Journal of Artificial Intelligence Research 63 (2018) 361-390 Submitted 03/2018; published 10/2018

Cooperative, Dynamics-Based, and Abstraction-Guided Multi-robot Motion Planning

Duong Le 05le@cua.edu
Erion Plaku plaku@cua.edu
Catholic University of America
Department of Electrical Engineering and Computer Science
Washington, DC 20064 USA

Abstract

This paper presents an effective, cooperative, and probabilistically-complete multi-robot motion planner that enables each robot to move to a desired location while avoiding collisions with obstacles and other robots. The approach takes into account not only the geometric constraints arising from collision avoidance, but also the differential constraints imposed by the motion dynamics of each robot. This makes it possible to generate collision-free and dynamically-feasible trajectories that can be executed in the physical world.

The salient aspect of the approach is the coupling of sampling-based motion planning to handle the complexity arising from the obstacles and robot dynamics with multi-agent search to find solutions over a suitable discrete abstraction. The discrete abstraction is obtained by constructing roadmaps to solve a relaxed problem that accounts for the obstacles but not the dynamics. Sampling-based motion planning expands a motion tree in the composite state space of all the robots by adding collision-free and dynamically-feasible trajectories as branches. Efficiency is obtained by using multi-agent search to find non-conflicting routes over the discrete abstraction which serve as heuristics to guide the motion-tree expansion. When little or no progress is made, the routes are penalized and the multi-agent search is invoked again to find alternative routes. This synergistic coupling makes it possible to effectively plan collision-free and dynamically-feasible motions that enable each robot to reach its goal. Experiments using vehicle models with nonlinear dynamics operating in complex environments, where cooperation among robots is required, show significant speedups over related work.

1. Introduction

Multi-robot systems provide a viable venue to enhance automation so as to increase productivity and reduce operational costs in an increasing number of applications ranging from exploration, inspection, surveillance, search-and-rescue, to transportation. In these settings, as part of the overall operations, the robots are often required to move to different locations while avoiding collisions with obstacles and other robots. As a fundamental requirement to enhance the autonomy, the multi-robot system must possess a motion-planning framework that can efficiently generate feasible motion plans so that each robot safely reaches its goal.

Multi-robot motion planning, however, poses significant challenges. The robots often have to navigate in unstructured, obstacle-rich environments, and pass through numerous narrow passages in order to reach their desired locations. Figure 1 shows an example. In such settings, cooperation among robots is often required so that they can avoid deadlock when preventing each other from reaching the corresponding destinations. Moreover, the

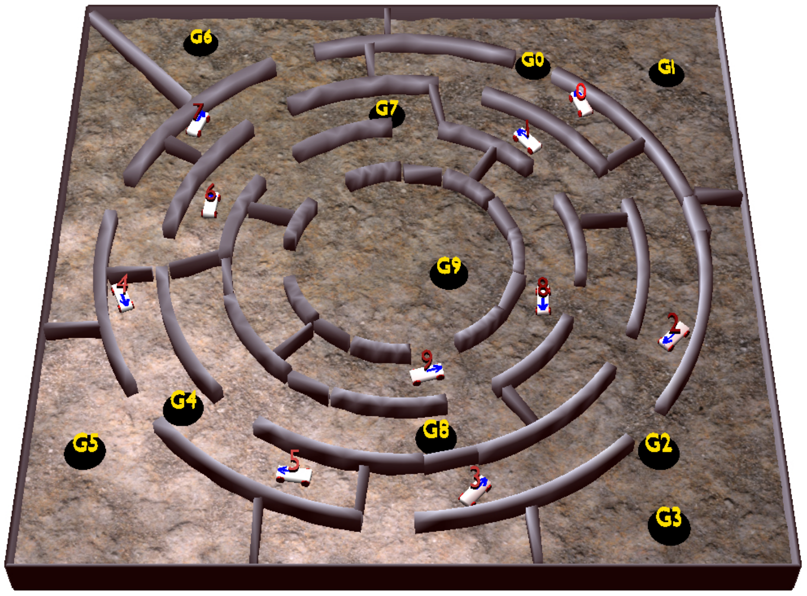


Figure 1: An example of a multi-robot motion-planning problem, where each robot is required to reach its goal ($\mathcal{G}_i$ for robot i) while avoiding collisions with the obstacles and the other robots. The planned trajectories must also obey the differential constraints imposed by the underlying robot dynamics. A physics game engine is used as the underlying simulator. In this and many other scenarios, cooperation among the robots is essential to reach the goals. Videos of solutions obtained by our approach on this and other scenes can be found at http://goo.gl/8muxwC. Figure best viewed in color and on screen.

robot motions are governed by their underlying dynamics. Robot dynamics express physical constraints on the feasible motions, such as ensuring a minimum turning radius, bounding the velocity, acceleration, and steering angle, or preventing the wheels from sliding sideways. Such constraints are generally expressed as a set of differential equations, which indicate how the robot moves as a result of applying external control inputs. As an example, differential equations for a vehicle model indicate how the position, orientation, and velocity change as a result of turning the steering wheel, accelerating, or braking. The differential equations, however, due to the complexity of the robotic systems, are generally nonlinear and high-dimensional. Moreover, differential constraints often lead to nonholonomic systems due to the controllable degrees of freedom being less than the total degrees of freedom. This is challenging since motion planning even for one robot is PSPACE-complete (Reif, 1979) when considering only collision avoidance and becomes undecidable when taking into account also the differential constraints imposed by the underlying robot dynamics (Branicky, 1995).

To develop an effective multi-robot motion planner, this paper couples the ability of sampling-based motion planning to handle the complexity arising from the obstacles and robot dynamics with the ability of multi-agent search to find solutions over a suitable discrete abstraction. To account for the dynamics, sampling-based motion planning expands a motion tree in the composite continuous state space of all the robots by incrementally adding collision-free and dynamically-feasible trajectories as branches. The expansion in the composite state space is essential to guarantee probabilistic completeness, which ensures that when solutions exist, one will be found with probability approaching one.

As the dimensionality of the composite state space increases with the number of robots, the challenge is how to effectively guide the motion-tree expansion. This is where multi-agent search becomes vital. Conceptually, the idea is to use multi-agent search as a heuristic to guide sampling-based motion planning as it expands the motion tree. As with any heuristic, a critical aspect is the design of an appropriate discrete abstraction over a suitable relaxed problem setting. We propose to obtain the discrete abstraction by constructing roadmaps over low-dimensional configuration spaces to solve a relaxed problem that accounts for the obstacles but not the robot dynamics. The objective is to capture the connectivity of the environment through a network of collision-free roads that would make it easy to reach the goals from any location.

A crucial aspect of the approach, termed `CoSMMAS` (Cooperative Sampling-based Multi-robot motion planning and Multi-agent Search), is that sampling-based motion planning and multi-agent search work in tandem. Each iteration of a core loop first relies on multi-agent search to find non-conflicting routes over the discrete abstraction. Afterwards, sampling-based motion planning seeks to expand the motion tree along these routes. When the obstacles and robot dynamics make it difficult to progress, the current routes are penalized so that multi-agent search can provide alternative routes to reach the goals in the next iteration. This is made possible by developing a mapping from the composite state space onto the discrete abstraction and using it to partition the motion tree into equivalence classes. Conceptually, an equivalence class groups together all the vertices in the motion tree that provide the same information with respect to the discrete abstraction. This means that non-conflicting routes computed by multi-agent search for an equivalence class can be used to guide the motion-tree expansion from any vertex belonging to that equivalence class. This synergistic coupling of sampling-based motion planning over the composite state space and multi-agent search over the discrete abstraction makes it possible to effectively plan collision-free and dynamically-feasible motions that enable each robot to reach its goal.

Experiments using vehicle models with nonlinear dynamics operating in complex environments, where cooperation among robots is required, show significant speedups over related work. Figure 1 shows an example of such a challenging problem involving 10 robots (50 degrees-of-freedom). In addition to differential equations, our approach can also be used with physics game engines such as Bullet (Coumans, 2012) and ODE (Smith, 2006), which provide an increased level of realism by modeling general rigid-body dynamics, friction, terrains, and other interactions of the robot with the world, which cannot be easily described analytically. Also note that the approach is agnostic to the inner workings of multi-agent search, so it can be used with any available method. We conducted experiments using three different methods, namely WHCA*(Silver, 2005), SIPP(Narayanan, Phillips, & Likhachev, 2012), and Push-and-Swap (Luna & Bekris, 2011).

A preliminary version of this work appeared in ICAPS (Le & Plaku, 2017). This work provides a more comprehensive description, extended discussion of related work, and offers a more developed version of the initial algorithm, several algorithmic improvements, and extended experimental evaluation.

2. Related Work

Our approach brings together concepts from robotics and AI. In these fields, related work on multi-robot motion planning can be generally divided into three categories depending on whether the geometry of the robots and their underlying dynamics are taken into account during planning, i.e., (i) points over graphs where neither the geometric representations nor the robot dynamics are considered (Section 2.1), (ii) the obstacles and geometric shapes of the robots are taken into account but their dynamics are not (Section 2.2), and (iii) the obstacles, geometric shapes of the robots, and their dynamics are taken into account (Section 2.3).

2.1 Multi-agent Search over Graphs

In a discrete setting, as in multi-agent pathfinding, a graph abstraction is often used to represent the world. Each robot is treated as a point, without any dynamics, that in one step can move to an adjacent vertex. Non-conflicting routes ensure that each robot reaches its goal and that no vertex is ever occupied by more than one robot. To solve this NP-hard problem (Yu & LaValle, 2013), decoupled and centralized multi-agent pathfinding approaches have been proposed (Felner et al., (2017) provides a comprehensive survey).

Decoupled approaches often consider agents one at a time, ensuring that the path planned for agent i avoids conflicts with the paths already planned for agents $1, \ldots, i-1$. This makes them fast, as they avoid searching over the composite space of all the agents, but suboptimal. In this setting, WHCA* conducts a cooperative space-time search, using hierarchical heuristics to guide the search and a dynamic window to limit the search depth (Silver, 2005). Map abstractions (Sturtevant & Buro, 2006), subgraph substructures (Ryan, 2008), conflict-oriented windows (Bnaya & Felner, 2014), and numerous other strategies have been proposed over the years to improve the performance.

To provide optimality, centralized approaches operate over the composite space of all the agents. Examples include the increasing-cost tree search (Sharon, Stern, Goldenberg, & Felner, 2013), conflict-based search (Sharon, Stern, Felner, & Sturtevant, 2015), M* (Wagner & Choset, 2015), A* and its variants (Standley & Korf, 2011; Goldenberg, Felner, Stern, Sharon, Sturtevant, Holte, & Schaeffer, 2014; Svancara & Surynek, 2017), which seek to divide the agents into independent groups, avoid surplus nodes, dynamically change the dimensionality based on conflicts, or develop effective heuristics.

Rule-based approaches devise specific rules for how agents should move to reach their goals while avoiding conflicts, but often require special properties to hold on the underlying graph (Botea & Surynek, 2015; Khorshid, Holte, & Sturtevant, 2011). Push-and-swap introduces rules for pushing an agent to an empty location and for swapping the location of two agents (Luna & Bekris, 2011; Sajid, Luna, & Bekris, 2012). Push-and-rotate divides the graph into subgraphs and uses push, swap, and rotate to find solutions (Wilde, Ter Mors, & Witteveen, 2014). Push-and-rotate is a complete algorithm for multi-agent pathfinding problems in which there are at least two empty vertices. Multi-agent pathfinders have also used auctions (Amir, Sharon, & Stern, 2015), answer-set programming (Erdem, Kisa, Öztok, & Schueller, 2013), or safe intervals (Narayanan et al., 2012).

2.2 Multi-robot Path Planning with Geometric Constraints but no Dynamics

In a continuous setting, when considering the robot geometries but not the dynamics, sampling-based approaches have often been used to solve multi-robot path-planning problems. Planning takes place in the configuration space, which captures the geometric constraints, as it represents position, orientation, and other degrees of freedom, but not the differential constraints imposed by the robot dynamics. The underlying idea in sampling-based approaches is to capture the connectivity of the configuration space by sampling collision-free configurations and connecting configurations to one another with collision-free paths (Choset, Lynch, Hutchinson, Kantor, Burgard, Kavraki, & Thrun, 2005; LaValle, 2006). `PRM` methods do so by constructing a roadmap (Kavraki, Švestka, Latombe, & Overmars, 1996), while `RRT` approaches expand a tree (LaValle & Kuffner, 2001; LaValle, 2011).

For multi-robot path planning, sampling-based methods can be used in a centralized, decoupled, or prioritized framework. In a centralized framework, planning takes place in the composite configuration space as all the robots are treated as one system. As a result, any sampling-based approach can be used. Centralized approaches provide probabilistic completeness but do not scale well due to the high-dimensionality of the composite configuration space. To improve the scalability of `PRM`, rather than explicitly constructing the roadmap in the composite configuration space, the composite roadmap is maintained implicitly as a product of the individual roadmaps in each configuration space (Solovey, Salzman, & Halperin, 2016). Our approach also leverages the idea of the implicit composite roadmap when constructing the discrete abstraction. The approach of Solovey, Salzmane, & Halperin (2016) does not take dynamics into account, so it ends by using graph search over the implicit roadmap. In distinction, our approach takes dynamics into account and uses graph paths over the implicit roadmap as heuristics to guide sampling-based motion planning.

Decoupled approaches plan the robot paths separately. Attempts are then made to coordinate the paths, e.g., via velocity tuning (Choset et al., 2005), or to repair the paths, e.g., via subdimensional expansion (Wagner, Kang, & Choset, 2012).

Prioritized[1] 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 can be fast but do not guarantee completeness, since coordination is not always possible and previously planned paths may make it impossible for the next robot to reach its destination.

2.3 Multi-robot Motion Planning with Geometric and Differential Constraints

When considering the robot geometries and the dynamics, sampling-based motion planners, e.g., `RRT` (LaValle & Kuffner, 2001; LaValle, 2011), `KPIECE` (Sucan & Kavraki, 2012), `GUST` (Plaku, 2015), often expand a motion tree by adding collision-free and dynamically-feasible trajectories as branches. Roadmaps cannot generally be used since each roadmap edge requires solving differential two-boundary value problems in order to generate a dynamically-feasible trajectory that connects its two end states. Analytical solutions to two-boundary value problems are available only in limited cases, while numerical methods leave gaps and

1. The term prioritized is often used in robotics to refer to decoupled approaches that treat the robots one at a time, similar to cooperative approaches in multi-agent pathfinding.

are computationally expensive (Keller, 1992; Cheng, Frazzoli, & LaValle, 2008). In contrast, the motion tree avoids two-boundary value problems since each branch is generated by applying control actions and numerically integrating the differential equations of motion.

As discussed, sampling-based motion planners, including those that expand a motion tree, can often 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 significance increase in the planning runtime as the number of robots is increased.

Our approach leverages the notion of using discrete search to guide the motion-tree expansion (Plaku, Kavraki, & Vardi, 2010; Plaku, 2015). These approaches, however, have been designed for a single robot. While it is possible to use them in a decoupled or prioritized framework, it remains open to develop a centralized version due to the coupling of discrete search and sampling-based motion planning. Moreover, as other sampling-based approaches that are not specifically designed for multiple robots, the performance tends to degrade rapidly as the multi-robot problems become more challenging. In contrast, our approach is specifically designed for multi-robot motion planning, leveraging multi-agent search as a heuristic to effectively guide the motion-tree expansion.

The proposed approach, as it is common in sampling-based motion planning, assumes a known map of the environment, including the geometry and placement of the obstacles. When a map is not available or when executed on 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 in a replanning framework as other sampling-based motion planners.

3. Problem Formulation

This section defines the robot models, including their underlying dynamics, motion trajectories, and the multi-robot motion-planning problem.

3.1 Robot Models and Underlying Dynamics

Each robot model is defined as a tuple $\mathcal{R}_i = \langle \mathcal{P}_i, \mathcal{S}_i, \mathcal{A}_i, f_i \rangle$ in terms of its geometric shape $\mathcal{P}_i$, state space $\mathcal{S}_i$, action space $\mathcal{A}_i$, and motion equations f_i.

The geometric shape $\mathcal{P}_i$ is used to ensure that the motions planned for robot $\mathcal{R}_i$ avoid collisions with the obstacles and the other robots.

The state space $\mathcal{S}_i$ is represented by a finite set of continuous variables. A robot state $s \in \mathcal{S}_i$ corresponds to an assignment of values to these variables. From a motion-planning perspective, the robot state often includes position, orientation, steering angle, and velocity. To facilitate presentation, the notations $\textsc{position}(s)$ and $\textsc{orientation}(s)$ are used to denote the position and orientation components of s.

A robot is controlled by applying external inputs, referred to as control actions. For a vehicle model, controls could include setting the acceleration and turning the steering wheel. The action space $\mathcal{A}_i$ is then defined as the set of all the control actions that can be applied to the robot. The control values are often bounded.

The motion equations f_i encapsulate the underlying robot dynamics by describing how the robot state changes as a result of applying control actions. The motion equations f_i are often expressed as a set of differential equations of the form

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

where $s \in \mathcal{S}_i$, $a \in \mathcal{A}_i$, and $\dot{s}$ is the derivative of s. The model allows for nonlinear equations with first, second, or higher order derivatives. Moreover, it allows for nonholonomic constraints, which are essential to model the underlying dynamics associated with robotic vehicles, since the controllable degrees of freedom are often less than the total degrees of freedom. A classical example of a nonholonomic system is a car that cannot move sideways.

Example: As an illustration, a vehicle can be modeled by defining its state as $s = (x, y, \theta, \psi, v)$ in terms of 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_i are defined as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos(\theta)\cos(\psi) \\ v\sin(\theta)\cos(\psi) \\ v\sin(\psi)/L \\ a_\omega \\ a_{\text{acc}} \end{bmatrix}, \tag{2}$$

where L is the distance between the back and front wheels. The shape $\mathcal{P}_i$ could represent the body of the vehicle and its wheels.

3.2 Dynamically-Feasible Trajectories

A trajectory $\zeta : [0, T] \rightarrow \mathcal{S}_i$ is a continuous function, parametrized by the duration $T \in \mathbb{R}^{\geq 0}$, that indicates the robot state at time $t \in [0, T]$. The trajectory ζ is dynamically feasible if it also satisfies the differential constraints imposed by the underlying robot dynamics, as specified by the motion equations f_i.

A dynamically-feasible trajectory $\zeta : [0, T] \rightarrow \mathcal{S}_i$ is obtained by starting at a state $s \in \mathcal{S}_i$ and applying a control function $\hat{a} : [0, T] \rightarrow \mathcal{A}_i$, where $\hat{a}(t)$ indicates the control actions applied at time $t \in [0, T]$. As a result of applying the control function $\hat{a}$, the robot state changes according to the motion equations f_i, giving rise to the trajectory ζ, i.e.,

$$\forall t \in [0, T] : \zeta(t) = s + \int_0^t f_i(\zeta(h), \hat{a}(h))dh. \tag{3}$$

Note that ζ is dynamically-feasible by construction since

$$\forall t : [0, T] : \dot{\zeta}(t) = f_i(\zeta(t), \hat{a}(t)). \tag{4}$$

From a computational aspect, the underlying dynamics and the effects of applying control actions are encapsulated by a function SIMULATE of the form

$$s_{\text{new}} \leftarrow \text{SIMULATE}(s, a, f_i, dt), \tag{5}$$

which computes the new robot state $s_{\text{new}} \in \mathcal{S}_i$, obtained by starting at the state $s \in \mathcal{S}_i$ and applying the control action $a \in \mathcal{A}_i$ for a time step $dt \in \mathbb{R}^{\geq 0}$. The function SIMULATE can be implemented using numerical integration, e.g., Runge-Kutta methods.

A control function $\hat{a}$ is often defined in terms of a sequence of control actions $\langle a_1, \ldots, a_\ell \rangle$, where each control action is applied for a time step dt. When starting from a state $s \in \mathcal{S}_i$ and applying $\langle a_1, \ldots, a_\ell \rangle$ in succession, a trajectory ζ is obtained as a sequence of states, where $\zeta(0) = s$ and, as j iterates from 1 to ℓ, the j-th state is computed as

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

3.3 Physics-Based Simulations

In addition to differential equations, our approach can also be used with physics game engines such as Bullet (Coumans, 2012) and ODE (Smith, 2006), which provide an increased level of realism by also modeling friction, gravity, terrains, and other interactions of the robot with the world, which cannot be easily described analytically. Physics game engines model general body dynamics and implement the function SIMULATE by computing the forces acting on the robot bodies and the new state resulting from applying those forces.

3.4 Environment and Collision Checking

The environment in which the robots $\mathcal{R}_1, \ldots, \mathcal{R}_n$ operate is represented by its bounding box $\mathcal{W}$, obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$, and goal regions $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$, where $\mathcal{O}_1, \ldots, \mathcal{O}_m, \mathcal{G}_1, \ldots, \mathcal{G}_n \subseteq \mathcal{W}$.

To generate collision-free motions, the robots $\mathcal{R}_1, \ldots, \mathcal{R}_n$ should avoid collisions with obstacles and each other. Collision checking depends on the shape of the robots and their placements in the environment. Given a robot state $s_i \in \mathcal{S}_i$, let $\textsc{placement}(\mathcal{P}_i, s_i)$ denote the placement of the shape $\mathcal{P}_i$ according to $\textsc{orientation}(s_i)$ and $\textsc{position}(s_i)$. Essentially, $\textsc{placement}(\mathcal{P}_i, s_i)$ is obtained by rotating $\mathcal{P}_i$ and then translating it. Collision checking is then encapsulated by a function $\textsc{collision} : \mathcal{S}_1 \times \ldots \times \mathcal{S}_n \rightarrow \{\texttt{true}, \texttt{false}\}$. Given a composite state $\langle s_1, \ldots, s_n \rangle \in \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$, where s_i denotes the state of the i-th robot, $\textsc{collision}(\langle s_1, \ldots, s_n \rangle) = \texttt{false}$ if and only if

- each robot is placed inside the bounding box $\mathcal{W}$, i.e.,

$$\bigcup_{i=1}^{n} \textsc{placement}(\mathcal{P}_i, s_i) \subseteq \mathcal{W} \tag{7}$$

- there is no robot-obstacle collision, i.e.,

$$\left(\bigcup_{i=1}^{n} \textsc{placement}(\mathcal{P}_i, s_i) \right) \cap \left(\bigcup_{j=1}^{m} \mathcal{O}_j \right) = \emptyset \tag{8}$$

- there is no robot-robot collision, i.e.,

$$\forall 1 \leq i < j \leq n : \textsc{placement}(\mathcal{P}_i, s_i) \cap \textsc{placement}(\mathcal{P}_j, s_j) = \emptyset \tag{9}$$

Collision-checking packages, such as PQP (Larsen, Gottschalk, Lin, & Manocha, 1999), FCL (Pan, Chitta, & Manocha, 2012), can be used to efficiently implement COLLISION.

3.5 Multi-robot Motion-Planning Problem

In multi-robot motion planning, the objective is to compute dynamically-feasible trajectories that enable each robot to reach a desired goal region while avoiding collisions with the other robots and the obstacles in the environment. Specifically, given

- robot models $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$, where $\mathcal{R}_i = \langle \mathcal{P}_i, \mathcal{S}_i, \mathcal{A}_i, f_i \rangle$ defines the i-th robot in terms of its shape $\mathcal{P}_i$, state space $\mathcal{S}_i$, action space $\mathcal{A}_i$, and motion equations f_i,
- environment in which the robots operate in terms of its bounding box $\mathcal{W}$, obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$, and goal regions $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$, and
- an initial state $\langle s_1^{\text{init}}, \ldots, s_n^{\text{init}} \rangle \in \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$, where $s_i^{\text{init}} \in \mathcal{S}_i$ denotes the initial state of robot $\mathcal{R}_i$,

the objective is to compute control functions $\hat{a}_1, \ldots, \hat{a}_n$ and the resulting dynamically-feasible trajectories $\zeta_1, \ldots, \zeta_n$, where $\hat{a}_i : [0, T] \to \mathcal{A}_i$ and $\zeta_i : [0, T] \to \mathcal{S}_i$ denote the control function and the trajectory for robot $\mathcal{R}_i$, such that

- each robot starts at the initial state and reaches its goal, i.e.,

$$\forall i \in \{1, \ldots, n\} : \zeta_i(0) = s_i^{\text{init}} \wedge \textsc{position}(\zeta_i(T)) \in \mathcal{G}_i, \tag{10}$$

- no robot-robot or robot-obstacle collisions occur, i.e.,

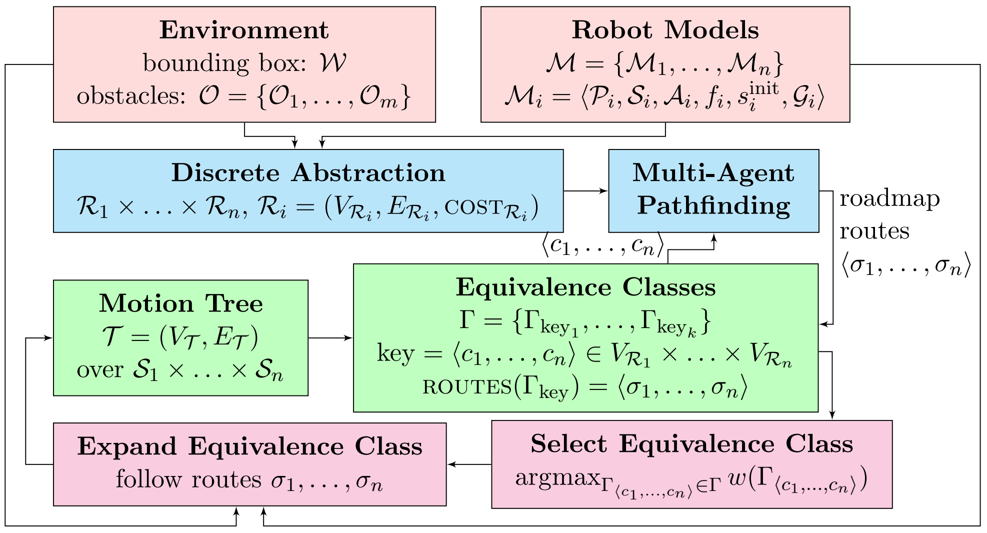
$$\forall t \in [0, T] : \textsc{collision}(\zeta_1(t), \ldots, \zeta_n(t)) = \texttt{false}. \tag{11}$$

4. Method

The approach has several components:

- a discrete abstraction based on a relaxed problem setting obtained by ignoring the underlying dynamics of each robot;
- multi-agent search over the discrete abstraction to find solutions in the relaxed problem setting that serve as heuristics to guide sampling-based motion planning;
- sampling-based expansion of a motion tree in the composite continuous state space of all the robots;
- partition of the motion tree into equivalence classes based on a mapping of the continuous state spaces onto the discrete abstraction; and
- interplay of sampling-based motion planning and multi-agent search to effectively expand the motion tree in the composite continuous state space of all the robots along non-conflicting routes obtained by the multi-agent search over the discrete abstraction.

A schematic representation is shown in Figure 2. The rest of the section describes these components in more detail.



Environment
bounding box: $\mathcal{W}$
obstacles: $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$
Robot Models
$\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$
$\mathcal{M}_i = \langle \mathcal{P}_i, \mathcal{S}_i, \mathcal{A}_i, f_i, s_i^{\text{init}}, \mathcal{G}_i \rangle$
Discrete Abstraction
$\mathcal{R}_1 \times \ldots \times \mathcal{R}_n$, $\mathcal{R}_i = (V_{\mathcal{R}_i}, E_{\mathcal{R}_i}, \text{COST}_{\mathcal{R}_i})$
Multi-Agent Pathfinding
roadmap routes $\langle \sigma_1, \ldots, \sigma_n \rangle$
$\langle c_1, \ldots, c_n \rangle$
Motion Tree
$\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$
over $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$
Equivalence Classes
$\Gamma = \{\Gamma_{\text{key}_1}, \ldots, \Gamma_{\text{key}_k}\}$
key $= \langle c_1, \ldots, c_n \rangle \in V_{\mathcal{R}_1} \times \ldots \times V_{\mathcal{R}_n}$
$\text{ROUTES}(\Gamma_{\text{key}}) = \langle \sigma_1, \ldots, \sigma_n \rangle$
Expand Equivalence Class
follow routes $\sigma_1, \ldots, \sigma_n$
Select Equivalence Class
$\text{argmax}_{\Gamma_{\langle c_1,\ldots,c_n\rangle} \in \Gamma}\, w(\Gamma_{\langle c_1,\ldots,c_n\rangle})$

Figure 2: Schematic illustration of the proposed approach.

4.1 Relaxed Problem Setting: Geometric Multi-robot Path Planning

The rationale for using a relaxed and simplified problem setting is that it can provide solutions that can serve as heuristics to guide the overall search in the composite state space of all the robots. As dynamics present a significant computational challenge, the simplified problem setting is obtained by ignoring the robot dynamics.

In this relaxed setting, each robot $\mathcal{R}_i$ retains its geometric shape $\mathcal{P}_i$ but its motions are no longer restricted by the differential equations f_i. In fact, each robot $\mathcal{R}_i$ can freely rotate and move in any direction. The notion of a robot state is replaced by the notion of a robot configuration, which represents only the position and orientation. For example, for the vehicle model described in Section 3.1, the configuration is defined as $c = \langle x, y, \theta \rangle$ in terms of the position (x, y) and orientation θ. Similarly, trajectories over the state space $\mathcal{S}_i$ are replaced with paths over the configuration space $\mathcal{C}_i$, where each path is described by rotations and translations.

The objective is then to compute paths so that each robot reaches its goal while avoiding collisions with the obstacles and the other robots. To solve the relaxed problem, first, a discrete abstraction is obtained by constructing roadmaps over the configuration spaces in order to provide routes along which the robots can avoid the obstacles. Second, multi-agent search is used over the roadmaps to find non-conflicting routes so that the robots also avoid each other. More details follow.

4.1.1 Discrete Abstraction via Roadmaps over Low-Dimensional Configuration Spaces

The discrete abstraction is obtained by constructing a roadmap $\mathcal{M}_i$ over each configuration space $\mathcal{C}_i$ as a network of roads, seeking to make it easy to reach the goal $\mathcal{G}_i$ from any location in the environment. Drawing from PRM (Kavraki et al., 1996), $\mathcal{M}_i$ is constructed by sampling collision-free configurations and connecting neighboring configurations with

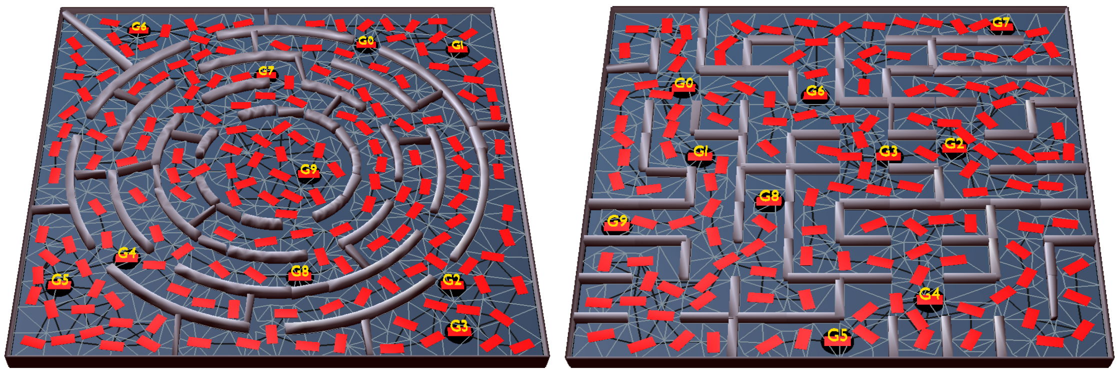


Figure 3: Examples of roadmaps and triangulation of the obstacle-free area $\mathcal{W}_{\text{free}}$.

collision-free paths whenever possible. Figure 3 shows an example. During the construction of $\mathcal{M}_i$ collision checking is done only between robot $\mathcal{R}_i$ and the obstacles.

The roadmap $\mathcal{M}_i = (V_{\mathcal{M}_i}, E_{\mathcal{M}_i}\textsc{cost}_{\mathcal{M}_i})$ is represented as an undirected, weighted, graph. Each $c \in V_{\mathcal{M}_i}$ corresponds to a collision-free configuration in $\mathcal{C}_i$. Each edge $(c', c'') \in E_{\mathcal{M}_i}$ corresponds to a collision-free path that connects c' and c''. The edge cost is defined as

$$\textsc{cost}_{\mathcal{M}_i}(c', c'') = \frac{\|\textsc{position}(c') - \textsc{position}(c'')\|_2}{\textsc{clearance}(\mathcal{O}, c', c'')}, \tag{12}$$

where $\textsc{clearance}(\mathcal{O}, c', c'')$ denotes the minimum distance from the obstacles in $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$ to the segment connecting $\textsc{position}(c')$ to $\textsc{position}(c'')$. By using clearance, edges that are close to the obstacles are assigned a higher cost so that minimum-cost paths to the goal are less likely to bring the robot close to the obstacles, which would make navigation more difficult. PQP (Larsen et al., 1999), FCL (Pan et al., 2012), and other collision-checking packages can be used to efficiently compute $\textsc{clearance}(\mathcal{O}, c', c'')$.

Pseudocode for constructing the roadmap $\mathcal{M}_i$ is shown in Alg. 1. The initial and goal configurations, denoted by c_i^{init} and c_i^{goal}, are first added to $V_{\mathcal{M}_i}$. The initial configuration is obtained from the initial state, i.e., $c_i^{\text{init}} = \langle \textsc{position}(s_i^{\text{init}}), \textsc{orientation}(s_i^{\text{init}}) \rangle$. 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.

Adding Configurations to the Roadmap: The roadmap is further populated by adding collision-free configurations. Each new configuration c added to the roadmap must maintain a minimum separation from the obstacles (denoted by $d_{\text{minSepCfgObs}}$) and from the nearest configuration in the roadmap (denoted by $d_{\text{minSepCfgs}}$). The minimum separation from the obstacles ensures that the roadmap configurations do not place the robot too close to the obstacles, making it easier for the motion-tree expansion (described later in the section) to follow the roadmap routes. The minimum separation from the nearest roadmap configuration ensures that roadmap paths that go through different configurations do not get too close to each other. This again is important during the multi-agent search so that alternative routes do not go through the same locations in the environment as previously explored routes. Efficient nearest-neighbors data structures are used to quickly compute the nearest roadmap configuration to c (Muja & Lowe, 2014).

Algorithm 1 Pseudocode for the construction of the roadmap $\mathcal{M}_i = (V_{\mathcal{M}_i}, E_{\mathcal{M}_i}, \textsc{cost}_{\mathcal{M}_i})$

Input: bounding box $\mathcal{W}$; obstacles $\mathcal{O}$; obstacle-free area $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus \bigcup_{j=1}^{m} \mathcal{O}_j$ and its triangulation; configuration space $\mathcal{C}_i$; robot shape $\mathcal{P}_i$; initial state s_i^{init}; goal $\mathcal{G}_i$

Output: $\mathcal{M}_i = (V_{\mathcal{M}_i}, E_{\mathcal{M}_i}, \textsc{cost}_{\mathcal{M}_i})$

```
// add initial and goal configurations to the roadmap
1: $c_i^{\text{init}} \leftarrow \langle \textsc{position}(s_i^{\text{init}}), \textsc{orientation}(s_i^{\text{init}}) \rangle$
2: $V_{\mathcal{M}_i} \leftarrow \{c_{\text{init}}\}$; $E_{\mathcal{M}_i} \leftarrow \emptyset$; attempts $\leftarrow \emptyset$
3: repeat $c_i^{\text{goal}} \leftarrow \langle \textsc{RandomPosition}(\mathcal{G}_i), \textsc{RandomOrientation}() \rangle$ until $\textsc{collision}(\mathcal{O}, c_i^{\text{goal}}) = \texttt{false}$
4: $V_{\mathcal{M}_i} \leftarrow V_{\mathcal{M}_i} \cup \{c_i^{\text{goal}}\}$
// populate the roadmap with configurations generated from the triangles in the triangulation of $\mathcal{W}_{\text{free}}$
5: $\Delta_{\text{init}} \leftarrow$ locate triangle in the triangulation of $\mathcal{W}_{\text{free}}$ that contains $\textsc{position}(c_i^{\text{init}})$
6: $\Delta_{\text{goal}} \leftarrow$ locate triangle in the triangulation of $\mathcal{W}_{\text{free}}$ that contains $\textsc{position}(c_i^{\text{goal}})$
7: $\mathcal{Q} \leftarrow$ create queue and insert $\Delta_{\text{init}}$ and $\Delta_{\text{goal}}$
8: visited $\leftarrow \{\Delta_{\text{init}}, \Delta_{\text{goal}}\}$
9: while $\textsc{empty}(\mathcal{Q}) = \texttt{false}$ do
10:    $\Delta \leftarrow \textsc{extract}(\mathcal{Q})$
11:    $\textsc{GenerateAndConnectCfg}(\mathcal{M}_i, \mathcal{O}, \Delta, \text{attempts})$
12:    if $\textsc{SameComponent}(\mathcal{M}_i, c_i^{\text{init}}, c_i^{\text{goal}}) = \texttt{true}$ then return $\mathcal{M}_i$
13:    for each $\Delta' \in \textsc{adjacent}(\Delta)$ and $\Delta' \notin$ visited do
14:       $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\Delta'\}$
15:       visited $\leftarrow$ visited $\cup \{\Delta'\}$
// populate the roadmap with configurations generated from $\mathcal{W}_{\text{free}}$
16: while $\textsc{SameComponent}(\mathcal{M}_i, c_i^{\text{init}}, c_i^{\text{goal}}) = \texttt{false}$ do
17:    $\textsc{GenerateAndConnectCfg}(\mathcal{M}_i, \mathcal{O}, \mathcal{W}_{\text{free}})$
18: return $\mathcal{M}_i$

(a) $\textsc{GenerateAndConnectCfg}(\mathcal{M}_i, \mathcal{O}, \mathcal{B}, \text{attempts})$
1: $c \leftarrow \textsc{GenerateCfg}(\mathcal{M}_i, \mathcal{O}, \mathcal{B})$
2: if $c \neq \texttt{null}$ then $\textsc{ConnectCfg}(\mathcal{M}_i, \mathcal{O}, c, \text{attempts})$

(b) $\textsc{GenerateCfg}(\mathcal{M}_i, \mathcal{O}, \mathcal{B})$
// $\mathcal{B}$: area from which to sample
1: ok $\leftarrow \texttt{false}$
2: for several times and ok $= \texttt{false}$ do
3:    $c \leftarrow \langle \textsc{RandomPosition}(\mathcal{B}), \textsc{RandomOrientation}() \rangle$
4:    ok $\leftarrow \textsc{collision}(\mathcal{O}, \mathcal{R}_i, c) = \texttt{false} \wedge \textsc{distance}(\mathcal{O}, c) \geq d_{\text{minSepCfgObs}} \wedge$
            $\textsc{distance}(c, \textsc{NearestNeighbor}(V_{\mathcal{M}_i}, c)) \geq d_{\text{minSepCfgs}}$
5: if ok $= \texttt{false}$ then return null
6: $V_{\mathcal{M}_i} \leftarrow V_{\mathcal{M}_i} \cup \{c\}$
7: return $c$

(c) $\textsc{ConnectCfg}(\mathcal{M}_i, \mathcal{O}, c, \text{attempts})$
1: neighs $\leftarrow \textsc{NearestNeighbors}(V_{\mathcal{M}_i}, c)$
2: for each $c_{\text{neigh}} \in$ neighs and $(c, c_{\text{neigh}}) \notin$ attempts do
3:    attempts $\leftarrow$ attempts $\cup \{(c, c_{\text{neigh}})\}$
4:    if $\textsc{CollisionFreePath}(\mathcal{O}, c, c_{\text{neigh}}) = \texttt{true}$ then
5:       $E_{\mathcal{M}_i} \leftarrow E_{\mathcal{M}_i} \cup \{(c, c_{\text{neigh}})\}$
6:       $\textsc{cost}(c, c_{\text{neigh}}) \leftarrow ||\textsc{position}(c) - \textsc{position}(c_{\text{neigh}})||_2 / \textsc{clearance}(\mathcal{O}, c, c_{\text{neigh}})$
```

The procedure for generating a configuration c that is added to the roadmap (Alg. 1:b) does so by repeatedly sampling a random orientation and a random position inside the obstacle-free area of the environment $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus \bigcup_{j=1}^{m} \mathcal{O}_j$ until the resulting robot placement is not in collision, the distance from c to the obstacles is at least $d_{\text{minSepCfgObs}}$, and the distance from c to its nearest roadmap configuration is at least $d_{\text{minSepCfgs}}$. When generating configurations, it is better to sample from $\mathcal{W}_{\text{free}}$ than $\mathcal{W}$ since only configurations from $\mathcal{W}_{\text{free}}$ could be added to the roadmap. The obstacle-free area $\mathcal{W}_{\text{free}}$ is computed efficiently using available triangulation packages (Shewchuk, 2002).

In order to improve sampling, attempts are made to generate configurations from the areas not too far away from existing roadmap configurations (Alg. 1:5–15). More precisely, the triangles in the triangulation of $\mathcal{W}_{\text{free}}$ that contain c_i^{init} and c_i^{goal} are added to a queue. Proceeding in a breadth-first manner, a triangle Δ is extracted from the queue, its unvisited neighbors are inserted into the queue, and attempts are made to generate an acceptable configuration inside Δ (using Alg. 1(b) GenerateCfg but sampling the position inside Δ as opposed to $\mathcal{W}_{\text{free}}$, and stopping after a number of attempts). If successful, the generated configuration is added to the roadmap. Attempts are made to connect to several of its neighbors via collision-free paths. This process is repeated until the queue becomes empty.

If after these steps, c_i^{init} and c_i^{goal} do not belong to the same roadmap component, additional configurations are added to the roadmap. At this time, configurations are generated from the entire $\mathcal{W}_{\text{free}}$ (Alg. 1:16–17).

Connecting the Roadmap Configurations: As mentioned, each time a configuration c is added to $V_{\mathcal{M}_i}$ attempts are made to connect it to several of its nearest neighbors via collision-free paths (Alg. 1(c)). Efficient nearest-neighbors data structures are used to quickly compute the nearest roadmap configuration to c (Muja & Lowe, 2014). For each neighbor c_{neigh}, the path connecting c to c_{neigh} is defined by a linear interpolation, i.e.,

$$\forall t \in [0,1] : \text{PATH}(c, c_{\text{neigh}}, t) = (1-t)c + t c_{\text{neigh}}. \tag{13}$$

If the path does not collide with the obstacles, then the edge (c, c_{neigh}) is added to $E_{\mathcal{M}_i}$. To check whether the path is in collision, several intermediate configurations $c_1, \ldots, c_k$ are generated along the path, where $\forall 1 \leq j \leq k : c_j \leftarrow \text{PATH}(c, c_{\text{neigh}}, (j+1)/k)$. The number k depends on the distance between c and c_{neigh}. It is generally set to $k = \lceil \|\text{POSITION}(c) - \text{POSITION}(c_{\text{neigh}})\|_2 / d_{\text{res}} \rceil$ to ensure that consecutive configurations are separated by at most a distance of d_{res}, where d_{res} is the desired collision-checking resolution. The robot is placed according to each of the configurations $c_1, \ldots, c_k$ and the resulting geometric shape is added to a mesh. In this way, the mesh will contain the area swept by the robot. If the mesh collides with the obstacles, the path is rejected. Computing the area swept by the robot results in faster collision checking (it is done only once) as opposed to individually checking for collision after each intermediate configuration.

The process of adding and connecting configurations to the roadmap is repeated until c_i^{init} and c_i^{goal} belong to the same roadmap connected component. A disjoint-set data structure is used to quickly determine the connected components. When it is possible to connect c_i^{init} to c_i^{goal} with collision-free paths, the probability that the roadmap does so approaches one rapidly, as shown by the probabilistic completeness of `PRM` (Kavraki et al., 1996).

Roadmap Sharing: Robots that have the same configuration space and shape can use the same roadmap. Specifically, let $\Psi = \{\Psi_1, \ldots, \Psi_k\}$ denote the partition of the robots $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ into sets, where each Ψ_j contains robots that have the same configuration space and shape. A roadmap for Ψ_j, denoted by $\mathcal{M}_{\Psi_j}$, is constructed by first adding the initial c_i^{init} and goal c_i^{init} configurations for each robot $\mathcal{R}_i \in \Psi_j$. The process of sampling and connecting configurations continues until each c_i^{init} belongs to the same roadmap component as c_i^{goal}. After $\mathcal{M}_{\Psi_j}$ is constructed, then $\mathcal{M}_i$ is set to point to $\mathcal{M}_{\Psi_j}$ for each robot $\mathcal{R}_i \in \Psi_j$. In particular, if all the robots are the same, then only one roadmap is constructed with $\mathcal{M}_1, \ldots, \mathcal{M}_n$ all pointing to it.

Note that an alternative approach to constructing individual roadmaps $\mathcal{M}_1, \ldots, \mathcal{M}_n$ over $\mathcal{C}_1, \ldots, \mathcal{C}_n$ would be to explicitly construct only one roadmap over the composite configuration space $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$. Such an approach would treat the robots as one system. However, due to the increased dimensionality, it imposes a significant computational cost, which renders the roadmap construction over $\mathcal{C}$ impractical (Choset et al., 2005, chap. 7).

4.1.2 Multi-agent Search over the Discrete Abstraction

After constructing the roadmaps $\mathcal{M}_1, \ldots, \mathcal{M}_n$, multi-agent search is used to compute non-conflicting routes where the robots avoid collisions with each other. Each roadmap $\mathcal{M}_i$ guarantees that robot $\mathcal{R}_i$ will not collide with the obstacles as it moves from one roadmap configuration to the next. Therefore, the multi-agent search has to avoid only robot-robot conflicts. Specifically, MultiAgentSearch($\mathcal{M}_1, \ldots, \mathcal{M}_n, c_1, \ldots, c_n, \mathcal{G}_1, \ldots, \mathcal{G}_n$) searches over $\mathcal{M}_1, \ldots, \mathcal{M}_n$ to compute non-conflicting routes $\sigma_1, \ldots, \sigma_n$ for each robot such that σ_i is over $\mathcal{M}_i$, starts at c_i, and ends at a roadmap configuration associated with the goal $\mathcal{G}_i$. Since in multi-agent search, each robot moves from one configuration to the next in one step, each roadmap edge is divided into several parts to ensure that the distance from one vertex to the next is no more than a user-specified step size.

The overall approach is agnostic to the inner workings of the multi-agent search, so it can be used in conjunction with any available method that operates over graphs. This paper includes experiments using three different methods, namely WHCA*(Silver, 2005), Push-and-Swap (Luna & Bekris, 2011), and SIPP (Narayanan et al., 2012).

4.2 Motion Tree in the Composite Continuous State Space

The relaxed problem setting presented in the previous section takes into account the robot shapes but not their dynamics. As a result, there is no guarantee that solutions corresponding to the relaxed problem setting are dynamically feasible. In fact, geometric and differential constraints imposed by the obstacles and the underlying dynamics may make it difficult or impossible for a robot to follow its route σ_i.

To account for the dynamics, a motion tree $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is incrementally expanded in the composite state space $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$. This is to ensure probabilistic completeness. Prioritized or decoupled approaches, which would expand one tree for each robot, often fail to find solutions in cases where robots have to coordinate their motions. As such, it is critical to expand the motion tree in the composite state space of all the robots. As described later, scalability and efficiency is obtained by using multi-agent search over the discrete abstraction to guide the motion-tree expansion.

Each vertex $v \in V_\mathcal{T}$ corresponds to a composite state, denoted by $\textsc{state}(v)$, where $\textsc{state}_i(v) \in \mathcal{S}_i$ represents the state associated with the robot $\mathcal{R}_i$. By construction, the vertex v is added to $V_\mathcal{T}$ only if $\textsc{collision}(\textsc{state}(v))$ = `false`, i.e., when the robots are placed according to $\textsc{state}(v)$, there is no robot-robot or robot-obstacle collision.

As any tree-based approach, a procedure for generating successor states is required to expand the motion tree. Specifically, a successor v_{new} is generated from v by applying some control actions $\langle a_1, \ldots, a_n \rangle$ component-wise and integrating the motion equations of each robot for one time step dt, i.e.,

$$\forall i \in \{1, \ldots, n\} : \text{STATE}_i(v_{\text{new}}) \leftarrow \text{SIMULATE}(\text{STATE}_i(v), a_i, f_i, dt). \tag{14}$$

If $\text{STATE}(v_{\text{new}})$ is not in collision, then the edge (v, v_{new}) is added to $E_{\mathcal{T}}$ and is labeled with the actions $\langle a_1, \ldots, a_n \rangle$ that were used to generate v_{new} from v.

The generation of a successor state by applying controls and integrating the motion equations is what makes it possible for a motion tree to account for the underlying robot dynamics. This is in contrast to roadmap approaches which are unable to account for the robot dynamics since each roadmap edge (v_a, v_b) requires exact steering to generate a dynamically-feasible trajectory $\zeta : [0, T] \rightarrow \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ that connects its states. This gives rise to a high-dimensional differential two-boundary value problem (boundary conditions: $\zeta(0) = \text{STATE}(v_a)$ and $\zeta(T) = \text{STATE}(v_b)$). Two-boundary value problems are notoriously challenging, especially for motion equations expressing constraints imposed by the underlying robot dynamics. Analytical solutions are generally not available, while numerical solutions leave gaps and are computationally demanding (Keller, 1992; Cheng et al., 2008).

To solve the multi-robot motion-planning problem, the motion tree $\mathcal{T}$ is rooted at the initial state $\langle s_1^{\text{init}}, \ldots, s_n^{\text{init}} \rangle$. As mentioned, $\mathcal{T}$ is incrementally expanded by selecting a vertex v from $\mathcal{T}$ and generating a successor v_{new} from v. The expansion continues until a vertex v_{new} is added where each robot has reached its goal, i.e., $\forall i \in \{1, \ldots, n\}$: $\text{POSITION}(\text{STATE}_i(v_{\text{new}})) \in \mathcal{G}_i$. The path $\langle v_1, \ldots, v_\ell \rangle$, with $v_1 = v_{\text{init}}$ and $v_\ell = v_{\text{new}}$, from the root of $\mathcal{T}$ to v_{new} is retrieved and the solution trajectory for each robot $\mathcal{R}_i$ is constructed as $\zeta_i = \langle \text{STATE}_i(v_1), \ldots, \text{STATE}_i(v_\ell) \rangle$.

4.3 Abstraction-Based Partitioning of the Motion Tree into Equivalence Classes

The efficiency of the overall approach depends on its ability to effectively expand the motion tree $\mathcal{T}$ so that it quickly finds a solution to the multi-robot motion-planning problem. The motion tree $\mathcal{T}$ could be expanded in an uninformed way by selecting at each iteration a vertex $v \in V_{\mathcal{T}}$ at random and applying controls to generate a successor v_{new} from v. Uninformed expansions, however, would have difficulty finding solutions since the composite state space is continuous and high-dimensional. Moreover, reaching the goals would require very deep expansions of the motion tree. In addition, the branching factor is infinite since the control actions are represented by continuous variables.

To address these challenges, we introduce multi-agent search to guide the motion-tree expansion. Since multi-agent search operates over the discrete abstraction, a mapping from the composite state space to the discrete abstraction must be first defined. Since the discrete abstraction is obtained via roadmaps, the mapping function maps states to roadmap configurations. Specifically, the mapping function $\text{MAP}_i : \mathcal{S}_i \rightarrow V_{\mathcal{M}_i}$ maps $s_i \in \mathcal{S}_i$ to the nearest configuration in the roadmap, i.e.,

$$\text{MAP}_i(s_i) = \text{NearestNeighbor}(V_{\mathcal{M}_i}, \langle \text{POSITION}(s_i), \text{ORIENTATION}(s_i) \rangle). \tag{15}$$

In case of ties for the nearest neighbor, s_i is mapped to the nearest roadmap configuration with the lowest index. The composite mapping function $\text{MAP} : \mathcal{S}_1 \times \ldots \times \mathcal{S}_n \rightarrow V_{\mathcal{M}_1} \times \ldots \times$

$V_{\mathcal{M}_n}$ is then defined as

$$\textsc{map}(\langle s_1, \ldots, s_n \rangle) = \langle \textsc{map}(s_1), \ldots, \textsc{map}(s_n) \rangle. \tag{16}$$

This mapping makes it possible to partition the motion tree $\mathcal{T}$ into equivalence classes. The idea is that vertices in $\mathcal{T}$ that provide the same discrete information should belong to the same equivalence class. In other words, if $\textsc{map}(\textsc{state}(v_a)) = \textsc{map}(\textsc{state}(v_b))$, then vertices $v_a, v_b \in V_{\mathcal{T}}$ should belong to the same equivalence class.

In this way, the equivalence class $\Gamma_{\langle c_1,\ldots,c_n \rangle}$ with $\langle c_1, \ldots, c_n \rangle \in V_{\mathcal{M}_1} \times \ldots \times V_{\mathcal{M}_n}$ groups together all the vertices in $\mathcal{T}$ that map to $\langle c_1, \ldots, c_n \rangle$, i.e.,

$$\Gamma_{\langle c_1,\ldots,c_n \rangle} = \{v : v \in V_{\mathcal{T}} \wedge \textsc{map}(\textsc{state}(v)) = \langle c_1, \ldots, c_n \rangle\}. \tag{17}$$

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

$$\Gamma = \{\Gamma_{\langle c_1,\ldots,c_n \rangle} : \langle c_1, \ldots, c_n \rangle \in V_{\mathcal{M}_1} \times \ldots \times V_{\mathcal{M}_n} \wedge |\Gamma_{\langle c_1,\ldots,c_n \rangle}| > 0\}. \tag{18}$$

As an implementation note, Γ is maintained as a hashmap. The key for an equivalence class $\Gamma_{\langle c_1,\ldots,c_n \rangle}$ is computed as $\textsc{key}(\langle c_1, \ldots, c_n \rangle) = \langle \textsc{index}(c_1), \ldots, \textsc{index}(c_n) \rangle$, where $\textsc{index}(c_i)$ denotes the id number for the configuration c_i in the roadmap $\mathcal{M}_i$, e.g., position in the vector that stores the roadmap vertices. When a new vertex v_{new} is added to the motion tree $\mathcal{T}$, its key is computed as $\textsc{key}(\textsc{map}(\textsc{state}(v_{\text{new}})))$. If the hashmap does not contain the key, then the equivalence class $\Gamma_{\textsc{map}(\textsc{state}(v_{\text{new}}))}$ is created and added to Γ; otherwise, it is retrieved from Γ. In both cases, v_{new} is added to $\Gamma_{\textsc{map}(\textsc{state}(v_{\text{new}}))}$.

What is the significance of partitioning $\mathcal{T}$ into equivalence classes? The partition makes it possible to leverage multi-agent search. In particular, when creating an equivalence class $\Gamma_{\langle c_1,\ldots,c_n \rangle}$, $\textsc{MultiAgentSearch}(\mathcal{M}_1, \ldots, \mathcal{M}_n, c_1, \ldots, c_n, \mathcal{G}_1, \ldots, \mathcal{G}_n)$ can be invoked to compute non-conflicting routes $\sigma_1, \ldots, \sigma_n$, where each σ_i starts at c_i and reaches $\mathcal{G}_i$. When selecting a vertex from $\Gamma_{\langle c_1,\ldots,c_n \rangle}$, the objective becomes to expand the motion tree $\mathcal{T}$ along the routes $\sigma_1, \ldots, \sigma_n$. Hence, the partition makes it possible to use multi-agent search as a heuristic to guide the motion-tree expansion.

4.4 Putting it All Together: Using Multi-agent Search over the Discrete Abstraction to Guide the Motion-Tree Expansion

The overall approach starts by constructing the roadmaps $\mathcal{M}_1, \ldots, \mathcal{M}_n$ to obtain the discrete abstraction. The motion tree $\mathcal{T}$ is rooted at the initial state $\langle s_1^{\text{init}}, \ldots, s_n^{\text{init}} \rangle$. The first equivalence class $\Gamma_{\textsc{map}(\textsc{state}(v_{\text{init}}))}$ is also created, which contains the root vertex v_{init}. Afterwards, the approach enters a core loop, where each iteration consists of the following:

- select an equivalence class $\Gamma_{\langle c_1,\ldots,c_n \rangle}$ from Γ;
- use multi-agent search over the discrete abstraction to obtain non-conflicting routes $\sigma_1, \ldots, \sigma_n$, where each σ_i starts at c_i, and reaches the goal $\mathcal{G}_i$; and
- expand the motion tree $\mathcal{T}$ from vertices in $\Gamma_{\langle c_1,\ldots,c_n \rangle}$ along the routes $\sigma_1, \ldots, \sigma_n$.

When an equivalence class $\Gamma_{\langle c_1,\ldots,c_n \rangle}$ is first created, $\textsc{MultiAgentSearch}(\mathcal{M}_1, \ldots, \mathcal{M}_n, c_1, \ldots, c_n, \mathcal{G}_1, \ldots, \mathcal{G}_n)$ is invoked to compute non-conflicting routes $\langle \sigma_1, \ldots, \sigma_n \rangle$, where each

σ_i starts at c_i and reaches the goal $\mathcal{G}_i$. To save computation 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 $\Gamma_{\langle c_1,\ldots,c_n\rangle}$.

When selecting an equivalence class $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ from Γ, priority is given to equivalence classes associated with short routes, since expansions along short routes are more likely to quickly lead each robot to its goal. As the motion tree $\mathcal{T}$ is expanded, new vertices are added, some of which could result in new equivalence classes being created. When the geometric and differential constraints imposed by the obstacles and the underlying robot dynamics make it difficult to expand $\mathcal{T}$ along the routes $\sigma_1, \ldots, \sigma_n$ associated with $\Gamma_{\langle c_1,\ldots,c_n\rangle}$, then $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ is penalized in order to promote expansions from other equivalence classes.

This process of selecting and expanding an equivalence class is repeated until a solution is found or a runtime limit is reached. A schematic illustration is shown in Figure 2. Pseudocode is shown in Alg. 2.

4.4.1 Selecting an Equivalence Class Based on Route Costs and Penalties

A weight is defined for each equivalence class $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ as an estimate of the importance of expanding the motion tree $\mathcal{T}$ from $\Gamma_{\langle c_1,\ldots,c_n\rangle}$. The equivalence class with the maximum weight is then selected for expansion. Specifically, the weight for $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ is defined as

$$w(\Gamma_{\langle c_1,\ldots,c_n\rangle}) = \frac{\alpha^{\textsc{NrSel}(\Gamma_{\langle c_1,\ldots,c_n\rangle})}}{\sum_{i=1}^{n}(\textsc{cost}(\sigma_i))^2}, \tag{19}$$

where $\textsc{NrSel}(\Gamma_{\langle c_1,\ldots,c_n\rangle})$ denotes the number of times $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ has been previously selected, and $\sigma_1, \ldots, \sigma_n$ denote the routes associated with $\Gamma_{\langle c_1,\ldots,c_n\rangle}$, $0 < \alpha < 1$.

In this way, equivalence classes associated with low-cost routes have a high weight in order to promote expansions from areas close to the goals. This constitutes the greedy aspect of the weight. As a purely greedy selection could lead to pitfalls, a penalty factor α ($0 < \alpha < 1$) is introduced to reduce the weight of $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ each time it is selected. This ensures that $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ cannot be selected 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. The penalty factor 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 imposed by the obstacles and the underlying robot dynamics.

4.4.2 Expanding an Equivalence Class along Roadmap Routes

After selecting $\Gamma_{\langle c_1,\ldots,c_n\rangle}$, the objective is to expand the motion tree $\mathcal{T}$ along the routes $\sigma_1, \ldots, \sigma_n$ associated with $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ (Alg. 2(b)). The idea here is to have each robot $\mathcal{R}_i$ move toward the configurations in σ_i one after the other. Since the routes are not necessarily dynamically feasible, the robots are given some flexibility when following the routes. Forcing a robot to follow its route exactly could be infeasible, which would cause the motion-tree expansion to make little or no progress. The motion-tree expansion stops as soon as a collision is found or a maximum number of expansion steps is reached. If during the expansion, each robot reaches its goal, then a solution is found.

More specifically, the expansion procedure starts by setting a target configuration c_i^{target} for each robot $\mathcal{R}_i$. The target configuration c_i^{target} is computed by sampling a collision-

Algorithm 2 Pseudocode for the proposed approach

Input: robot models $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$, $\mathcal{R}_i = \langle \mathcal{P}_i, \mathcal{S}_i, \mathcal{A}_i, f_i \rangle$; bounding box $\mathcal{W}$; obstacles $\mathcal{O}$; goal regions $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$; initial state $\langle s_1^{\text{init}}, \ldots, s_n^{\text{init}} \rangle$; 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 initial to goal; or `null` if no solution

```
1: $\mathcal{W}_{\text{free}} \leftarrow$ triangulate obstacle-free area $\mathcal{W} \setminus \mathcal{O}$
2: for $i = 1 \ldots n$ do
3:     $\mathcal{M}_i = (V_{\mathcal{M}_i}, E_{\mathcal{M}_i}, \text{COST}_{\mathcal{M}_i}) \leftarrow \text{ROADMAP}(\mathcal{W}, \mathcal{O}, \mathcal{W}_{\text{free}}, \mathcal{C}_i, \mathcal{P}_i, s_i^{\text{init}}, \mathcal{G}_i)$
4: $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T}) \leftarrow (\emptyset, \emptyset)$; $\Gamma \leftarrow \emptyset$
5: AddVertex($\mathcal{T}, \Gamma, \langle s_1^{\text{init}}, \ldots, s_n^{\text{init}} \rangle$, parent $\leftarrow$ null, $\langle a_1, \ldots, a_n \rangle \leftarrow$ null)
6: while TIME() $< t_{\max}$ do
7:     $\Gamma_{\langle c_1, \ldots, c_n \rangle} \leftarrow$ SelectEquivalenceClass($\Gamma$) //routes were computed by multi-agent search when the
8:     $\langle \sigma_1, \ldots, \sigma_n \rangle \leftarrow$ RetrieveRoutes($\Gamma_{\langle c_1, \ldots, c_n \rangle}$) //equivalence class was first created
9:     $v_{\text{goal}} \leftarrow$ ExpandMotionTree($\mathcal{T}, \Gamma, \Gamma_{\langle c_1, \ldots, c_n \rangle}, \sigma_1, \ldots, \sigma_n$)
10:    if $v_{\text{goal}} \neq$ null then return $\langle \zeta_1, \ldots, \zeta_n \rangle \leftarrow$ RetrieveTrajectories($\mathcal{T}, v_{\text{new}}$)
11: return null
```

```
(a) ExpandMotionTree($\mathcal{T}, \Gamma, \Gamma_{\langle c_1, \ldots, c_n \rangle}, \sigma_1, \ldots, \sigma_n$)
1: for $i = 1 \ldots n$ do
2:     $c_i^{\text{target}} \leftarrow$ FirstTarget($\sigma_i$)
3: $v \leftarrow$ SelectVertex($\Gamma_{\langle c_1, \ldots, c_n \rangle}, c_1^{\text{target}}, \ldots, c_n^{\text{target}}$)
4: for several steps do
5:     solved $\leftarrow$ true
6:     for $i = 1 \ldots n$ do
7:         $s_i \leftarrow \text{STATE}_i(v)$
8:         $a_i \leftarrow \text{CONTROLLER}(s_i, c_i^{\text{target}})$
9:         $s_i^{\text{new}} \leftarrow \text{SIMULATE}(s_i, a_i, f_i, dt)$
10:        if POSITION($s_i^{\text{new}}$) $\notin \mathcal{G}_i$ then solved $\leftarrow$ false
11:        if NEAR($s_i^{\text{new}}, c_i^{\text{target}}$) = true then $c_i^{\text{target}} \leftarrow$ NextTarget($\sigma_i$)
12:    if COLLISION($s_1^{\text{new}}, \ldots, s_n^{\text{new}}$) = true then break
13:    $v_{\text{new}} \leftarrow$ AddVertex($\mathcal{T}, \Gamma, \langle s_1^{\text{new}}, \ldots, s_n^{\text{new}} \rangle, v, \langle a_1, \ldots, a_n \rangle$)
14:    if solved then return $v_{\text{new}}$
15:    $v \leftarrow v_{\text{new}}$
16: return null
```

```
(b) AddVertex($\mathcal{T}, \Gamma, \langle s_1^{\text{new}}, \ldots, s_n^{\text{new}} \rangle, v, \langle a_1, \ldots, a_n \rangle$)
1: $v_{\text{new}} \leftarrow$ new motion-tree vertex with $\langle s_1^{\text{new}}, \ldots, s_n^{\text{new}} \rangle$ as its state
2: $(v, v_{\text{new}}) \leftarrow$ new edge labeled with the control actions $\langle a_1, \ldots, a_n \rangle$
3: $V_\mathcal{T} \leftarrow V_\mathcal{T} \cup \{v_{\text{new}}\}$; $E_\mathcal{T} \leftarrow E_\mathcal{T} \cup \{(v, v_{\text{new}})\}$
4: $\langle c_1^{\text{new}}, \ldots, c_n^{\text{new}} \rangle \leftarrow \text{MAP}(s_1^{\text{new}}, \ldots, s_n^{\text{new}})$
5: $\Gamma_{\langle c_1^{\text{new}}, \ldots, c_n^{\text{new}} \rangle} \leftarrow \text{FIND}(\Gamma, \langle c_1^{\text{new}}, \ldots, c_n^{\text{new}} \rangle)$
6: if $\Gamma_{\langle c_1^{\text{new}}, \ldots, c_n^{\text{new}} \rangle}$ = null then
7:     $\Gamma_{\langle c_1^{\text{new}}, \ldots, c_n^{\text{new}} \rangle} \leftarrow$ new equivalence class
8:     $\langle \sigma_1, \ldots, \sigma_n \rangle \leftarrow$ MultiAgentSearch($\mathcal{M}_1, \ldots, \mathcal{M}_n, c_1^{\text{new}}, \ldots, c_n^{\text{new}}, \mathcal{G}_1, \ldots, \mathcal{G}_n$)
9:     store $\langle \sigma_1, \ldots, \sigma_n \rangle$ with $\Gamma_{\langle c_1^{\text{new}}, \ldots, c_n^{\text{new}} \rangle}$
10: INSERT($\Gamma_{\langle c_1^{\text{new}}, \ldots, c_n^{\text{new}} \rangle}, v_{\text{new}}$)
11: INSERT($\Gamma, \Gamma_{\langle c_1^{\text{new}}, \ldots, c_n^{\text{new}} \rangle}$)
12: return $v_{\text{new}}$
```

free configuration near $\sigma_i[2]$ (since $\sigma_i[1] = c_i$). Sampling near $\sigma_i[2]$ as opposed to setting $c_i^{\text{target}} = \sigma_i[2]$ is preferred to give more flexibility to the robot, since σ_i is not necessarily dynamically feasible. As such, it could be infeasible for the robot $\mathcal{R}_i$ to exactly follow σ_i.

After setting the targets, the procedure seeks 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.,

$$v = \operatorname*{argmin}_{v' \in \Gamma_{\langle c_1,\ldots,c_n\rangle}} \sum_{i=1}^{n} \|\textsc{position}(c_i) - \textsc{position}(\textsc{state}_i(v'))\|_2. \tag{20}$$

A collision-free and dynamically-feasible trajectory is generated in the composite state space $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ by starting at v and moving toward $\langle c_1^{\text{target}},\ldots,c_n^{\text{target}}\rangle$. Specifically, a successor v_{new} is obtained from v by applying some control actions and integrating the motion equations for one time step dt. If v_{new} is in collision, then the expansion terminates. If v_{new} is not in collision, then v_{new} and the edge (v, v_{new}) are added to the motion tree $\mathcal{T}$. This process is repeated for several steps by setting v to v_{new} at the end of each step, so that the expansion continues from the new vertex added to $\mathcal{T}$.

A proportional-integrative-derivative (PID) controller (Spong, Hutchinson, & Vidyasagar, 2005) is used to select the control actions that steer robot $\mathcal{R}_i$ toward c_i^{target}. For a vehicle, the PID controller selects controls that turn the wheels and then move toward c_i^{target}.

When $\textsc{state}_i(v_{\text{new}})$ gets near c_i^{target} (within some predefined distance), the next target c_i^{target} is computed by sampling a collision-free configuration near the next configuration in σ_i. In this way, the robot $\mathcal{R}_i$ can progress from one configuration in σ_i to the next.

When adding v_{new} to the motion tree $\mathcal{T}$, the partition Γ is updated accordingly, as described earlier. If $\textsc{state}(v_{\text{new}})$ has reached all the goals, then the algorithm terminates successfully since a solution is found.

4.5 Runtime Analysis and Probabilistic Completeness

When constructing a roadmap $\mathcal{M}_i$, most of the time is spent in collision checking and nearest-neighbors computations. Collision checking based on sweep-and-prune algorithms 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 and $|\mathcal{P}_i|$ denotes the number of vertices for the geometric shape of robot $\mathcal{R}_i$ (Larsen et al., 1999; Tracy, Buss, & Woods, 2009; Pan et al., 2012). Nearest neighbors can be computed in $O(k \log |V_{\mathcal{M}_i}|)$ time, where k is the number of neighbors (Beygelzimer, Kakade, & Langford, 2006; Muja & Lowe, 2014).

The number of collision-checking and nearest-neighbors calls depends on the characteristics of the environment, the placement of the initial and goal configurations, the probability distribution used for generating collision-free configurations, the length of the roadmap edges, and many other factors. Providing a bound on the number of calls remains an open problem. In fact, analysis of roadmap approaches have focused on showing probabilistic completeness as a function of the number of samples (Kavraki et al., 1996; Ladd & Kavraki, 2004; Karaman & Frazzoli, 2011), which are also applicable to our roadmap constructions.

After the roadmap construction, our approach enters its core loop (Alg. 2:6). The runtime here is dominated by calls to selecting an equivalence class from Γ, inserting a new equivalence class into Γ, selecting a vertex from a given equivalence class, simulating motion dynamics, collision checking, mapping, and multi-agent search. Maintaining a heap data structure for the equivalence classes, makes it possible to retrieve the equivalence class with maximum weight in $O(1)$ time and to insert a new class into Γ in $O(\log |\Gamma|)$

time. Using efficient nearest-neighbors data structures, the procedure for selecting a vertex from an equivalence class $\Gamma_{\langle c_1,\ldots,c_n\rangle}$ runs in $O(\log|\Gamma_{\langle c_1,\ldots,c_n\rangle}|)$ time (or $O(\log|V_{\mathcal{T}}|)$ since $|\Gamma_{\langle c_1,\ldots,c_n\rangle}| \leq |V_{\mathcal{T}}|$). MAP computes one nearest neighbor from each roadmap, so it runs in $O(\sum_{i=1}^{n}\log|V_{\mathcal{M}_i}|)$ time. The complexity of MultiAgentSearch depends on the particular method being used, so it is denoted here as $t_{\textsc{MultiAgentSearch}}$.

SIMULATE uses Runge-Kutta to integrate the motion equations, so it runs in $O(d_1 + \ldots + d_n)$ time for all the robots, where d_i is the dimensionality of $\mathcal{S}_i$. COLLISION checks for robot-obstacle and robot-robot collisions. Robot-obstacle collision checking runs in $O((n_{\mathcal{O}} + n_{\mathcal{P}})\log(n_{\mathcal{O}} + n_{\mathcal{P}}))$ time, where $n_{\mathcal{P}} = \sum_{i=1}^{n}|\mathcal{P}_i|$. Robot-robot collision checking runs in $O(nn_{\mathcal{P}}\log n_{\mathcal{P}})$ time, since checking robot $\mathcal{R}_i$ against all the other robots runs in $O(n_{\mathcal{P}}\log n_{\mathcal{P}})$ time. Hence, COLLISION runs in $O(n_{\mathcal{O}}\log n_{\mathcal{O}} + n\, n_{\mathcal{P}}\log n_{\mathcal{P}})$ time.

Let n_{MP} denote the number of times the main while loop is entered (Alg. 2:6). Then, selecting an equivalence class and selecting a vertex are each invoked n_{MP} times. COLLISION and MAP are invoked $O(n_{\mathrm{MP}})$ times since a user-defined constant bounds the number of times the for loop in Alg. 2(a):4 is entered. For the same reason, it follows that SIMULATE is invoked $O(n\, n_{\mathrm{MP}})$ times. MultiAgentSearch is invoked once per equivalence class, so $|\Gamma|$ times. Putting it all together, the runtime complexity is

$$O(|\Gamma| t_{\textsc{MultiAgentSearch}} + n_{\mathrm{MP}}(\log|V_{\mathcal{T}}| + \sum_{i=1}^{n} d_i + n_{\mathcal{O}}\log n_{\mathcal{O}} + n\, n_{\mathcal{P}}\log n_{\mathcal{P}} + \sum_{i=1}^{n}\log|V_{\mathcal{M}_i}|)). \tag{21}$$

Note that $|\Gamma| \leq |V_{\mathcal{T}}| \leq n_{\mathrm{MP}}$.

Sampling-based motion planners generally offer probabilistic completeness, which guarantees that when a solution exists, it will be found with probability approaching one as time goes to infinity. Our approach also offers probabilistic completeness. The claim follows from the analysis by Plaku (2015) which can be applied to sampling-based motion planners that partition the motion tree into equivalence classes. The analysis requires guaranteeing that no equivalence will be selected indefinitely. This was one of the reasons for introducing the penalty factor in the definition of the weight for an equivalence class (Eqn. 19). Although the analysis by Plaku is presented for a single robot, it still applies in our case since $\mathcal{T}$ is expanded over the composite state space $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$. For the purposes of the analysis, multiple robots are considered as one system operating in $\mathcal{S}$.

5. Experiments and Results

Experimental validation is provided in simulation using vehicle models with nonlinear dynamics operating in complex environments, as shown in Figure 1 and 4, where cooperation among the robots is often required to make it possible for each robot to reach its goal. Experiments are also conducted with vehicles modeled by physics game engines which provide an increased level of realism by also modeling friction, terrains, general rigid-body dynamics, and other interactions of the robots with the world, which cannot be easily described analytically by motion equations.

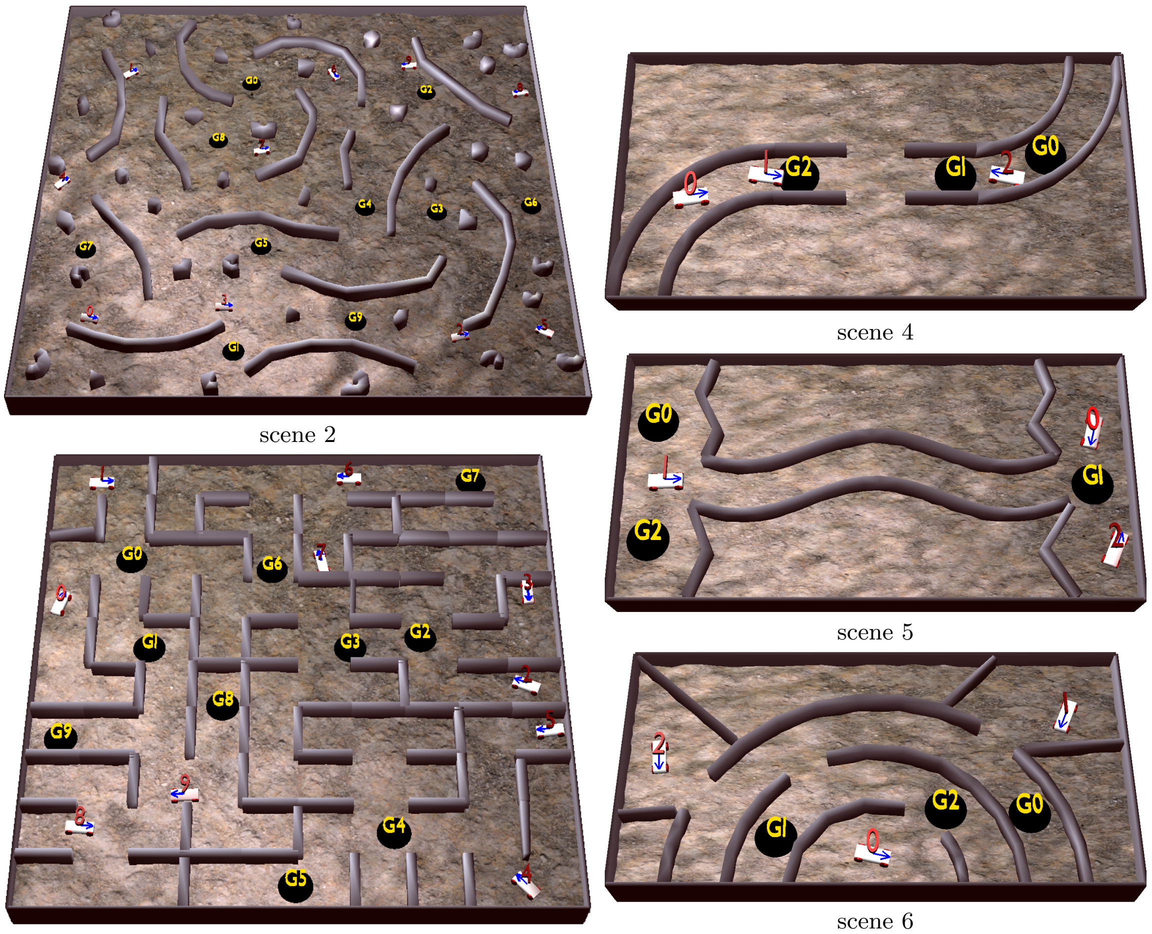


Figure 4: Scenes used in the experiments (scene 1 is shown in Figure 1). Scenes are shown with the bumpy terrain, which is used for the experiments with the physics-based vehicle models, using Bullet as the underlying physics game engine. For the experiments with the vehicle models based on differential equations of motion (Section 3.1), the bumpy terrain is replaced with a flat terrain (shown in Figure 3). Videos of solutions obtained by our approach can be found at http://goo.gl/8muxwC. Figure best viewed in color and on screen.

The efficiency and scalability of the approach is demonstrated in comparison to related work across different environments and as the number of robots is increased. Experiments also evaluate the approach when used with different multi-agent search methods, namely WHCA* (Silver, 2005), Push-and-Swap (Luna & Bekris, 2011), and SIPP (Narayanan et al., 2012).

5.1 Vehicle Models and Problem Instances

Experiments are conducted using vehicles whose motions are modeled by differential equations and by physics game engines. The vehicle model based on differential equations of motion is described in Section 3.1. The physics-based vehicle model uses the game engine Bullet (Coumans, 2012) as the underlying simulator. Bullet simulates the interactions of the vehicle with the environment by taking into account the friction from the wheels, the slip caused by friction, suspension damping, suspension stiffness, and compression. Due to the complexity of the interactions between the vehicle and the terrain, Bullet, as other physics game engines, relies on numerical simulations of general rigid-body dynamics since analytical formulas are generally not available.

Experiments are conducted using 6 scenes, as shown in Figure 1 and 4. Scenes 1, 2, and 3 feature unstructured and obstacle-rich environments. Scenes 1 and 3 require some level of cooperation among the robots as the motions of one robot toward its goal region could interfere with the motions of another robot. Scene 2 requires little or no cooperation among the robots. Scenes 4, 5, 6 are specifically designed to test the ability of multi-robot motion planners to solve problems where cooperation among robots is essential.

A problem instance is defined by the scene, number of robots, initial placement of the robots, and goal regions. For each scene and number of robots n a set of 60 problem instances is generated, denoted by $\mathcal{I}_{\langle \text{scene}, n\rangle}$, by randomly placing the robots and the goals in the environment. To make the test cases more challenging, rather than randomly sampling from the entire $\mathcal{W}$, the robots and the goals are placed at random locations inside certain manually-selected areas. More specifically, for each scene and number of robots n, we define areas $\mathcal{E}_1^{\text{init}}, \ldots, \mathcal{E}_n^{\text{init}}$ and $\mathcal{E}_1^{\text{goal}}, \ldots, \mathcal{E}_n^{\text{goal}}$. The robot $\mathcal{R}_i$ and the goal $\mathcal{G}_i$ are then placed at random inside $\mathcal{E}_i^{\text{init}}$ and $\mathcal{E}_i^{\text{goal}}$, respectively.

To evaluate the performance, each multi-robot motion planner is run on each of the problem instances. For a scene and number of robots n, results report the mean runtime obtained over the 60 problem instances in $\mathcal{I}_{\langle \text{scene}, n\rangle}$, after dropping the best and worst five runs to avoid the influence of outliers. The runtime measures everything from reading the input file to reporting that a solution is found (including the roadmap constructions for our approach). Experiments were run on an Intel Core i7 (1.90GHz).

5.2 Comparisons to Prioritized and Centralized Approaches

To evaluate the competitiveness of the approach, comparisons are carried out using a centralized and a prioritized version of `RRT` (LaValle & Kuffner, 2001), denoted by `cRRT` and `pRRT`. `RRT` is a widely-used sampling-based motion planner. The implementations of `cRRT` and `pRRT` have been fine-tuned and use goal bias, multi-step expansions, and efficient nearest-neighbor data structures, as advocated in the literature.

Comparisons are also done with a prioritized version of `GUST` (Plaku, 2015), denoted by `pGUST`. `GUST` was selected since it has has been shown to be significantly faster than `RRT` and other sampling-based motion planners when solving challenging problems with dynamics. We could only use a prioritized version of `GUST`, since, as mentioned in the related work section, a centralized version is not straightforward and has yet to be developed.

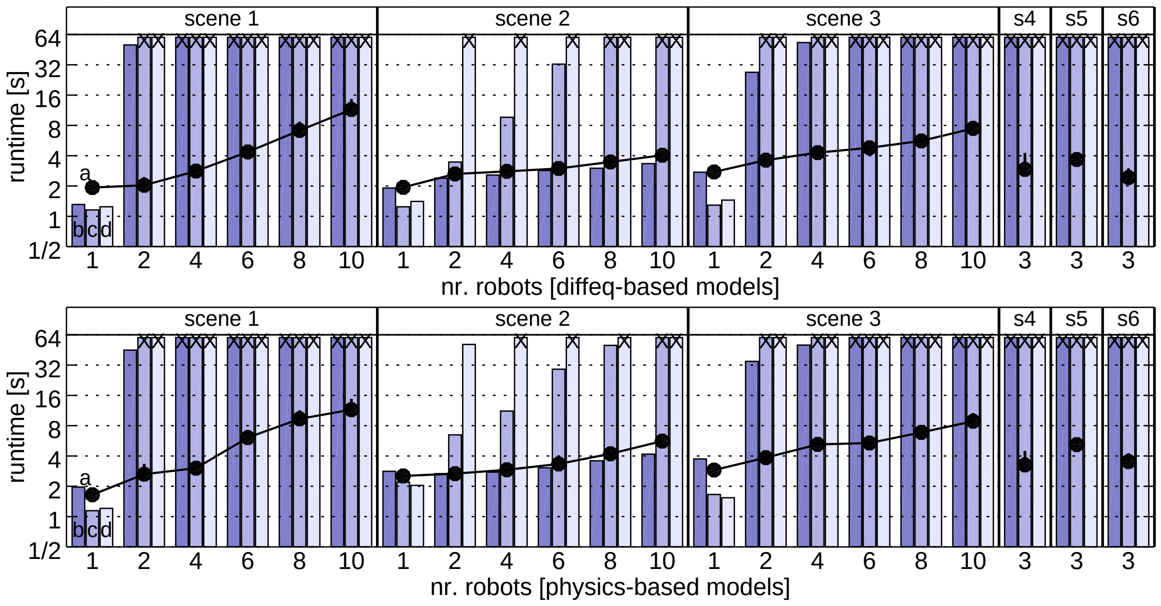


Figure 5: Results when comparing (a) our approach to (b) `pGUST`, (c) `pRRT`, and (d) `cRRT`. Runtime measures everything from reading the input file to reporting that a solution is found (including for our approach the time to build the roadmaps). Missing entries indicate failure by the planner to solve 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).

Figure 5 shows the results of the comparisons. Our approach is significantly faster than `cRRT`, `pRRT`, and `pGUST`. The speedups become higher as the number of robots is increased and for the scenes where cooperation among the robots is essential to reach the goals.

The other approaches have difficulty solving these challenging problems. The runtime of `cRRT` degrades rapidly as more and more robots are added to the scene. This is due to the high-dimensionality of the composite state space $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ and the lack of global guidance, which makes it difficult for `cRRT` to expand the motion tree toward the goals $\mathcal{G}_1, \ldots, \mathcal{G}_n$. `pRRT` is faster than `cRRT` since `pRRT` plans the motions of the robots one at a time, so it avoids searching over the composite state space. As any prioritized approach, however, `pRRT` has difficulty finding solutions in scenes where cooperation among the robots is required. `pRRT` also has difficulty findings solutions when the number of robots is increased. Even in scene 2, which requires little or no cooperation, `pRRT` times out as the number of robots is increased.

`pGUST` is faster than `pRRT` since `pGUST` uses discrete search to guide the motion-tree expansion toward goal $\mathcal{G}_i$ when planning the motions of robot $\mathcal{R}_i$. `pGUST` does best in Scene 2 where cooperation among the robots is not essential. Note that even for this scene, our approach, which expands the motion tree in the composite state space, is as fast as `pGUST`. For the other scenes (Scenes 1, 3, 4, 5, 6), where cooperation is important, `pGUST`

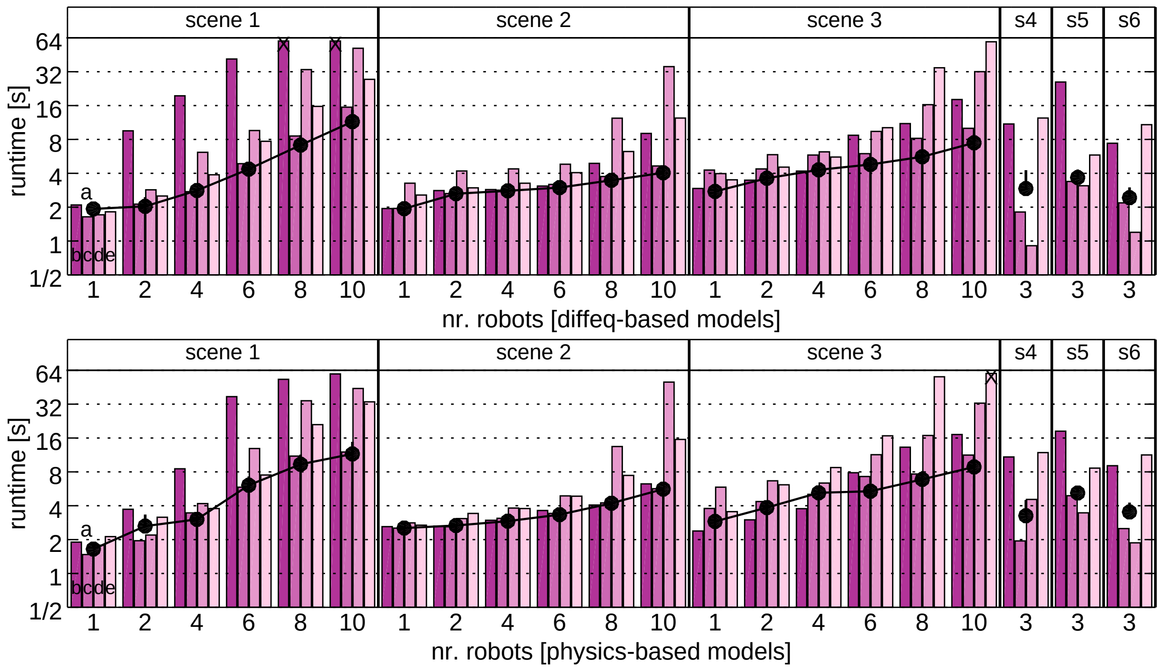


Figure 6: Results of our overall approach CoSMMAS when varying the multi-agent search: (a) CoSMMAS[WHCA*, window size: 5], (b) CoSMMAS[WHCA*, window size: 2], (c) CoSMMAS[WHCA*, window size: 10], (d) CoSMMAS[SIPP], (e) CoSMMAS[Push-and-Swap].

has difficulty finding solutions as trajectories planned for previous robots often prevent the current robot from reaching its goal.

Our approach shows remarkable efficiency in solving these challenging multi-robot motion-planning problems, even as the number of robots is increased. Our approach is particularly well-suited for scenes where cooperation among robots is essential. While cRRT, pRRT, and pGUST all timed out in scenes 4, 5, 6 (set to 90s per run), our approach was able to find solutions in 1-3s. This is as a result of using multi-agent search over the discrete abstraction to effectively guide the motion-tree expansion.

5.3 Impact of Multi-agent Search

As mentioned, CoSMMAS is agnostic to the inner workings of the multi-agent search, so it can be used with any available method. To evaluate the impact of the multi-agent search on the overall performance of CoSMMAS, experiments were run using WHCA* with three different window sizes (2, 5, and 10) (Silver, 2005), SIPP (Narayanan et al., 2012), and Push-and-Swap (Luna & Bekris, 2011).

Results in Figure 6 show that CoSMMAS works well with WHCA*, SIPP, and Push-and-Swap. The best performance is obtained when using WHCA* (with window size set to 5). For WHCA*, the window size can have an impact on the overall performance. The window size can be thought of as the number of look-ahead moves to resolve conflicts. A small

				CoSMMAS with multi-agent search as				
	cRRT	pRRT	pGUST	CSIPP	PaS	WHCA*2	WHCA*5	WHCA*10
Runtime (s)	93494	81216	56928	16654	16918	21963	7790	8231
Percentage	30.74%	26.53%	18.59%	5.78%	6.16%	6.68%	2.72%	2.81%

Table 1: Runtime for each of the methods used in the experiments over all the problem instances (1260 in total) and over both diffeq- and physics-based vehicle models.

window size works well in scenes requiring little or no cooperation among the robots but can have a hard time resolving conflicts in scenes where cooperation among the robots is essential. A large window size causes WHCA* to generate larger portions of non-conflicting routes. This increases the runtime to compute non-conflicting routes, but could potentially reduce the number of reroutings as longer non-conflicting routes are generated. Results in Figure 6 show that the overall 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 overall runtime.

5.4 Runtime Distribution

Table 1 shows the runtime of each of the methods used in the experiments over all the problem instances and over both diffeq- and physics-based vehicle models. Essentially, it is the runtime over everything that was used for the experiments in this paper. The results show that CoSMMAS is significantly faster than the other methods.

Figure 7 shows the runtime distribution for various components of CoSMMAS. Although the construction of roadmaps takes some time (about 1-2s), it more than makes up for it by providing a network of roads, seeking to make it easy to reach the goals from any location in the environment. Multi-agent search also takes a considerable percentage of the runtime, especially as the number of robots increases. This also pays off as it provides non-conflicting routes that are used to effectively guide the motion-tree expansion. Overall, the roadmaps and multi-agent search make it possible for CoSMMAS to shift the load from the motion-tree expansion in the composite continuous state space to multi-agent search over graphs. In this way, non-conflicting routes obtained by multi-agent search 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.

6. Discussion

This paper developed an effective, cooperative, and probabilistically-complete multi-robot motion planner which took into account the obstacles, robot geometries, and the underlying robot dynamics. The efficiency of the approach is derived from coupling sampling-based motion planning to handle the complexity arising from the obstacles and robot dynamics with multi-agent search to find solutions over a suitable discrete abstraction. We obtained the discrete abstraction by constructing roadmaps over low-dimensional configuration spaces to solve a relaxed problem that accounts for the obstacles but not the robot dynamics.

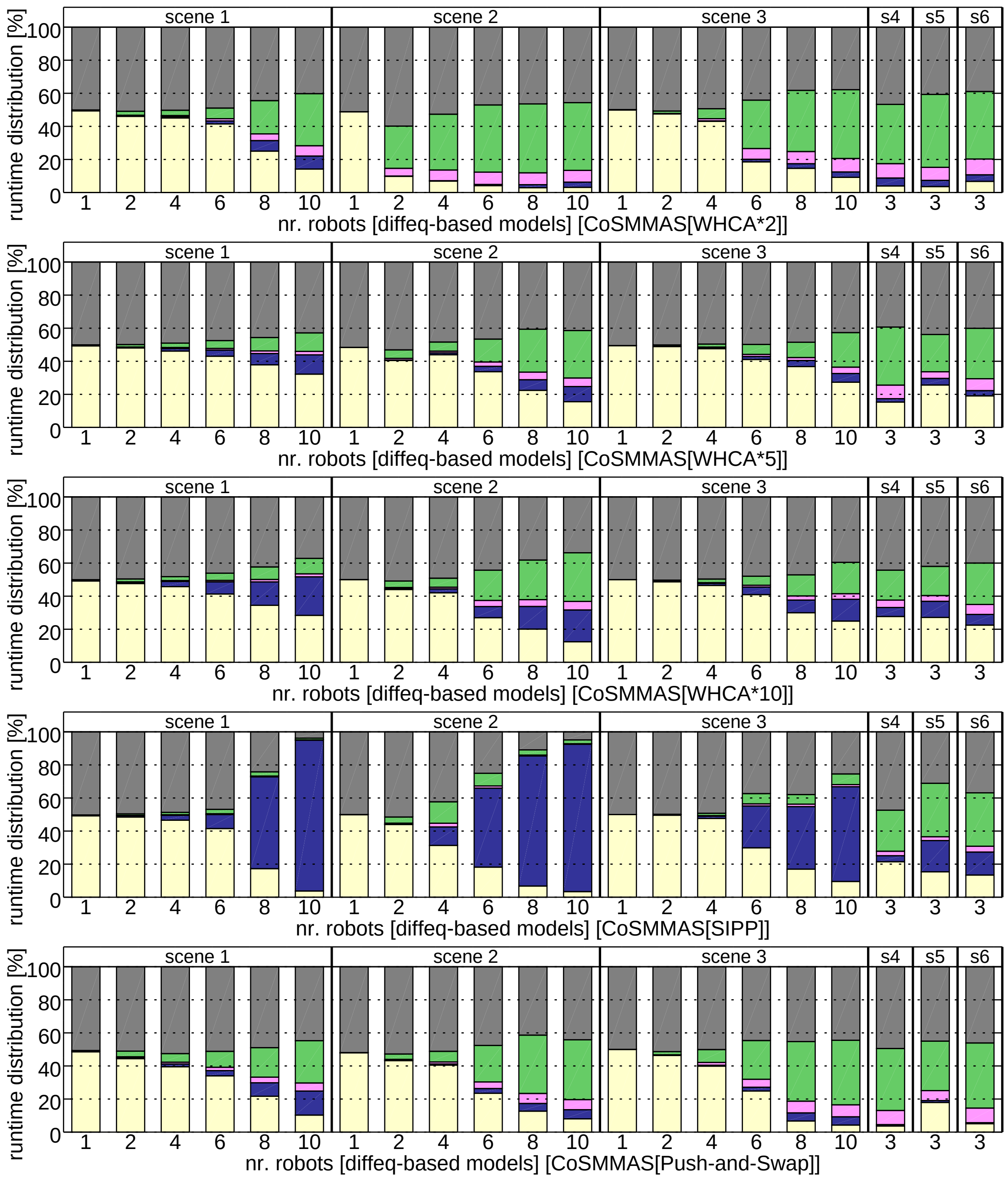


Figure 7: Runtime distribution as a percentage of the total runtime for the various components of `CoSMMAS` (from bottom to top): (a) create roadmaps (Alg. 2:1-3), (b) MultiAgentSearch (Alg. 2(b):8), (c) simulate (Alg. 2(a):9), (d) collision (Alg. 2(a):12), and (e) other. Similar distributions are obtained also for the instances with the physics-based vehicle models.

Experiments demonstrated significant speedups over related work, especially in scenarios where cooperation among robots is required to reach the goals.

This work opens up several research directions. Advances in multi-agent search can significantly improve the efficiency and scalability of the framework by reducing the runtime to compute the non-conflicting routes. Moreover, the quality of the routes can also be improved, for example, by requiring the multi-agent search to compute non-conflicting routes that maximize the separation among the robots and the clearance from the obstacles. Such routes would be easier to follow, which would significantly reduce the runtime for the motion-tree expansion. Another direction is to enable the team of robots to perform complex tasks given by PDDL or some other high-level formalism.

Acknowledgements

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

References

Amir, O., Sharon, G., & Stern, R. (2015). Multi-agent pathfinding as a combinatorial auction. In *AAAI Conference on Artificial Intelligence*, pp. 2003–2009, Austin, TX.

Beygelzimer, A., Kakade, S., & Langford, J. (2006). Cover trees for nearest neighbor. In *International Conference on Machince Learning*, pp. 97–104, Pittsburgh, PA.

Bnaya, Z., & Felner, A. (2014). Conflict-oriented windowed hierarchical cooperative A*. In *IEEE International Conference on Robotics and Automation*, pp. 3743–3748, Hong Kong, China.

Botea, A., & Surynek, P. (2015). Multi-agent path finding on strongly biconnected digraphs. In *AAAI Conference on Artificial Intelligence*, pp. 2024–2030, Austin, TX.

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., & 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., & Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.

Coumans, E. (2012). Bullet physics engine.. http://bulletphysics.org/.

Erdem, E., Kisa, D. G., Öztok, U., & Schueller, P. (2013). A general formal framework for pathfinding problems with multiple agents. In *AAAI Conference on Artificial Intelligence*, pp. 290–296, Bellevue, WA.

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

Goldenberg, M., Felner, A., Stern, R., Sharon, G., Sturtevant, N. R., Holte, R. C., & Schaeffer, J. (2014). Enhanced partial expansion A*. *Journal of Artificial Intelligence Research*, *50*, 141–187.

Karaman, S., & 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., & 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.* Dover, New York, NY.

Khorshid, M. M., Holte, R. C., & Sturtevant, N. R. (2011). A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Symposium on Combinatorial Search*, pp. 76–83, Barcelona, Spain.

Ladd, A. M., & 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., & 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. (2006). *Planning Algorithms.* Cambridge University Press, Cambridge, MA.

LaValle, S. M. (2011). Motion planning: The essentials. *IEEE Robotics & Automation Magazine*, *18*(1), 79–89.

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

Le, D., & Plaku, E. (2017). Cooperative multi-robot sampling-based motion planning with dynamics. In *International Conference on Planning and Scheduling*, pp. 513–521, Pittsburgh, PA.

Luna, R., & Bekris, K. E. (2011). Push and swap: Fast cooperative path-finding with completeness guarantees. In *International Joint Conference on Artificial Intelligence*, pp. 294–300, Barcelona, Spain.

Muja, M., & Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(11), 2227–2240.

Narayanan, V., Phillips, M., & Likhachev, M. (2012). Anytime safe interval path planning for dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4708–4715, Algarve, Portugal.

Pan, J., Chitta, S., & Manocha, D. (2012). FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation*, pp. 3859–3866, St. Paul, MN.

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

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

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

Ryan, M. R. K. (2008). Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research*, *31*, 497–542.

Sajid, Q., Luna, R., & Bekris, K. E. (2012). Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *Symposium on Combinatorial Search*, pp. 88–96, Niagara Falls, Canada.

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

Sharon, G., Stern, R., Goldenberg, M., & Felner, A. (2013). The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, *195*, 470–495.

Shewchuk, J. R. (2002). Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, *22*(1-3), 21–74.

Silver, D. (2005). Cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 1, pp. 117–122, Marina del Ray, CA.

Smith, M. (2006). Open dynamics engine.. www.ode.org.

Solovey, K., Salzman, O., & Halperin, D. (2016). Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. *International Journal of Robotics Research*, *35*(16), 501–513.

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

Standley, T., & Korf, R. (2011). Complete algorithms for cooperative pathfinding problems. In *International Joint Conference on Artificial Intelligence*, pp. 668–673, Barcelona, Spain.

Sturtevant, N. R., & Buro, M. (2006). Improving collaborative pathfinding using map abstraction. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 80–85, Marina del Rey, CA.

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

Svancara, J., & Surynek, P. (2017). New flow-based heuristic for search algorithms solving multi-agent path finding. In *International Conference on Agents and Artificial Intelligence*, pp. 451–458, Porto, Portugal.

Tracy, D. J., Buss, S. R., & Woods, B. M. (2009). Efficient large-scale sweep and prune methods with AABB insertion and removal. In *IEEE Conference on Virtual Reality*, pp. 191–198, Lafayette, LA.

Wagner, G., & Choset, H. (2015). Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, *219*, 1–24.

Wagner, G., Kang, M., & Choset, H. (2012). Probabilistic path planning for multiple robots with subdimensional expansion. In *IEEE International Conference on Robotics and Automation*, pp. 2886–2892, St. Paul, MN.

Wilde, B. d., Ter Mors, A. W., & Witteveen, C. (2014). Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, *51*, 443–492.

Yu, J., & LaValle, S. M. (2013). Structure and intractability of optimal multi-robot path planning on graphs.. In *AAAI Conference on Artificial Intelligence*, pp. 1443–1449, Bellevue, WA.