partitions Λ into groups $\mathcal{X}_1, \ldots, \mathcal{X}_{16}$, where $\mathcal{X}_j$ includes the vertices that have c_j as their target, i.e., $\mathcal{X}_1 = \{b_1\}, \mathcal{X}_2 = \{b_2, b_3\}, \mathcal{X}_3 = \{b_4, b_5, b_6\}, \mathcal{X}_4 = \{b_7, b_8\}, \mathcal{X}_5 = \{b_9, b_{10}\}, \mathcal{X}_6 = \{b_{11}\}, \mathcal{X}_7 = \{b_{12}, b_{13}, b_{14}\}, \mathcal{X}_8 = \{b_{15}, b_{16}, b_{17}\}, \mathcal{X}_9 = \{b_{18}\}, \mathcal{X}_{10} = \ldots = \mathcal{X}_{16} = \emptyset$.

A. Coordinated Expansion

Given a state $\langle s_1, \ldots, s_n \rangle \in \mathcal{S}$ and routes $\{\sigma_1, \ldots \ \sigma_n\}$, the objective is to compute trajectories $\{\zeta_1, \ldots, \zeta_n\}$, such that ζ_i closely follows σ_i, avoids the obstacles and the other robots, and progresses toward or even reaches $\mathcal{G}_i$. To achieve this, we develop a route follower, $\textsc{follow}(s_i, \sigma_i, \zeta_1, \ldots, \zeta_{i-1})$, which returns a trajectory ζ_i that follows σ_i and avoids the obstacles and robots $1, \ldots i-1$ as they move according to $\zeta_1, \ldots, \zeta_{i-1}$.

When following σ_i, it is important to reach $\sigma_i(1), \ldots, \sigma_i(|\sigma_i|)$ in succession. Skipping some or reaching them out of order could potentially lead to significant deviations from the planned roadmap routes, which in turn could prevent the other robots from reaching their goals. Fig. 4 provides an illustration.

The coordinated expansion invokes FOLLOW with one robot at a time, as shown in Alg. 2. A random order is selected to provide flexibility as a fixed order could make certain expansions difficult. The states along the computed trajectories $\{\zeta_1, \ldots, \zeta_n\}$ are added to the motion tree with the edges going from one state to the next, i.e.,

$$\langle \zeta_1(1), \ldots), \zeta_n(1) \rangle \rightarrow \ldots \rightarrow \langle \zeta_1(\ell), \ldots), \zeta_n(\ell) \rangle. \qquad (13)$$

This requires $\zeta_1, \ldots, \zeta_n$ to have the same duration ℓ. If some ζ_i has not reached $\mathcal{G}_i$, then $\ell = \min\{|\zeta_1|, \ldots, |\zeta_n|\}$. If every ζ_i has reached its goal $\mathcal{G}_i$, then $\ell = \max\{|\zeta_1|, \ldots, |\zeta_n|\}$. In such case, if $|\zeta_i| < \ell$, then the last state of ζ_i is appended to ζ_i several times until $|\zeta_i| = \ell$. FOLLOW guarantees that ζ_i has robot i stopped when it reaches $\mathcal{G}_i$, so appending the last state just makes robot i wait at $\mathcal{G}_i$.

Since sampling-based planners cannot determine when a solution does not exist (probabilistically complete), FOLLOW is run for a limited number of iterations. When it does not reach the end of σ_i, FOLLOW returns a trajectory that has reached $\sigma_i(1), \ldots, \sigma_i(j)$ where j is as large as possible. This enables the approach to make further progress.

In the next iteration of the core loop of the framework (Alg. 1), the coordinated expansion could be used to expand $\mathcal{T}$ from another state along different routes. In this way, the approach explores new areas and continues to progress toward $\mathcal{G}_1, \ldots, \mathcal{G}_n$ until a solution is found.

B. Route Following

$\textsc{follow}(s_i, \sigma_i, \zeta_1, \ldots, \zeta_{i-1})$ returns a trajectory ζ_i that follows σ_i (seeks to reach $\sigma_i(1), \ldots, \sigma_i(|\sigma_i|)$ in succession) while avoiding the obstacles and robots $1, \ldots i-1$ as they move according to $\zeta_1, \ldots, \zeta_{i-1}$. Let $\textsc{polyline}(\sigma_i)$ denote the polyline connecting $\sigma_i(1), \ldots, \sigma_i(|\sigma_i|)$. While following σ_i, FOLLOW is required to be within d_{follow} distance from $\textsc{polyline}(\sigma_i)$. Another parameter, d_{reach}, determines the reach radius for each $\sigma_i(j)$. This gives FOLLOW some flexibility since exact following may not be possible due to constraints imposed by the obstacles and the dynamics.

FOLLOW creates a new motion tree, denoted by Λ, each time it is invoked. To facilitate the expansion of Λ along the route σ_i, groups $\mathcal{X}_1, \ldots, \mathcal{X}_{|\sigma_i|}$ are also created, where $\mathcal{X}_j$ keeps track of the vertices in Λ that have $\sigma_i(j)$ as their target. By design, if $v \in \mathcal{X}_j$, then the trajectory from the root of Λ to v has reached $\sigma_i(1), \ldots, \sigma_i(j-1)$ in succession.

Initially, the motion tree Λ is rooted at s_i and the root vertex is added to $\mathcal{X}_1$. FOLLOW runs until it reaches the end of σ_i or exceeds the bound on the number of iterations. At each iteration, FOLLOW selects the non-empty group $\mathcal{X}_j$ with maximum weight and then expands Λ from $\mathcal{X}_j$ toward $\sigma_i(j)$. The weight is defined as

$$w(\mathcal{X}_j) = \alpha^{j/|\sigma_i|} \beta^{\textsc{NrSel}(\mathcal{X}_j)}, \qquad (14)$$

Algorithm 2 Coordinated Expansion

```
Input: start state $\langle s_1, \ldots, s_n \rangle$; routes $\{\sigma_1, \ldots, \sigma_n\}$;
  $\langle \mathcal{W}, \mathcal{O}, \mathcal{G}_1, \ldots, \mathcal{G}_n, \mathcal{M}_1, \ldots, \mathcal{M}_n \rangle$
Output: trajectories $\{\zeta_1, \ldots, \zeta_n\}$, one for each robot, such that $\zeta_i$ closely
  follows $\sigma_i$, avoids the obstacles and the other robots, and makes progress
  toward or even reaches the goal $\mathcal{G}_i$.
1: $\pi \leftarrow$ RANDOMPERMUTATION($\{1, 2, \ldots, n\}$)
2: for $i = 1 \ldots n$ do
3:   $\zeta_{\pi_i} \leftarrow$ FOLLOW($s_{\pi_i}, \sigma_{\pi_i}, \zeta_{\pi_1}, \ldots, \zeta_{\pi_{i-1}}$)
4: MAKESAMEDURATIONS($\zeta_1, \ldots, \zeta_n$)
5: return $\{\zeta_1, \ldots, \zeta_n\}$

FOLLOW($s_i, \sigma_i, \zeta_1, \ldots, \zeta_{i-1}$)
1: $\Lambda \rightarrow$ INITIALIZEMOTIONTREE($s_i$); $\mathcal{X}_1 \leftarrow \{$ROOT($\Lambda$)$\}$
2: for $iter = 1 \ldots maxNrIters$ do
3:   $\mathcal{X}_j \leftarrow \mathrm{argmax}_{\mathcal{X}_z \in \mathcal{X}}\, w(\mathcal{X}_z)$
4:   $v \leftarrow v' \leftarrow$ SELECTVERTEX($\mathcal{X}_j$); $s_0^{PID} \leftarrow v.s$
5:   $p_{target} \leftarrow$ SAMPLETARGET($\sigma_i(j+1), d_{follow}, d_{reach}$)
6:   for $k = 1 \ldots maxNrControllerSteps$ do
7:     $a \leftarrow$ CONTROLLER($s_{k-1}^{PID}, p_{target}$)
8:     $s_k^{PID} \leftarrow$ SIMULATE($s_{k-1}^{PID}, a, f_i, dt$)
9:     if DISTANCE($s_k^{PID}$, POLYLINE($\sigma_i$)) $> d_{follow}$ or
       COLLISION($s_k^{PID}, \zeta_1(v.depth+k), \ldots, \zeta_{i-1}(v.depth+k)$) then
       break
10:    $v_{new}.[s, depth] \leftarrow$ NEWVERTEX($s_k^{PID}, v'.depth + 1$)
11:    ADDVERTEXANDEDGE($\Lambda, v_{new}, (v', v_{new})$); $v' \leftarrow v_{new}$
12:    if DISTANCE($s_k^{PID}, \sigma_i(j)$) $\leq d_{reach}$ then
13:      if $j = |\sigma_i|$ then return trajectory from root of $\Lambda$ to $v_{new}$
14:      $\mathcal{X}_{j+1} \leftarrow \mathcal{X}_{j+1} \cup \{v_{new}\}$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathcal{X}_{j+1}\}$; break
15:    else $\mathcal{X}_j \leftarrow \mathcal{X}_j \cup \{v_{new}\}$
16: $zmax \leftarrow \max\{z : z \in \{1, \ldots, |\sigma_i|\} \wedge |\mathcal{X}_z| > 0\}$
17: $v_{sol} \leftarrow$ SELECTVERTEXATRANDOM($\mathcal{X}_{zmax}$)
18: return trajectory from root of $\Lambda$ to $v_{sol}$
```

where $\alpha > 1$, $0 < \beta < 1$, and NRSEL($\mathcal{X}_j$) denotes the number of times $\mathcal{X}_j$ has been previously selected. The term $\alpha^{j/|\sigma_i|}$ promotes expansions from groups that are close to the end of σ_i. The term $\beta^{\mathrm{NRSEL}(\mathcal{X}_j)}$ is a penalty to avoid becoming trapped when expansions from $\mathcal{X}_j$ repeatedly fail due to constraints imposed by obstacles and robot dynamics.

After selecting $\mathcal{X}_j$, the objective becomes to expand Λ from $\mathcal{X}_j$ so that it reaches $\sigma_i(j)$. To promote expansions from different states, a vertex v is selected at random from $\mathcal{X}_j$. To provide flexibility, v is extended toward a point p_{target} sampled inside CIRCLE($\sigma_i(j), d_{follow}$). Sampling p_{target} near $\sigma_i(j)$ enables FOLLOW to generate different trajectories and reach other points from which it may be easier to reach $\sigma_i(j)$.

A collision-free and dynamically-feasible trajectory ζ is generated from v toward p_{target} by applying controls and integrating the motion equations f_i for several steps. A proportional-integral-derivative (PID) controller [28] is used to generate controls that steer the robot toward p_{target}, e.g., by turning the wheels and then moving straight. If DISTANCE($p_{target}, \sigma_i(|\sigma_i|)$) $\leq d_{reach}$, then the controller will also stop the vehicle when it gets to p_{target}.

Let s_k^{PID} denote the state obtained at the k-th iteration of the PID controller. If s_k^{PID} is in collision or its distance to POLYLINE(σ_i) is more than d_{follow}, then s_k^{PID} is discarded, the trajectory generation from v stops, and $\zeta = \{s_1^{PID}, \ldots, s_{k-1}^{PID}\}$ is returned. Note that collisions are determined not only with the obstacles but also with robots $1, \ldots, i-1$ as they move according to $\zeta_1, \ldots, \zeta_{i-1}$. To check for such collisions, let $d(v)$ denote the length of the path in the motion tree Λ from its root to v. Since s_k^{PID} is generated k steps after v, the other robots would be in states $\zeta_1(d(v)+k), \zeta_2(d(v)+k), \ldots, \zeta_{i-1}(d(v)+k)$, so these states are checked for collisions with robot i. The trajectory generation from v also stops, successfully, when DISTANCE($s_k^{PID}, \sigma_i(j)$) $\leq d_{reach}$. In this case, $\zeta = \{s_1^{PID}, \ldots, s_k^{PID}\}$. Each state of ζ is added as a vertex to the motion tree Λ with the previous state as its parent (first state has v as its parent). States $\zeta(1), \ldots, \zeta(|\zeta| - 1)$ are added to $\mathcal{X}_j$. The last state is added to $\mathcal{X}_{j+1}$ if it is within d_{reach} distance from $\sigma_i(j)$; otherwise, it is added to $\mathcal{X}_j$.

If FOLLOW is unable to reach the end of σ_i, then it returns a trajectory that has reached $\sigma_i(1), \ldots, \sigma_i(j)$ where j is as large as possible. This corresponds to finding the non-empty group $\mathcal{X}_z$ with the largest z, i.e.,

$$zmax = \max\{z : z \in \{1, \ldots, |\sigma_i|\} \wedge |\mathcal{X}_z| > 0\}. \qquad (15)$$

Since $\mathcal{X}_{zmax}$ could have more than one vertex, a vertex v' is selected at random and the trajectory from the root of Λ to v' is returned. This enables the approach to still progress even when FOLLOW is unable to reach the end of σ_i.

V. Experiments and Results

Experiments are conducted using complex environments and vehicle models with nonlinear dynamics (Section II).

A. Problem Instances

Scenes are shown in Figs. 1 and 5. For each combination of a scene and number of robots n, 60 different problem instances are generated. In scenes 1–5, the robots and goals are placed at random inside $\mathcal{W}$. In scenes 6–10, the robots and goals are placed at random inside certain areas of $\mathcal{W}$ to make the problems more challenging as the robots would have to carefully coordinate their motions to reach the goals.

B. Comparisons to Related Work

Our approach is compared to [24], referred here as LP, which was specifically designed for multi-robot motion planning and was shown to be significantly faster than other methods. Both our approach and LP use WHCA* [29] as MULTIAGENTSEARCH.

Comparisons are also done with a prioritized version of RRT [18], denoted by pRRT, which generates motions plans one robot at a time, taking into account the planned motions of the previous robots. We also include in the comparisons two prioritized RRT variants. The first one, denoted by pAnyAngleRRT, expands RRT along a route obtained by an any-angle pathfinder [25]. The second one, denoted by pA*RRT, uses an A* expansion of RRT [26]. The implementations of pRRT, pAnyAngleRRT, and pA*RRT use goal bias, multi-step expansions, and efficient nearest-neighbor data structures, as recommended in the literature [23], [30], [31].

For a given scene and number of robots n, each planner is run on each of the 60 problem instances. Results report the mean runtime and solution length after dropping runs that are below the first quartile or above the third quartile to avoid

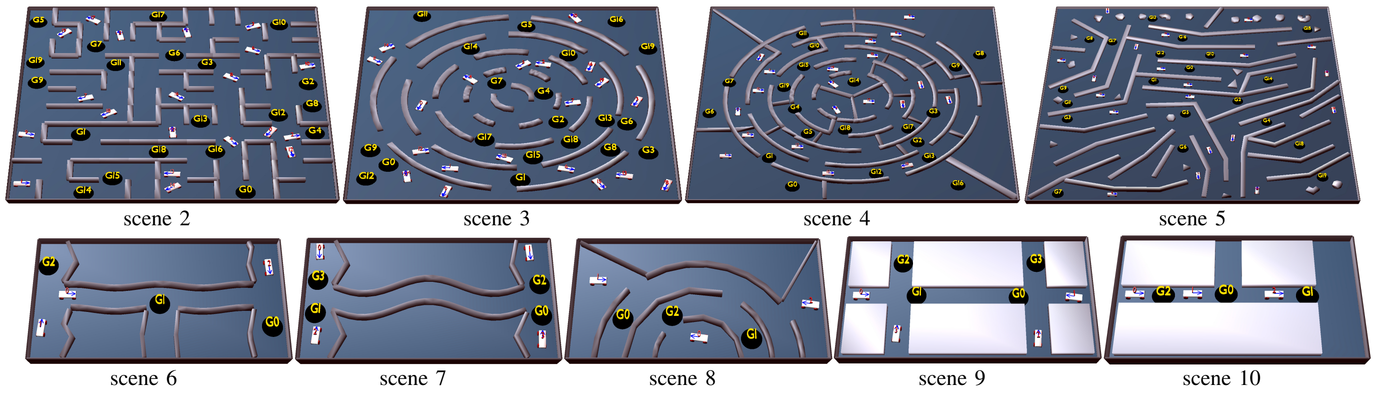

Fig. 5. Scenes 2–10 used in the experiments (scene 1 shown in Fig. 1). Videos of solutions can be found at goo.gl/ZXNqbd.

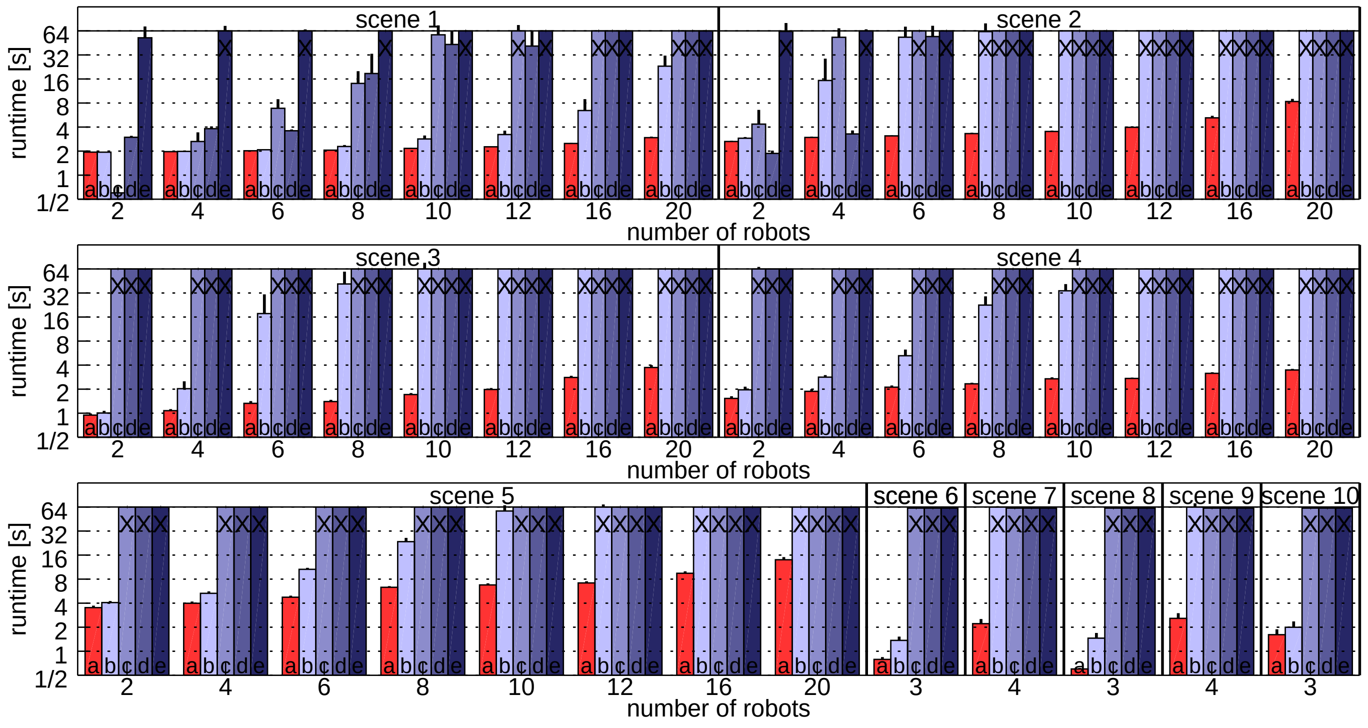

Fig. 6. Runtime results when comparing (a) our approach to (b) LP, (c) pRRT, (d) pAnyAngleRRT, and (e) pA*RRT. Entries marked with X indicate failure to solve the problem instances within the runtime limit of 60s per run. Each bar indicates one standard deviation.

the influence of outliers. Runtime measures everything from reading the input until reporting a solution or reaching the runtime limit (set to 60s). Experiments were run on an Intel Core i7 (1.90GHz).

Fig. 6 shows that our approach is significantly faster than pRRT, pAnyAngleRRT, pA*RRT, and LP. Prioritized RRT and its variants time out even on small instances with 2–6 robots. Only on scene 1, pRRT and pAnyAngleRRT are able to solve problem instances with up to 8 robots. Scene 1 has many open areas, which makes it less likely for the motions of one robot to prevent the other robots from reaching their goals. In the other scenes, where cooperation and coordination among the robots is essential, pRRT and its variants often time out without finding a solution.

LP is faster than pRRT, pAnyAngleRRT, pA*RRT. As the number of robots increases, however, scalability starts to become an issue as LP uses a centralized motion-tree expansion, which suffers from the high dimensionality of the composite state space. As a result, LP often times out on instances with more than 10 robots. In contrast, our approach remains fast, only requiring 4–8s for problems with 20 robots. The efficiency of our approach derives from the coordinated expansion, which closely follows the routes computed by the multi-agent search.

Note that in scenes 6–10, the robots have to carefully coordinate their motions to reach the goals. Our approach again is significantly faster than LP. The issue in these cases for LP is not scalability, but the high likelihood that, in tight spaces, the centralized expansion leads to frequent collisions, making it difficult for LP to expand the motion tree. In contrast, our approach uses a coordinated expansion based on FOLLOW. FOLLOW plans one robot at a time, taking into account the previous plans, as it follows the routes. Even when

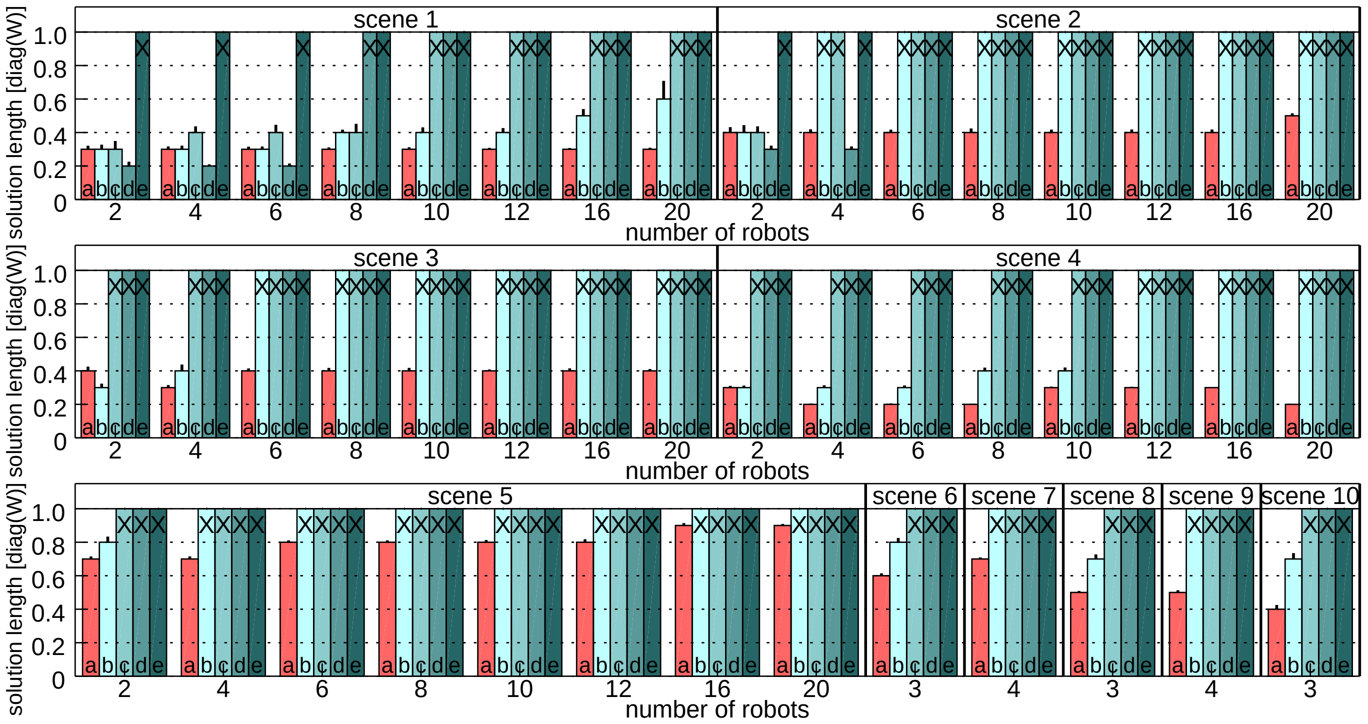

Fig. 7. Results on solution length for (a) our approach, (b) `LP`, (c) `pRRT`, (d) `pAnyAngleRRT`, and (e) `pA*RRT`. Solution length for a problem instance with n robots is measured as the average length of the solution trajectories for the n robots. It is then scaled by the length of the diagonal in $\mathcal{W}$.

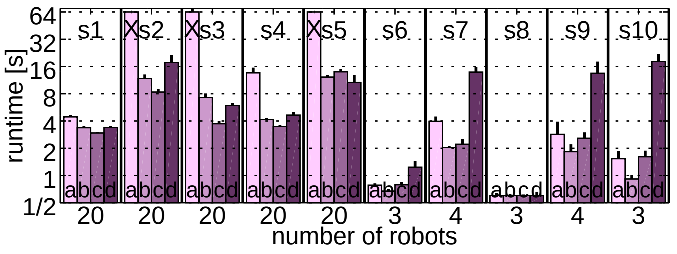

Fig. 8. Runtime of our approach when varying the maximum number of iterations for FOLLOW as (a) 5, (b) 50, (c) 350, (d) 5000.

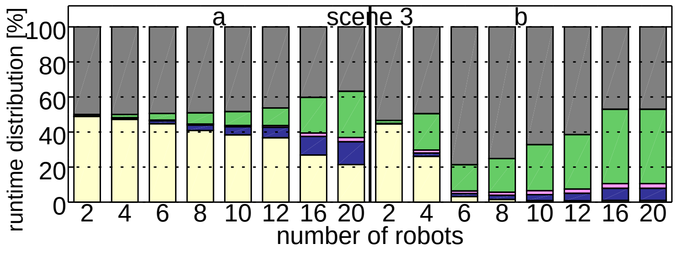

Fig. 9. Runtime distribution of various components for (a) our approach and (b) `LP`. From top to bottom: roadmap construction, multi-agent search, SIMULATE, collision-checking, other.

it is unable to reach the end of a route σ_i, FOLLOW often still makes considerable progress and reaches $\sigma_i(1), \ldots, \sigma_i(k)$. In this way, the motion tree continues to expand rapidly toward the goals.

Fig. 7 shows that our approach is able to generate shorter solutions than `LP`. Since both our approach and `LP` use the same multi-agent graph search, the better performance by our approach is due to FOLLOW which closely follows the routes. When `pAnyAngleRRT` is able to solve the problems, it tends to generate short solutions. This is due to the any-angle path bias, which only expands inside a narrow corridor along the route to the goal. Moreover, `pAnyAngleRRT` does not update its route, so it aggressively tries to find a solution along that route. Often, this causes `pAnyAngleRRT` to become stuck, but when it succeeds, the solution would be short. In contrast, our approach will abandon the current route when the expansion makes little progress, seeking to expand along alternative routes, which could be longer. This is essential to maintaining runtime efficiency.

C. *Number of Iterations for* FOLLOW

Fig. 8 shows the runtime of our approach when varying the maximum number of iterations for FOLLOW (Alg. 2:2). When the number is small, FOLLOW makes minimal progress, slowing down the motion-tree expansion, especially in closed scenes. When the number is large, FOLLOW could waste time with a route that is impossible to follow due to constraints imposed by the dynamics and obstacles. FOLLOW, however, works well for a wide range of values. We recommend, as we did for the experiments, choosing a value at random from $[100, 500]$ each time FOLLOW is called.

D. *Runtime Distribution*

Fig. 9 shows the runtime distribution for various components of our approach and `LP`. As the number of robots increases, the percentage taken by multi-agent search also increases. This indicates that multi-agent search effectively guides the motion-tree expansion, reducing the time spent exploring parts of the state space that do not lead to a solution. When compared to LP, the results show that our approach significantly reduces the percentage taken by collision checking. This is due to the coordinated expansion, which closely follows the routes. As these routes are non-conflicting (in the abstraction), by closely following them in the continuous state space, our approach reduces the number of collisions that occur during the motion-tree expansion.

VI. Discussion

This work leveraged the idea of using multi-agent search to guide sampling-based motion planning in order to effectively solve challenging multi-robot motion-planning problems with dynamics. We developed a coordinated expansion of the motion tree guided by multi-agent search. A key aspect was the route follower, which closely followed the route for robot i while avoiding collisions with the previous robots. Experiments demonstrated significant speedups over related work. One direction for future research, in order to facilitate the motion-tree expansion, is to require from multi-agent search to find non-conflicting routes that maximize separation among the robots and clearance from the obstacles. Another direction is to incorporate high-level tasks.

Our approach maintains the probabilistic completeness of LP [24]. The framework remains the same except that it replaces the centralized expansion with the coordinated expansion. Our approach, as in prior work, expands the motion tree in the composite state space $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$, which is essential to guarantee probabilistic completeness.

The centralized expansion in LP simply runs a controller for each robot for multiple steps, stopping as soon as a collision is detected, and adding the intermediate states to the motion tree. Such behavior can be obtained by the coordinated expansion by setting $\mathit{maxNrIters} = 1$ (Alg. 2:FOLLOW:2) and $d_{\mathit{follow}} = \inf$ (Alg. 2:FOLLOW:9).

Also note that our approach tends to generate short solutions since the motion tree is expanded along routes obtained by multi-agent search. A future research direction would be to provide optimality guarantees.

References

[1] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *AAAI Conference on Artificial Intelligence*, Bellevue, WA, 2013, pp. 1443–1449.

[2] D. Silver, "Cooperative pathfinding," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 1, Marina del Rey, CA, 2005, pp. 117–122.

[3] N. R. Sturtevant and M. Buro, "Improving collaborative pathfinding using map abstraction," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Marina del Rey, CA, 2006, pp. 80–85.

[4] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Symposium on Combinatorial Search*, Prague, Czech Republic, 2014, pp. 19–27.

[5] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *International Joint Conference on Artificial Intelligence*, Barcelona, Spain, 2011, pp. 668–673.

[6] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. R. Sturtevant, R. C. Holte, and J. Schaeffer, "Enhanced partial expansion a*," *Journal of Artificial Intelligence Research*, vol. 50, pp. 141–187, 2014.

[7] J. Svancara and P. Surynek, "New flow-based heuristic for search algorithms solving multi-agent path finding," in *International Conference on Agents and Artificial Intelligence*, Porto, Portugal, 2017, pp. 451–458.

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

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

[10] A. Felner, R. Stern, E. Shimony, E. Boyarski, M. Goldenerg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek, "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *Symposium on Combinatorial Search*, Pittsburgh, PA, 2017, pp. 29–37.

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

[12] O. Amir, G. Sharon, and R. Stern, "Multi-agent pathfinding as a combinatorial auction," in *AAAI Conference on Artificial Intelligence*, Austin, TX, 2015, pp. 2003–2009.

[13] E. Erdem, D. G. Kisa, U. Öztok, and P. Schueller, "A general formal framework for pathfinding problems with multiple agents," in *AAAI Conference on Artificial Intelligence*, Bellevue, WA, 2013, pp. 290–296.

[14] M. M. Khorshid, R. C. Holte, and N. R. Sturtevant, "A polynomial-time algorithm for non-optimal multi-agent pathfinding," in *Symposium on Combinatorial Search*, Barcelona, Spain, 2011, pp. 76–83.

[15] Q. Sajid, R. Luna, and K. E. Bekris, "Multi-agent pathfinding with simultaneous execution of single-agent primitives," in *Symposium on Combinatorial Search*, Niagara Falls, Canada, 2012, pp. 88–96.

[16] B. d. Wilde, A. W. Ter Mors, and C. Witteveen, "Push and rotate: a complete multi-agent pathfinding algorithm," *Journal of Artificial Intelligence Research*, vol. 51, pp. 443–492, 2014.

[17] A. Botea and P. Surynek, "Multi-agent path finding on strongly biconnected digraphs," in *AAAI Conference on Artificial Intelligence*, Austin, TX, 2015, pp. 2024–2030.

[18] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[19] I. A. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.

[20] E. Plaku, "Region-guided and sampling-based tree search for motion planning with dynamics," *IEEE Transactions on Robotics*, vol. 31, pp. 723–735, 2015.

[21] L. E. Kavraki, P. Švestka, 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.

[22] H. Keller, *Numerical Methods for Two-Point Boundary-Value Problems*. New York, NY: Dover, 1992.

[23] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[24] D. Le and E. Plaku, "Cooperative multi-robot sampling-based motion planning with dynamics," in *International Conference on Planning and Scheduling*, Pittsburgh, PA, 2017, pp. 513–521, best Robotics Paper.

[25] L. Palmieri, S. Koenig, and K. O. Arras, "RRT-based nonholonomic motion planning using any-angle path biasing," in *IEEE International Conference on Robotics and Automation*, Stockholm, Sweden, 2016, pp. 2775–2781.

[26] J. Li, S. Liu, B. Zhang, and X. Zhao, "RRT-A* motion planning algorithm for non-holonomic mobile robot," in *IEEE Annual Conference of the Society of Instrument and Control Engineers of Japan*, Sapporo, Japan, 2014, pp. 1833–1838.

[27] M. Brunner, B. Brüggemann, and D. Schulz, "Hierarchical rough terrain motion planning using an optimal sampling-based method," in *IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 5539–5544.

[28] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley and Sons, 2005.

[29] D. Silver, "Cooperative pathfinding," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 1, Marina del Ray, CA, 2005, pp. 117–122.

[30] S. M. LaValle, *Planning Algorithms*. Cambridge, MA: Cambridge University Press, 2006.

[31] ——, "Motion planning: The essentials," *IEEE Robotics & Automation Magazine*, vol. 18, no. 1, pp. 79–89, 2011.