Cooperative Multi-Robot Sampling-Based Motion Planning with Dynamics

Duong Le and Erion Plaku
Department of Electrical Engineering and Computer Science
Catholic University of America, Washington DC, 22064

Abstract

This paper develops an effective, cooperative, and probabilistically-complete multi-robot motion planner. The approach takes into account geometric and differential constraints imposed by the obstacles and the robot dynamics by using sampling to expand a motion tree in the composite state space of all the robots. Scalability and efficiency is achieved by using solutions to a simplified problem representation that does not take dynamics into account to guide the motion-tree expansion. The heuristic solutions are obtained by constructing roadmaps over low-dimensional configuration spaces and relying on cooperative multi-agent graph search to effectively find graph routes. Experimental results with second-order vehicle models operating in complex environments, where cooperation among the robots is required to find solutions, demonstrate significant improvements over related work.

Introduction

An increasing number of robotic applications, ranging from exploration to search-and-rescue, require a team of robots to navigate in unstructured, obstacle-rich, environments. A fundamental component of enhancing the autonomy for a team of robots is the ability to plan dynamically-feasible motions that enable each robot to reach its destination while avoiding collisions with obstacles and other robots.

Multi-robot motion planning poses significant challenges. The planned motions must not only avoid collisions but also obey the robot dynamics, which constrain, for example, its velocity, direction, turning radius, and acceleration. This is challenging since motion planning even for one robot is PSPACE-complete (Reif 1979) when considering only collision avoidance and becomes undecidable when coupled with differential constraints imposed by the dynamics (Branicky 1995). Moreover, in many scenarios, the robots need to cooperate to avoid deadlock, for example, when blocking each other from reaching the corresponding destinations.

Multi-robot approaches can be divided based on whether the geometric and differential constraints are respected: (i) points over graphs with no dynamics, (ii) geometric shapes but no dynamics, and (iii) geometric shapes and dynamics.

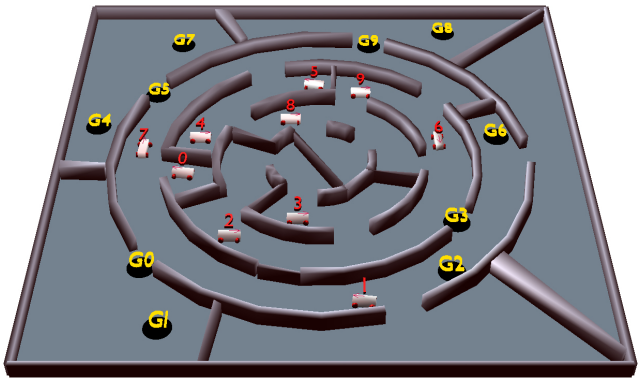

Figure 1: Each robot, whose dynamics are given by second-order differential equations, needs to reach its goal ($\mathcal{G}_i$ for robot i) while avoiding collisions with obstacles and other robots. In this and many other scenarios, the robots need to coordinate their motions to avoid blocking each other. Videos of solutions obtained by our approach on this and other scenes can be found at https://goo.gl/sQ6irb. Figure best viewed in color and on screen.

Multi-robot planning for point robots with no dynamics has often been studied in the context of multi-agent pathfinding over graphs. Cooperative approaches often plan paths for the agents separately, and repair those paths when conflicts arise (Silver 2005; Jansen and Sturtevant 2008). Global approaches often search over the composite space of all the agents. The conflict-based search avoids operating over the composite space by maintaining a search tree that represents the conflicts that arise among the agents (Sharon et al. 2015). Other approaches employ auctions (Amir, Sharon, and Stern 2015) or answer-set programming (Erdem et al. 2013).

When considering the robot geometries but not the dynamics, sampling-based approaches have often been used. Centralized approaches operate over the composite configuration space, treating the robots as one system. This makes it possible to use any sampling-based approach, e.g., `PRM` (Kavraki et al. 1996), `RRT` (LaValle and Kuffner 2001), but does not scale well due to the high-dimensionality of the composite configuration space. To improve the scalability of `PRM`, the composite roadmap can be maintained implicitly

as a product of the individual roadmaps over each configuration space (Solovey, Salzman, and Halperin 2015). Decoupled approaches plan the robot paths separately, and then use velocity tuning to coordinate the paths (Choset et al. 2005) or subdimensional expansion to repair the paths (Wagner, Kang, and Choset 2012). Prioritized approaches also plan the robot paths separately, but treat the planned paths for robots $1, \ldots, i-1$ as moving obstacles when planning the path for the i-th robot. Decoupled or prioritized approaches cannot guarantee completeness, since coordination is not always possible and previously planned paths may make it impossible for the next robot to reach its destination.

When considering the robot geometries and the dynamics, sampling-based motion planners often expand a motion tree. Roadmaps cannot generally be used since each roadmap edge requires exact steering to generate a dynamically-feasible trajectory that connects its states. This gives rise to two-boundary value problems, which can be solved analytically only in limited cases, while numerical methods are expensive (Keller 1992; Cheng, Frazzoli, and LaValle 2008). A motion tree is incrementally expanded from the initial state by adding collision-free and dynamically-feasible trajectories as branches (LaValle and Kuffner 2001; Plaku 2015; Şucan and Kavraki 2012). Exact steering is not needed since each branch is generated by applying control actions and numerically integrating the differential constraints. Motion-tree planners can be used in a decoupled, prioritized, or centralized setting to plan for multiple robots. However, due to the complexity of the problem, there is a significant degradation in planning runtime as the number of robots is increased.

Contribution This paper develops an efficient, cooperative, and probabilistically-complete multi-robot motion planner that takes into account geometric and differential constraints imposed by the obstacles and the robot dynamics by expanding a motion tree in the composite state space of all the robots. Scalability and efficiency is achieved by using solutions to a simplified problem representation that does not take dynamics into account. The heuristic solutions are obtained by constructing roadmaps over low-dimensional configuration spaces and relying on cooperative multi-agent graph search to effectively find graph routes. The motion tree is partitioned into equivalence classes based on a mapping from robot states to roadmap vertices. Each equivalence class is associated with a heuristic cost based on the length of the roadmap routes for each robot to reach its goal. To promote efficiency, priority is given to expansions from equivalence classes with small heuristic costs. When the expansion fails to make progress, the corresponding equivalence class is penalized in order to promote the discovery of alternative routes to the goal. Experimental results with second-order vehicle models operating in complex environments, where cooperation among the robots is required to find solutions, demonstrate significant speedups over related work.

The proposed approach leverages the notion of using discrete search to guide the motion-tree expansion (Plaku, Kavraki, and Vardi 2010; Kiesel, Burns, and Ruml 2012; Le and Plaku 2014; Plaku 2015). These related approaches, however, have been designed for a single robot. It remains open to effectively extend these approaches to multiple robots. Drawing from Solovey et al., (2015), the composite roadmap is not constructed explicitly but rather is maintained implicitly via the edges function which combines the individual roadmaps. This related work does not take dynamics into account, so it ends by using graph search over the implicit roadmap. In distinction, our work takes dynamics into account and uses graph paths over the implicit roadmap as heuristics to guide the motion-tree expansion.

The proposed approach, as other sampling-based motion planners, assumes a known map of the environment, no uncertainty in the executed actions, and no noise. When a map is not available or when executed in a real robot, sampling-based motion planners are commonly used in a replanning framework. This paper does not focus on replanning, but the approach can be used with any replanning framework for sampling-based motion planners.

Problem Formulation

This section defines the robot model, motion trajectories, and the multi-robot motion-planning problem.

Robot Model and Motion Trajectory

Each robot model is defined by its geometric shape $\mathcal{P}_i$, state space $\mathcal{S}_i$, action space $\mathcal{A}_i$, and motion equations f_i. $\mathcal{S}_i$ consists of a set of variables that describe the robot state, such as position, orientation, steering angle, and velocity. $\mathcal{A}_i$ defines the control actions that can be applied to the robot, such as setting the acceleration and steering rate. The motion equations encapsulate the underlying robot dynamics and are often expressed as a set of differential equations

$$\dot{s} = f_i(s, a), \tag{1}$$

which describe how the state $s \in \mathcal{S}_i$ changes when applying the control action $a \in \mathcal{A}_i$. The new state $s_{\text{new}} \in \mathcal{S}$, obtained by applying a to s for one time step dt, is computed by

$$s_{\text{new}} \leftarrow \text{SIMULATE}(s, a, f_i, dt), \tag{2}$$

which numerically integrates f_i, e.g., via Runge-Kutta.

As an example, the state $s = (x, y, \theta, \psi, v)$ of the vehicle model used in the experiments defines the position (x, y), orientation θ, steering angle ψ, and velocity v. The vehicle is controlled by setting the acceleration a_{acc} and steering rate a_ω. The motion equations f are defined as

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

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

where L is the distance between the back and front wheels.

A dynamically-feasible trajectory $\zeta_i : \{1, \ldots, \ell\} \rightarrow \mathcal{S}_i$ is defined by a start state $s \in \mathcal{S}_i$ and a sequence of control actions $\langle a_i^1, \ldots, a_i^{\ell-1} \rangle$, where $a_i^j \in \mathcal{A}_i$. The trajectory ζ_i is obtained by starting at s and applying the control actions in succession, i.e., $\zeta_i(1) = s$ and $\forall j \in \{2, \ldots, \ell\}$:

$$\zeta_i(j) = \text{SIMULATE}(\zeta_i(j-1), a_i^{j-1}, f_i, dt). \tag{5}$$

To facilitate the presentation, $\text{POS}(s)$ and $\text{ORIENTATION}(s)$ denote the position and orientation components of a state $s \in \mathcal{S}_i$, and $\text{PLACEMENT}(\mathcal{P}_i, s)$ denotes the placement of the shape $\mathcal{P}_i$ according to $\text{POS}(s)$ and $\text{ORIENTATION}(s)$.

Multi-Robot Motion-Planning Problem

The environment in which the robots operate is defined by its bounding box $\mathcal{W}$ and obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$, where $\mathcal{O}_j \subseteq \mathcal{W}$. Let $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ denote the set of the robot models, where $\mathcal{M}_i = \langle \mathcal{P}_i, \mathcal{S}_i, \mathcal{A}_i, f_i, s_i^{\text{init}}, \mathcal{G}_i \rangle$ defines the i-th robot in terms of its geometric shape $\mathcal{P}_i$, state space $\mathcal{S}_i$, action space $\mathcal{A}_i$, motion equations f_i, initial state $s_i^{\text{init}} \in \mathcal{S}_i$, and goal region $\mathcal{G}_i \subseteq \mathcal{W}$. Let dt denote the time step.

Let $\text{COLLISION} : \mathcal{S}_1 \times \ldots \times \mathcal{S}_n \rightarrow \{\texttt{false}, \texttt{true}\}$ denote the collision-checking function. Given a composite state $\langle s_1, \ldots, s_n \rangle$, where s_i denotes the state of the i-th robot, $\text{COLLISION}(s_1, \ldots, s_n) = \texttt{false}$ if and only if

- each robot is in $\mathcal{W}$, i.e., $\bigcup_{i=1}^{n} \text{PLACEMENT}(\mathcal{P}_i, s_i) \subseteq \mathcal{W}$,
- there is no robot-obstacle collision, i.e., $(\bigcup_{i=1}^{n} \text{PLACEMENT}(\mathcal{P}_i, s_i)) \cap (\bigcup_{j=1}^{m} \mathcal{O}_j) = \emptyset$, and
- there is no robot-robot collision, i.e., $\forall 1 \leq i < j \leq n$: $\text{PLACEMENT}(\mathcal{P}_i, s_i) \cap \text{PLACEMENT}(\mathcal{P}_j, s_j) = \emptyset$.

This paper uses PQP (Larsen et al. 1999) to efficiently implement COLLISION.

In multi-robot motion planning, the objective is to compute a dynamically-feasible trajectory ζ_i for each robot that starts at the initial state s_i^{init} and reaches the goal $\mathcal{G}_i$ while avoiding collisions with obstacles and other robots. More formally, let $\{\langle a_1^1, \ldots, a_1^{\ell-1} \rangle, \ldots, \langle a_n^1, \ldots, a_n^{\ell-1} \rangle\}$ denote a set of control action sequences. Let $\{\zeta_1, \ldots, \zeta_n\}$ denote the resulting dynamically-feasible trajectories, where $\zeta_i : \{1, \ldots, \ell\} \rightarrow \mathcal{S}_i$ is obtained by starting at initial state s_i^{init} and applying $\langle a_i^1, \ldots, a_i^{\ell-1} \rangle$ in succession, as defined by Eqn. 5. These trajectories constitute a solution to the multi-robot motion-planning problem when

- each ζ_i reaches $\mathcal{G}_i$, i.e., $\text{POS}(\zeta_i(\ell)) \in \mathcal{G}_i$, and
- no robot-robot or robot-obstacle collisions occur, i.e., $\forall j \in 1, \ldots, \ell$: $\text{COLLISION}(\zeta_1(j), \ldots, \zeta_n(j)) = \texttt{false}$.

Geometric Multi-Robot Path Planning

The overall approach, presented in the next section, uses solutions to a simplified problem representation as heuristic guides. The simplified representation is obtained by considering motion planning for each robot in a low-dimensional configuration space $\mathcal{C}_i$ rather than the state space $\mathcal{S}_i$. Since $\mathcal{C}_i$ discards the dynamics, the robot is free to move and rotate in any direction. As an illustration, for the vehicle model used in this paper, a configuration $c \in \mathcal{C}_i$ can be defined as $c = \langle x, y, \theta \rangle$ by considering only the position and orientation components of the state, i.e., $\mathcal{C}_i = SE(2)$. The configuration of a state $s \in \mathcal{S}_i$ is denoted by $\text{CFG}(s)$.

Before relating the details of the overall approach, we present here our approach for the geometric multi-robot path-planning problem.

Roadmap Construction

Drawing from PRM (Kavraki et al. 1996), the approach builds a roadmap over each $\mathcal{C}_i$ as an undirected, weighted, graph $\mathcal{R}_i = (V_{\mathcal{R}_i}, E_{\mathcal{R}_i}, \text{COST}_{\mathcal{R}_i})$ by sampling and connecting collision-free configurations. Fig. 2 shows an example.

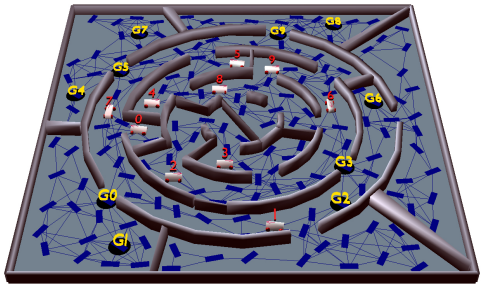

Figure 2: An example of a roadmap.

A vertex $c_i \in V_{\mathcal{R}_i}$ corresponds to a collision-free configuration in $\mathcal{C}_i$. An edge $(c_i, c_i') \in E_{\mathcal{R}_i}$ denotes a collision-free path connecting c_i to c_i'. During the construction of $\mathcal{R}_i$, collision checking is done only between the i-th robot and the obstacles. The cost of an edge $(c_i, c_i') \in E_{\mathcal{R}_i}$ is defined as

$$\text{COST}_{\mathcal{R}_i}(c_i, c_i') = \rho_i(c_i, c_i') / \text{CLEARANCE}(c_i, c_i'), \qquad (6)$$

where $\rho_i : \mathcal{C}_i \times \mathcal{C}_i \rightarrow \mathbb{R}^{\geq 0}$ is a distance metric over $\mathcal{C}_i$ and $\text{CLEARANCE}(c_i, c_i')$ denotes the minimum distance from the obstacles to the segment connecting $\text{POS}(c_i)$ to $\text{POS}(c_i')$.

$\text{CLEARANCE}(c_i, c_i')$ is also used in the edge cost since, without it, shortest paths are likely to bring the robot close to the obstacles, making navigation more difficult. PQP (Larsen et al. 1999) or other collision-detection packages can be used to efficiently compute $\text{CLEARANCE}(c_i, c_i')$.

When constructing the roadmap $\mathcal{R}_i$, the initial and goal configurations, denoted by c_i^{init} and c_i^{goal}, are first added to $V_{\mathcal{R}_i}$. The initial configuration is obtained from the initial state, i.e., $c_i^{\text{init}} = \text{CFG}(s_i^{\text{init}})$. The goal configuration is obtained by repeatedly sampling a random position inside $\mathcal{G}_i$ and a random orientation until the resulting robot placement is not in collision. The roadmap is further populated by sampling random configurations in $\mathcal{C}_i$ and keeping those that are not in collision. Afterwards, attempts are made to connect each configuration to several of its nearest neighbors with collision-free paths. The path connecting configurations c_i and c_i', denoted by $\text{PATH}(c_i, c_i')$, is defined by interpolation in $\mathcal{C}_i$. If $\text{PATH}(c_i, c_i')$ does not collide with the obstacles, then the edge (c_i, c_i') is added to $E_{\mathcal{R}_i}$. Path collision checking is done using a subdivision or an incremental approach (Choset et al. 2005). The process of sampling and connecting collision-free configurations is repeated until c_i^{init} and c_i^{goal} belong to the same roadmap connected component. When a solution exist, the probability that the roadmap finds it approaches one rapidly, as shown by the probabilistic completeness of PRM (Kavraki et al. 1996).

If the robots have the same configuration space and shape, i.e., $\mathcal{C}_1 = \ldots = \mathcal{C}_n$ and $\mathcal{P}_1 = \ldots = \mathcal{P}_n$, then only one roadmap is constructed. The initial and goal configurations of each robot are added to the roadmap and the process of sampling and connecting configurations continues until each c_i^{init} belongs to the same roadmap component as c_i^{goal}.

Note that an alternative approach to constructing roadmaps $\mathcal{R}_1, \ldots, \mathcal{R}_n$ over $\mathcal{C}_1, \ldots, \mathcal{C}_n$ would be to explicitly construct a roadmap over the composite configuration

space $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$. Such an approach, however, is known to impose a significant computational cost due to the increased dimensionality, rendering the roadmap construction impractical (Choset et al. 2005, chap. 7).

Multi-Robot Roadmap Search

After constructing roadmaps $\mathcal{R}_1, \ldots, \mathcal{R}_n$, discrete search is used to find paths that enable each robot to reach its goal while avoiding collisions with the other robots (the roadmap construction ensures that there would be no collisions with the obstacles). Specifically, MULTIROADMAPSEARCH$(\mathcal{R}_1, \ldots, \mathcal{R}_n, c_1, \ldots, c_n)$ computes collision-free paths $\sigma_1, \ldots, \sigma_n$ such that each σ_i is over $\mathcal{R}_i$, starts at c_i, and ends at c_i^{goal}.

MULTIROADMAPSEARCH is defined over the composite graph $\mathcal{R} = \mathcal{R}_1 \times \ldots \times \mathcal{R}_n$, which is not computed explicitly but rather implicitly via the function, i.e.,

$$\text{EDGES}(\langle c_1, \ldots, c_n \rangle) = \{\langle c'_1, \ldots, c'_n \rangle : (c_1, c'_1) \in E_{\mathcal{R}_1} \wedge \ldots \wedge (c_n, c'_n) \in E_{\mathcal{R}_n} \wedge \quad (7)$$

$$\text{COLLISION}(\langle c_1, \ldots, c_n \rangle, \langle c'_1, \ldots, c'_n \rangle) = \texttt{false}\}. \quad (8)$$

A composite edge $(\langle c_1, \ldots, c_n \rangle, \langle c'_1, \ldots, c'_n \rangle)$ is collision free when the robots do not collide with each other as they move along $\text{PATH}(c_1, c'_1), \ldots, \text{PATH}(c_n, c'_n)$. Caching is used to speed up collision checking by remembering configuration pairs which have been previously checked.

Any state-of-the-art multi-agent graph search can be used to implement MULTIROADMAPSEARCH. This paper uses Windowed Hierarchical Cooperative A* (WHCA*)(Silver 2005) since it is efficient and scalable.

Method

The geometric approach presented in the previous section takes into account the robot shapes but not the dynamics. As such, there is no guarantee that the geometric solutions are dynamically feasible. We present here the complete version of our approach, which takes into account the robot shapes and dynamics, and produces collision-free and dynamically-feasible solutions. The approach uses roadmap routes to the goal regions as heuristics to effectively guide the motion-tree expansion. Fig. 3 shows a schematic representation.

Motion Tree in the Composite State Space

To account for the dynamics, the overall approach expands a motion tree $\mathcal{T}$ in the composite state space $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$. The motion tree $\mathcal{T}$, which is maintained as a directed graph $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$, is rooted at $\langle s_1^{\text{init}}, \ldots, s_n^{\text{init}} \rangle$ and is incrementally expanded by adding new vertices and edges. Each vertex $v \in V_\mathcal{T}$ is associated with a collision-free composite state, denoted by $v.\text{states}$. The notation $v.\text{states}[i]$ denotes the state corresponding to the i-th robot. An edge $(v, v') \in E_\mathcal{T}$ is associated with a collision-free and dynamically-feasible motion from $v.\text{states}$ to $v'.\text{states}$. The edge (v, v') is labeled with the control actions $\langle a_1, \ldots, a_n \rangle$ that were applied to $v.\text{states}$ to generate $v'.\text{states}$, i.e., $\forall i \in \{1, \ldots, n\} : v'.\text{states}[i] = \text{SIMULATE}(v.\text{states}[i], a_i, f_i, dt)$.

A solution to the multi-robot motion-planning problem is found when a new vertex v_{new} is added to $\mathcal{T}$ such that each state in $v_{\text{new}}.\text{states}$ has reached the corresponding goal, i.e., $\forall i \in \{1, \ldots, n\} : \text{POS}(v_{\text{new}}.\text{states}[i]) \in \mathcal{G}_i$. In such a case, the solution trajectory ζ_i for the i-th robot corresponds to $\langle v_1.\text{states}[i], \ldots, v_\ell.\text{states}[i] \rangle$, where $\langle v_1, \ldots, v_\ell \rangle$ with $v_\ell = v_{\text{new}}$ denotes the path from the root of $\mathcal{T}$ to v_{new}.

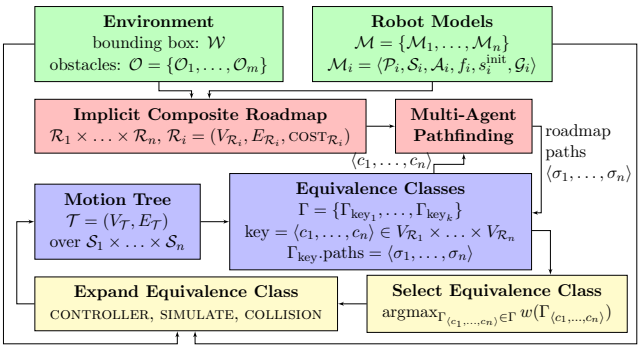

Figure 3: Schematic illustration of the proposed approach.

Using the Roadmaps and Discrete Search to Guide the Motion-Tree Expansion

A crucial aspect of the approach is the use of the roadmaps $\mathcal{R}_1, \ldots, \mathcal{R}_n$ to guide the motion-tree expansion. In fact, the roadmaps are used to partition $\mathcal{T}$ into equivalence classes. An equivalence class contains all the vertices in $\mathcal{T}$ that map to the same tuple of roadmap configurations. Specifically, a mapping function $\text{MAP} : \mathcal{S}_1 \times \ldots \times \mathcal{S}_n \to V_{\mathcal{R}_1} \times \ldots \times V_{\mathcal{R}_n}$ is defined from states to roadmap configurations as

$$\text{MAP}(\langle s_1, \ldots, s_n \rangle) = \langle \text{MAP}_1(s_1), \ldots, \text{MAP}_n(s_n) \rangle \quad (9)$$

where each $\text{MAP}_i : \mathcal{S}_i \to V_{\mathcal{R}_i}$ is defined as

$$\text{MAP}_i(s) = \operatorname*{argmin}_{c \in V_{\mathcal{R}_i}} \rho_i(c, \text{CFG}(s)). \quad (10)$$

In other words, $s \in \mathcal{S}_i$ is mapped to the nearest configuration in $V_{\mathcal{R}_i}$. Using this mapping, $\langle c_1, \ldots, c_n \rangle \in V_{\mathcal{R}_1} \times \ldots \times V_{\mathcal{R}_n}$ defines the equivalence class

$$\Gamma_{\langle c_1, \ldots, c_n \rangle} = \{v : v \in V_\mathcal{T} \wedge \langle c_1, \ldots, c_n \rangle = \text{MAP}(v.\text{states})\}. \quad (11)$$

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

$$\Gamma = \{\Gamma_{\langle c_1, \ldots, c_n \rangle} : \langle c_1, \ldots, c_n \rangle \in V_{\mathcal{R}_1} \times \ldots \times V_{\mathcal{R}_n} \wedge |\Gamma_{\langle c_1, \ldots, c_n \rangle}| > 0\}. \quad (12)$$

As an implementation note, Γ is maintained as a hashmap indexed by $\langle c_1, \ldots, c_n \rangle$. When v_{new} is added to $\mathcal{T}$, its key is computed as $\text{MAP}(v_{\text{new}}.\text{states})$. If $\Gamma_{\text{MAP}(v_{\text{new}}.\text{states})} \notin \Gamma$, then it is created and added to Γ; otherwise, it is retrieved from Γ. In both cases, v_{new} is added to $\Gamma_{\text{MAP}(v_{\text{new}}.\text{states})}$.

The partition Γ allows us to leverage multi-agent graph search to guide the motion-tree expansion. Specifically, when an equivalence class $\Gamma_{\langle c_1, \ldots, c_n \rangle}$ is first created, MULTIROADMAPSEARCH$(\mathcal{R}_1, \ldots, \mathcal{R}_n, c_1, \ldots, c_n)$ is invoked to compute collision-free (but not necessarily dynamically feasible) roadmap paths $\sigma_1, \ldots, \sigma_n$ for each robot to

its goal. These paths are stored in the data structure representing $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ as $\Gamma_{\langle c_1,\ldots,c_n\rangle}.\mathrm{paths}$. The motion-tree expansion seeks to expand $\mathcal{T}$ from vertices in $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ along $\Gamma_{\langle c_1,\ldots,c_n\rangle}.\mathrm{paths}$. Preference is given to equivalence classes whose associated paths have lower cost than other equivalence classes, since expansions along those paths are more likely to quickly lead each robot to its goal. When the expansion fails to make progress, which could happen due to the geometric and differential constraints imposed by the obstacles and the dynamics, the approach penalizes $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ to promote expansions from other equivalence classes.

The procedures for selecting and expanding an equivalence class are invoked repeatedly until a solution is found or a runtime limit is reached. Pseudocode is shown in Alg. 1.

Selecting an Equivalence Class A weight is defined for each equivalence class and the equivalence class with the maximum weight is selected for expansion (Alg. 1:6). Specifically, the weight for $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ is defined as

$$w(\Gamma_{\langle c_1,\ldots,c_n\rangle}) = \frac{\alpha^{\Gamma_{\langle c_1,\ldots,c_n\rangle}.\mathrm{nrSel}}}{\sum_{i=1}^{n}(\mathrm{COST}(\Gamma_{\langle c_1,\ldots,c_n\rangle}.\mathrm{paths}[i]))^2}, \tag{13}$$

where $0 < \alpha < 1$ and $\Gamma_{\langle c_1,\ldots,c_n\rangle}.\mathrm{nrSel}$ denotes the number of times $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ has been previously selected.

In this way, preference is given to equivalence classes associated with low-cost roadmap paths. The parameter α is a penalty factor to avoid selecting the same equivalence class indefinitely. In fact, repeatedly selecting $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ will continue to reduce its weight so that eventually some other equivalence class will end up having a greater weight and thus be selected for expansion. This is essential to ensure probabilistic completeness and avoid becoming stuck when the motion-tree expansion from $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ repeatedly fails due to the geometric and differential constraints.

Expanding an Equivalence Class along Roadmap Routes After selecting $\Gamma_{\langle c_1,\ldots,c_n\rangle}$, the approach seeks to expand $\mathcal{T}$ along $\Gamma_{\langle c_1,\ldots,c_n\rangle}.\mathrm{paths}$ (Alg. 1:7–22). To expand $\mathcal{T}$ along $\sigma_i = \Gamma_{\langle c_1,\ldots,c_n\rangle}.\mathrm{paths}[i]$, a target configuration c_i^{target} for the i-th robot is first set by sampling a collision-free configuration near $\sigma_i[2]$ (since $\sigma_i[1] = c_i$). Sampling near $\sigma_i[2]$ as opposed to setting $c_i^{\mathrm{target}} = \sigma_i[2]$ is preferred to give more flexibility to the expansion, since the roadmap paths are not necessarily dynamically feasible. As such, forcing the robot to follow those paths exactly could be infeasible.

After setting the targets, attempts are made to expand $\mathcal{T}$ from the closest vertex v in $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ to $\langle c_1,\ldots,c_n\rangle$, i.e.,

$$v = \operatorname*{argmin}_{v' \in \Gamma_{\langle c_1,\ldots,c_n\rangle}} \sum_{i=1}^{n} \rho_i(c_i, \mathrm{CFG}(v'.\mathrm{states}[i])). \tag{14}$$

A collision-free and dynamically-feasible trajectory is generated from v toward $\langle c_1^{\mathrm{target}},\ldots,c_n^{\mathrm{target}}\rangle$ by applying control actions and integration the motion equations for several time steps, stopping when a collision is found. A proportional-integrative-derivative (PID) controller (Spong, Hutchinson, and Vidyasagar 2005) is used to select the control actions that steer the i-th robot toward c_i^{target} (Alg. 1:15). For a vehicle, the PID controller selects actions that turn the wheels and then move toward c_i^{target}. When the trajectory being expanded gets near the target (within some predefined distance), c_i^{target} is set by sampling a collision-free configuration near the next position in σ_i (Alg. 1:18). If the new state is in collision, the trajectory generation stops (Alg. 1:19), and the approach goes back to selecting an equivalence class. Otherwise, a new vertex is added to $\mathcal{T}$ (Alg. 1:20). The partition Γ is updated accordingly, as described earlier. If the new vertex represents states where each robot has reached its goal, then the algorithm terminates successfully since a solution is found (Alg. 1:21).

Algorithm 1 Pseudocode for the proposed approach

Input: bounding box $\mathcal{W}$; obstacles $\mathcal{O}$; robot models $\mathcal{M} = \{\mathcal{M}_1,\ldots,\mathcal{M}_n\}$, $\mathcal{M}_i = \langle \mathcal{P}_i, \mathcal{S}_i, \mathcal{A}_i, f_i, s_i^{\mathrm{init}}, \mathcal{G}_i\rangle$; 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 initial to goal; or $\perp$ if no solution

1: **for** $i = 1 \ldots n$ **do**
2: $\quad \mathcal{R}_i = (V_{\mathcal{R}_i}, E_{\mathcal{R}_i}, \mathrm{COST}_{\mathcal{R}_i}) \leftarrow \textsc{Roadmap}(\mathcal{C}_i, \mathcal{P}_i, s_i^{\mathrm{init}}, \mathcal{G}_i)$
3: $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}}) \leftarrow (\emptyset, \emptyset)$; $\quad \Gamma \leftarrow \emptyset$
4: $\textsc{AddVertexAndUpdateEQC}(\mathcal{T}, \Gamma, s_1^{\mathrm{init}}, \ldots, s_n^{\mathrm{init}}, \mathrm{parent} \leftarrow \perp, a_1 \leftarrow \perp, \ldots, a_n \leftarrow \perp)$
5: **while** $\textsc{time}() < t_{\max}$ **do**
6: $\quad \Gamma_{\langle c_1,\ldots,c_n\rangle} \leftarrow \textsc{SelectEQC}(\Gamma)$
7: $\quad$ **for** $i = 1 \ldots n$ **do**
8: $\quad\quad \sigma_i \leftarrow \Gamma_{\langle c_1,\ldots,c_n\rangle}.\mathrm{paths}[i]$
9: $\quad\quad c_i^{\mathrm{target}} \leftarrow \textsc{FirstTarget}(\sigma_i)$
10: $\quad v \leftarrow \textsc{SelectVertex}(\Gamma_{\langle c_1,\ldots,c_n\rangle}, c_1^{\mathrm{target}}, \ldots, c_n^{\mathrm{target}})$
11: $\quad$ **for** several steps **do**
12: $\quad\quad$ solved $\leftarrow$ `true`
13: $\quad\quad$ **for** $i = 1 \ldots n$ **do**
14: $\quad\quad\quad s_i \leftarrow v.\mathrm{state}[i]$
15: $\quad\quad\quad a_i \leftarrow \textsc{controller}(s_i, c_i^{\mathrm{target}})$
16: $\quad\quad\quad s_i^{\mathrm{new}} \leftarrow \textsc{simulate}(s_i, a_i, f_i, dt)$
17: $\quad\quad\quad$ **if** $\textsc{pos}(s_i^{\mathrm{new}}) \notin \mathcal{G}_i$ **then** solved $\leftarrow$ `false`
18: $\quad\quad\quad$ **if** $\textsc{near}(s_i^{\mathrm{new}}, c_i^{\mathrm{target}})$ **then** $c_i^{\mathrm{target}} \leftarrow \textsc{NextTarget}(\sigma_i)$
19: $\quad\quad$ **if** $\textsc{collision}(s_1^{\mathrm{new}}, \ldots, s_n^{\mathrm{new}})$ **then break**
20: $\quad\quad v_{\mathrm{new}} \leftarrow \textsc{AddVertexAndUpdateEQC}(\mathcal{T}, \Gamma, s_1^{\mathrm{new}}, \ldots, s_n^{\mathrm{new}}, v, a_1, \ldots, a_n)$
21: $\quad\quad$ **if** solved **then return** $\langle \zeta_1, \ldots, \zeta_n\rangle \leftarrow \textsc{trajs}(\mathcal{T}, v_{\mathrm{new}})$
22: $\quad\quad v \leftarrow v_{\mathrm{new}}$
23: **return** $\perp$

$\textsc{AddVertexAndUpdateEQC}(\mathcal{T}, \Gamma, s_1^{\mathrm{new}}, \ldots, s_n^{\mathrm{new}}, v, a_1, \ldots, a_n)$
1: $v_{\mathrm{new}} \leftarrow$ new motion-tree vertex
2: $v_{\mathrm{new}}.\mathrm{parent} \leftarrow v$
3: $v_{\mathrm{new}}.\mathrm{states} \leftarrow \langle s_1^{\mathrm{new}}, \ldots, s_n^{\mathrm{new}}\rangle$
4: $v_{\mathrm{new}}.\mathrm{actions} \leftarrow \langle a_1, \ldots, a_n\rangle$
5: $\langle c_1^{\mathrm{new}}, \ldots, c_n^{\mathrm{new}}\rangle \leftarrow \textsc{map}(\mathcal{R}_1, \ldots, \mathcal{R}_n, s_1^{\mathrm{new}}, \ldots, s_n^{\mathrm{new}})$
6: $\Gamma_{\langle c_1^{\mathrm{new}},\ldots,c_n^{\mathrm{new}}\rangle} \leftarrow \textsc{find}(\Gamma, \langle c_1^{\mathrm{new}}, \ldots, c_n^{\mathrm{new}}\rangle)$
7: **if** $\Gamma_{\langle c_1^{\mathrm{new}},\ldots,c_n^{\mathrm{new}}\rangle} = \perp$ **then**
8: $\quad \Gamma_{\langle c_1^{\mathrm{new}},\ldots,c_n^{\mathrm{new}}\rangle} \leftarrow$ new equivalence class
9: $\quad \Gamma_{\langle c_1^{\mathrm{new}},\ldots,c_n^{\mathrm{new}}\rangle}.\mathrm{paths} \leftarrow \textsc{MultiRoadmapSearch}(\mathcal{R}_1, \ldots, \mathcal{R}_n, c_1^{\mathrm{new}}, \ldots, c_n^{\mathrm{new}})$
10: $\textsc{insert}(\Gamma_{\langle c_1^{\mathrm{new}},\ldots,c_n^{\mathrm{new}}\rangle}, v_{\mathrm{new}})$
11: $\textsc{insert}(\Gamma, \Gamma_{\langle c_1^{\mathrm{new}},\ldots,c_n^{\mathrm{new}}\rangle})$; **return** v_{new}

Runtime Analysis and Probabilistic Completeness

The construction of each roadmap $\mathcal{R}_i$ is dominated by collision checking and nearest-neighbors computations. Using sweep-and-prune algorithms, collision checking between the i-th robot and the obstacles runs in $O((n_{\mathcal{O}} + |\mathcal{P}_i|)\log(n_{\mathcal{O}} + |\mathcal{P}_i|))$ time, where $n_{\mathcal{O}} = \sum_{i=1} |\mathcal{O}_i|$ denotes the total number of vertices for the obstacles. Computing the k-nearest neighbors runs in $O(k \log |V_{\mathcal{R}_i}|)$ time by using efficient searches (Beygelzimer, Kakade, and Langford 2006). Determining a bound on the number of calls to collision checking and nearest neighbors remains open since it depends on the probability of generating collision-free configurations and the length of the edges. Hence, the analysis has focused on providing bounds on the probability of finding a solution with respect to the roadmap size (Kavraki et al. 1996; Ladd and Kavraki 2004; Karaman and Frazzoli 2011). Such analysis also applies to our roadmap constructions.

Below we discuss the runtime complexity of the motion-tree expansion (Alg. 1:5–22). SELECTEQC (Alg. 1:6) runs in $O(\log |\Gamma|)$ time. SELECTVERTEX (Alg. 1:10) runs in $O(\log |V_{\mathcal{T}}|)$ time. SIMULATE (Alg. 1:16) runs in $O(d)$ time, where d is the maximum number of the state variables for $\mathcal{S}_1, \ldots, \mathcal{S}_n$, since it uses Runge-Kutta to integrate the motion equations. COLLISION (Alg. 1:19) runs in $O(n_{\mathcal{O}} \log n_{\mathcal{O}} + n\, n_{\mathcal{P}} \log n_{\mathcal{P}})$ time, where $n_{\mathcal{O}} = \sum_{i=1} |\mathcal{O}_i|$ and $n_{\mathcal{P}} = \sum_{i=1}^{n} |\mathcal{P}_i|$. ADDVERTEXANDUPDATEEQC (Alg. 1:4,20) invokes MAP and MULTIROADMAPSEARCH. MAP runs in $O(\sum_{i=1}^{n} \log |V_{\mathcal{R}_i}|)$ time. The complexity of MULTIROADMAPSEARCH depends on the particular multi-agent graph search method being used. Since each composite edge in MULTIROADMAPSEARCH invokes COLLISION for checking robot-robot collisions, the runtime of expanding a node is $O(n\, n_{\mathcal{P}} \log n_{\mathcal{P}})$. Using an A* variant gives $O((b^*)^k)$ on the number of nodes expanded, denoted by $n_{\mathrm{NodesRmSearch}}$, where b^* is the effective branching factor and k is the minimum length of the solution.

Let n_{MP} denote the number of times the motion-tree expansion is invoked, i.e., entering the while loop. Then, SELECTEQC and SELECTVERTEX are invoked n_{MP} times. Since the number of control steps (Alg. 1:11) is bounded by a user-defined constant, COLLISION and MAP are invoked $O(n_{\mathrm{MP}})$ times and SIMULATE is invoked $O(n\, n_{\mathrm{MP}})$ times. MULTIROADMAPSEARCH is invoked once per equivalence class, so $|\Gamma|$ times. Putting it all together, and using the fact that $|\Gamma| \leq |V_{\mathcal{T}}|$, the runtime complexity is

$$O(n_{\mathrm{MP}}(\log |V_{\mathcal{T}}| + n\, d + n_{\mathcal{O}} \log n_{\mathcal{O}} + \sum_{i=1}^{n} \log |V_{\mathcal{R}_i}| + n_{\mathrm{NodesRmSearch}}\, n\, n_{\mathcal{P}} \log n_{\mathcal{P}})). \quad (15)$$

Bounding n_{MP} remains an open problem. The proposed approach, however, is probabilistically complete, which guarantees that a solution will be found, when it exists, with probability approaching one. The claim follows from the analysis in (Plaku 2015) which can be applied to sampling-based motion planners that use discrete search to guide the motion-tree expansion. Although the analysis in (Plaku 2015) is presented for a single robot, it still applies in our case since $\mathcal{T}$ is expanded over the composite state space $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$. Thus, multiple robots, for the purposes of the analysis, are considered as one system operating in $\mathcal{S}$.

Experiments and Results

Experiments are conducted with second-order vehicle models operating in complex environments (Fig. 1 and 4), where cooperation among the robots is often required to find solutions. Experiments compare the approach to related work and evaluate the planning runtime as the number of robots is increased. The robot models, scenes, and the proposed approach are made publicly available (Release 2017).

Problem Instances A problem instance is defined by a scene and the number of robots. For a given scene and number of robots n, 60 problem instances were generated by randomly placing the robots and the goals. To make the test cases more challenging, rather than sampling from $\mathcal{W}$, the initial states and goals were sampled to be inside certain manually-selected areas. Fig. 1 and 4 show some examples. Each multi-robot motion planner is run on each of the problem instances. Results report the runtime and solution cost after dropping the best and worst five runs to avoid the influence of the outliers. The runtime includes everything from reading the input until finding a solution. Solution cost is measured as the distance traveled by the robots. Experiments were run on an Intel Core i7 (1.90GHz).

Comparisons to Related Work The proposed approach is compared to a centralized and a prioritized version of `RRT` (LaValle and Kuffner 2001), denoted by `cRRT` and `pRRT`. The implementations use goal bias, efficient nearest-neighbor search, and multi-step expansions, as advocated in the literature. We also compare to a prioritized version of `GUST` (Plaku 2015), denoted by `pGUST`. `GUST` was selected due to its computational efficiency, which derives from using discrete search to guide the motion-tree expansion. We could only use a prioritized version of `GUST`, since, as mentioned in the discussion of related work, it remains open to effectively extend `GUST` as a centralized approach.

Results in Fig. 5 show that our approach is significantly faster than `cRRT`, `pRRT`, and `pGUST`. The runtime improvements become more prominent as the number of robots is increased and on the scenes where cooperation among the robots is required to find solutions. The runtime of `cRRT` degrades as the number of robots is increased. As `cRRT` lacks global guidance, the high-dimensionality of the composite state space makes it difficult for `cRRT` to effectively expand the motion tree. `pRRT` performs better than `cRRT` since it operates over the individual state spaces. `pGUST` is faster than `pRRT` since it uses discrete search to guide the motion-tree expansion. In open scenes (scene 2), where cooperation is not essential, `pGUST` is faster than our approach as the robot trajectories do not interfere with each other. `pRRT` and `pGUST`, however, as prioritized approaches, have difficulty finding solutions when a previously planned trajectory prevents the next robot from reaching its goal. This is prevalent in scenes 1, 3, and especially 4, 5, 6, where cooperation is required to find a solution. Our approach shows remarkable efficiency in solving these complex problems. In fact, `cRRT`,

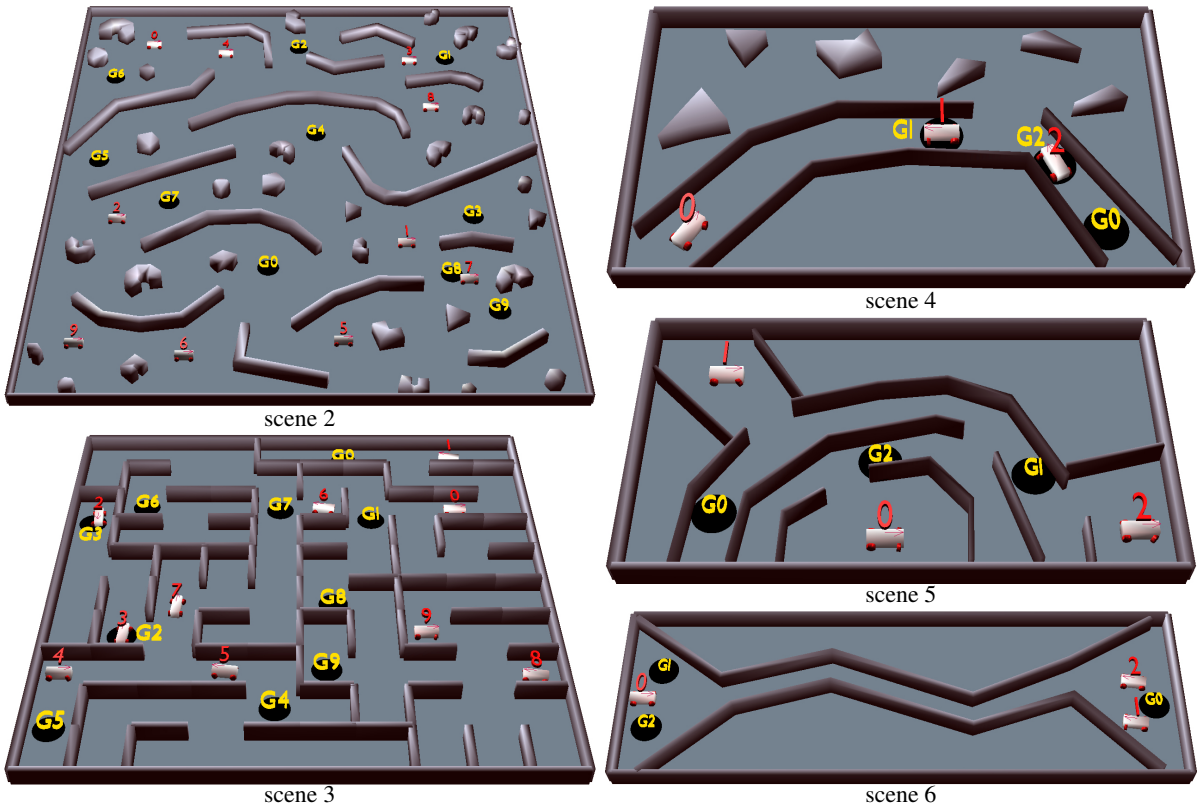

Figure 4: Scenes used in the experiments (scene 1 shown in Fig. 1). Videos of solutions obtained by our approach on these scenes can be found at https://goo.gl/sQ6irb. Figure best viewed in color and on screen.

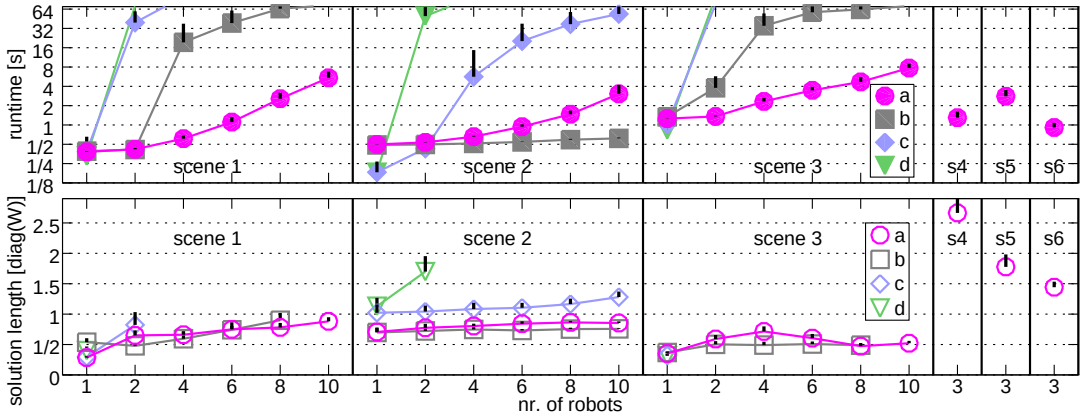

Figure 5: Runtime and solution-length results when comparing (a) our approach to (b) pGUST, (c) pRRT, and (d) cRRT. Runtime measures everything from reading the input file to reporting that a solution is found (including for our approach the time to build the roadmaps). Missing entries indicate failure by the planner to solve the problem instances within the runtime limit (set to 90s per run). Note in particular that only our approach could solve scenes 4, 5, 6 (denoted as s4, s5, s6 in the figure). Solution length is scaled by dividing it by the length of the diagonal of $\mathcal{W}$. Each bar indicates one standard deviation.

pRRT, and pGUST all timed out in scenes 4, 5, 6 (set to 90s per run), while our approach found solutions in 1-3s. This is due to using routes obtained by MULTIROADMAPSEARCH to effectively guide the motion-tree expansion.

Results in Fig. 5 also show that our approach is able to find shorter solutions than cRRT or pRRT. This is again due to using MULTIROADMAPSEARCH to guide the motion-tree expansion. Since cRRT and pRRT lack global guidance, the obtained solutions tend to be long. pGUST uses discrete search to guide the motion-tree expansion, so it finds so-

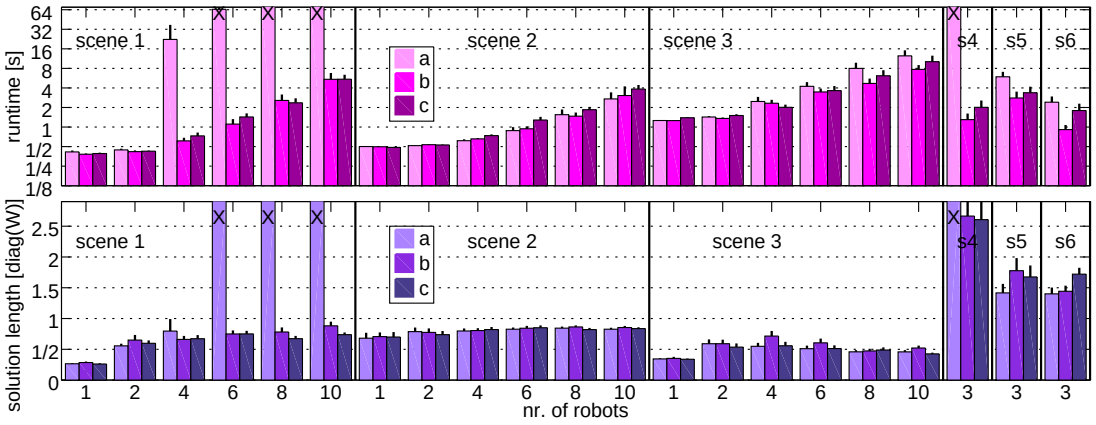

Figure 6: Runtime and solution-length results of our overall approach when varying the window-size parameter of WHCA*(Silver 2005), which was used as MULTIROADMAPSEARCH. Experiments were run with window sizes of (a) 2, (b) 5, and (c) 10. Entries marked with X indicate failure to find a solution within the runtime limit (90s per run).

lutions in similar length as our approach. As a note, under certain assumptions, in the single robot case, there are sampling-based motion planners that guarantee probabilistic optimality (Karaman and Frazzoli 2010; Li, Littlefield, and Bekris 2016), but it comes at a significant increase in the runtime cost due to the rewiring of the motion-tree branches.

Impact of MULTIROADMAPSEARCH Fig. 6 shows the results when varying the window-size parameter of WHCA*(Silver 2005), which we used to implement MULTIROADMAPSEARCH. The window size can be thought of as the number of look-ahead moves to resolve conflicts. A small window size works well in open scenes but can have a hard time resolving conflicts in scenes where robots have to exchange places over a long and narrow passage. A large window size causes WHCA* to precalculate larger proportions of the routes. Even though it reduces the number of reroutings, each rerouting has higher cost. Results in Fig. 6 show that the overall planning runtime is reduced when the window size is neither too small nor too large. It is an interesting problem to find an optimal window size or to dynamically adjust the window size to reduce the planning runtime. As shown in Fig. 6, the window size has less of an impact on the length of the solution trajectories.

Runtime Distribution Fig. 7 shows the runtime distribution for various components of our approach. Note that MULTIROADMAPSEARCH takes a significant portion. This indicates that our approach is able to shift the load from the motion-tree expansion, which is slow as it is over the composite state space, to multi-agent pathfinding over graphs. In other words, routes obtained by MULTIROADMAPSEARCH effectively guide the motion-tree expansion so that it spends little time exploring parts of the composite state space that are not needed to find a solution.

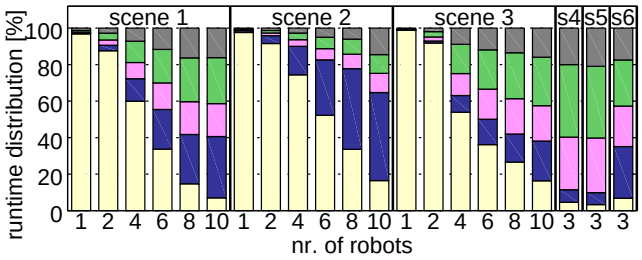

Figure 7: Runtime distribution as a percentage of the total runtime for the various components of our approach (from bottom to top): (a) create roadmaps (Alg. 1:1-2), (b) MULTIROADMAPSEARCH (Alg. 1:20), (c) SIMULATE (Alg. 1:16), (d) COLLISION (Alg. 1:19), and (e) other.

Discussion

This paper developed an effective, cooperative, and probabilistically-complete multi-robot motion planner that took into account the geometric and differential constraints imposed by the obstacles and the robot dynamics. The premise of this work was that the motion-tree expansion in the composite state space could be effectively guided by using solutions in a simplified geometric setting that did not take dynamics into account. The heuristic solutions were obtained by constructing roadmaps over low-dimensional configuration spaces and relying on cooperative multi-agent graph search to effectively find graph routes.

This work opens up several research directions. Advances in multi-agent pathfinding over graphs can directly increase the efficiency and scalability of the framework by reducing the runtime and improving the quality of the guides. In the other direction, feedback from the motion-tree expansion can be used by multi-agent pathfinders to avoid or prune routes that are deemed infeasible. Other research directions include incorporating high-level planning formalisms to enable a team of robots to perform complex tasks.

Acknowledgements

This work is supported by NSF IIS-1449505 and NSF IIS-1548406.

References

Amir, O.; Sharon, G.; and Stern, R. 2015. Multi-agent pathfinding as a combinatorial auction. In *AAAI National Conference on Artificial Intelligence*, 2003–2009.

Beygelzimer, A.; Kakade, S.; and Langford, J. 2006. Cover trees for nearest neighbor. In *Interantional Conference on Machince Learning*, 97–104.

Branicky, M. S. 1995. Universal computation and other capabilities of continuous and hybrid systems. *Theoretical Computer Science* 138(1):67–100.

Cheng, P.; Frazzoli, E.; and LaValle, S. 2008. Improving the performance of sampling-based motion planning with symmetry-based gap reduction. *IEEE Transactions on Robotics* 24(2):488–494.

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

Şucan, I. A., and Kavraki, L. E. 2012. A sampling-based tree planner for systems with complex dynamics. *IEEE Transactions on Robotics* 28(1):116–131.

Erdem, E.; Kisa, D. G.; Öztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI National Conference on Artificial Intelligence*, 290–296.

Jansen, M. R., and Sturtevant, N. R. 2008. Direction maps for cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 185–190.

Karaman, S., and Frazzoli, E. 2010. Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE Conference on Decision and Control*, 7681–7687.

Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30(7):846–894.

Kavraki, L. E.; Švestka, P.; Latombe, J. C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

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

Kiesel, S.; Burns, E.; and Ruml, W. 2012. Abstraction-guided sampling for motion planning. In *Symposium on Combinatorial Search*, 162–163. Also as UNH CS Technical Report 12-01.

Ladd, A. M., and Kavraki, L. E. 2004. Measure theoretic analysis of probabilistic path planning. *IEEE Transactions on Robotics and Automation* 20(2):229–242.

Larsen, E.; Gottschalk, S.; Lin, M. C.; and Manocha, D. 1999. Fast proximity queries with swept sphere volumes. Tr99-18, Department of Computer Science, University of N. Carolina, Chapel Hill.

LaValle, S. M., and Kuffner, J. J. 2001. Randomized kinodynamic planning. *International Journal of Robotics Research* 20(5):378–400.

Le, D., and Plaku, E. 2014. Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 212–217.

Li, Y.; Littlefield, Z.; and Bekris, K. E. 2016. Asymptotically optimal sampling-based kinodynamic planning. *International Journal of Robotics Research* 35:528–564.

Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2010. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Transactions on Robotics* 26(3):469–482.

Plaku, E. 2015. Region-guided and sampling-based tree search for motion planning with dynamics. *IEEE Transactions on Robotics* 31:723–735.

Reif, J. 1979. Complexity of the mover's problem and generalizations. In *IEEE Symposium on Foundations of Computer Science*, 421–427.

Release, P. 2017. Supplementary material. Author names and web link suppressed to maintain anonimity during the review process.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Silver, D. 2005. Cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 1, 117–122.

Solovey, K.; Salzman, O.; and Halperin, D. 2015. Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. In *Algorithmic Foundations of Robotics*. 591–607.

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

Wagner, G.; Kang, M.; and Choset, H. 2012. Probabilistic path planning for multiple robots with subdimensional expansion. In *IEEE International Conference on Robotics and Automation*, 2886–2892.