To appear in the *Journal of Experimental & Theoretical Artificial Intelligence*
Vol. 00, No. 00, Month 20XX, 1–23

Interactive Search for Action and Motion Planning with Dynamics

E. Plaku* and Duong Le

Department of Electrical Engineering and Computer Science
Catholic University of America, Washington DC, USA 22064

(*Received 00 Month 20XX; accepted 00 Month 20XX*)

This paper proposes an interactive search approach, termed INTERACT, which couples sampling-based motion planning with action planning in order to effectively solve the combined task- and motion-planning problem. INTERACT is geared toward scenarios involving a mobile robot operating in a fully-known environment consisting of static and movable objects. INTERACT makes it possible to specify a task in the planning-domain definition language (PDDL) and automatically computes a collision-free and dynamically-feasible trajectory that enables the robot to accomplish the task. The coupling of sampling-based motion planning with action planning is made possible by expanding a tree of feasible motions and partitioning it into equivalence classes based on the task predicates. Action plans provide guidance as to which a equivalence class should be further expanded. Information gathered during the motion-tree expansion is used to adjust the action costs in order to effectively guide the expansion toward the goal. This interactive process of selecting an equivalence class, expanding the motion tree to implement its action plan, and updating the action costs and plans to reflect the progress made is repeated until a solution is found. Experimental validation is provided in simulation using a robotic vehicle to accomplish sophisticated pick-and-place tasks. Comparisons to previous work show significant improvements.

Keywords: sampling-based motion planning; AI planning; PDDL; robot dynamics

1. Introduction

Addressing the combined task- and motion-planning problem is becoming increasingly important as a growing number of diverse robotics applications in navigation, search-and-rescue, manipulation, and surgical procedures involve reasoning with both discrete actions and continuous motions. In these settings, there is a need to increase the capabilities of the robotic system so that it can plan a collision-free and dynamically-feasible motion trajectory that enables the robot to accomplish the specified task. This problem has two crucial aspects: (i) planning in the space of high-level actions, and (ii) planning in the continuous space of feasible motions.

Action planning seeks to break down the overall task into a sequence of logical discrete actions, often relying on a simplified and abstract representation of the world that does not take into account the robot dynamics, obstacle avoidance, and other complex physical constraints. This has made it possible to specify high-level tasks using expressive logical models and planning-domain definition languages such as STRIPS (Fikes & Nilsson, 1971), PDDL (McDermott et al., 1998), ADL (Pednault, 1994), HAL (Marthi,

*Corresponding author. Email: plaku@cua.edu

Russell, & Wolfe, 2007), and efficiently plan high-level actions that accomplish the specified tasks (Bonet & Geffner, 2001; Botea, Enzenberger, Müller, & Schaeffer, 2005; Coles & Coles, 2011; Helmert, 2006; Hoffmann & Nebel, 2001; Nakhost & Müller, 2012; Richter & Westphal, 2010; Rintanen, 2012; Sievers, Ortlieb, & Helmert, 2012).

From the other end of the spectrum, motion planners based on probabilistic sampling allow for richer world representations that take into account geometric constraints imposed by obstacles and differential constraints imposed by robot dynamics (Şucan & Kavraki, 2012; Devaurs, Simeon, & Cortés, 2013; Hsu, Kindel, Latombe, & Rock, 2002; Ladd & Kavraki, 2005; LaValle & Kuffner, 2001; Le & Plaku, 2014; Plaku, 2013, 2015; Plaku, Kavraki, & Vardi, 2010; Wells & Plaku, 2015). This has been possible by considering simpler tasks, such as reachability, where the objective is to reach a desired location while avoiding collisions.

Decoupled approaches, which treat action and motion planning separately, have had limited success due to the intertwined dependencies between the high-level aspects of the task and the low-level motions needed to implement the task. Due to constraints imposed by robot dynamics and collision-avoidance requirements, it may be difficult or impossible to implement a particular action. Thus, a central issue is determining which actions are feasible. This gives rise to circular dependencies as the feasibility of an action plan is determined by planning a motion trajectory that implements the action plan but generating such motion trajectory requires the action plan to be feasible. These issues are further exacerbated by the computational complexity of motion planning. In fact, motion planning with dynamics is undecidable (Branicky, 1995). Sampling-based motion planners offer only probabilistic completeness, which guarantees that a solution will be found if it exists with probability approaching one but cannot determine whether or not a solution exists (Choset et al., 2005).

To address these intertwined dependencies, this paper proposes INTERACT (Interactive Search for Action and Motion Planning), which tightly couples sampling-based motion planning with action planning. INTERACT makes it possible to specify a high-level task in PDDL and automatically computes a collision-free and dynamically-feasible trajectory that enables the robot to accomplish the task. The key insight is to simultaneously search and selectively explore the discrete space of actions and the continuous space of feasible motions. Starting from the initial state, a motion tree is incrementally expanded by adding collision-free and dynamically-feasible motions as branches. The interaction between action planning and motion planning is made possible by partitioning the motion tree into equivalence classes based on the task predicates. Action plans computed for each equivalence class provide guidance as to which and how an equivalence class should be further expanded. A workspace decomposition is used to facilitate the expansion of an equivalence class along regions compatible with its action plan. If no progress is made, the corresponding action costs are increased to reduce the likelihood of being included in future action plans, and a new action plan is computed for the selected equivalence class. This interactive process of selecting an equivalence class, expanding the motion tree to implement its action plan, and updating the action costs and plans is repeated until a solution is found. By tightly coupling sampling-based motion planning with action planning, INTERACT is able to efficiently compute collision-free and dynamically-feasible trajectories that satisfy PDDL specifications.

1.1. *Related Work*

Approches seeking to combine task and motion planning can be broadly divided into three categories: (i) the motion planner drives the overall search and uses the symbolic

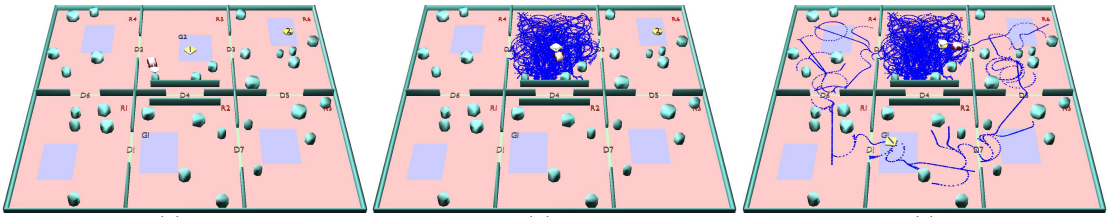
action planner to compute action plans that can guide the search; (ii) the symbolic action planner drives the overall search and uses the motion planner to geometrically instantiate symbolic actions; and (iii) the planning interface seeks to capture both the task description and the motion-planning semantics associated with the actions.

The proposed approach, INTERACT, and the `aSyMov` planner (Cambon, Alami, & Gravot, 2009) belong to the first category. `aSyMov` combines probabilistic roadmaps (PRMs) (Kavraki, Švestka, Latombe, & Overmars, 1996) with symbolic action planning. While INTERACT shares with `aSyMov` the general idea of using the PDDL planner to guide sampling-based motion planning, it makes several contributions. First, `aSyMov` cannot take into account the differential constraints imposed by the robot dynamics since each PRM edge requires exact steering between its configurations. Exact solutions to this two-boundary value problem are available only in limited cases while numerical solutions leave large gaps in the roadmap and render its construction impractical (Cheng, Frazzoli, & LaValle, 2008; Keller, 1992). In contrast, INTERACT expands a motion tree, which does not require exact steering, but only the ability to simulate the dynamics. Other distinct features of INTERACT include the introduction of the equivalence classes, the use of the workspace decomposition to facilitate the motion-tree expansion, and the update of the action costs and plans to reflect the progress made by the motion-tree expansion.

Another set of approaches considers the motion planner as a geometric reasoner which is invoked by the action planner during the search to geometrically instantiate the symbolic actions. In this context, PDDL is extended with semantic attachments which allows for the evaluation of grounded predicates and changing of fluents by external functions, making it possible to seamlessly invoke a path planner (Dornhege et al., 2012). Further work along this line developed an object-oriented language to make it easier to combine task and motion planning (Hertle, Dornhege, Keller, & Nebel, 2012). Due to the complexity of motion planning (undecidable when dynamics are considered and PSPACE-complete when dynamics are ignored), the geometric reasoner is often based on a sampling-based approach, which is computationally efficient but offers only probabilistic completeness. This makes it impossible to distinguish between an infeasible action and the inability of the path planner to find a solution within the given time limit. As a result, such approaches are often incomplete. Moreover, the path planner is invoked numerous times to instantiate different actions, which can considerably increase the runtime.

Hierarchical task networks (HTNs) and other hierarchical formulations are also used in combination with geometric path planners to break down the task into subtasks and prune the search space (Kaelbling & Lozano-Pérez, 2011; Wolfe, Marthi, & Russell, 2010). The search space is also pruned by combining constraints from symbolic action plans and the kinematic model of the robot into a network of geometric constraints (Lagriffoul, Dimitrov, Bidot, Saffiotti, & Karlsson, 2014).

Other approaches seek to merge task- and motion-planning at the level of the problem description by using formalisms to capture both the task description and the motion-planning semantics associated with the actions. Examples include knowledge-based reasoning (Choi & Amir, 2009), high-level causal reasoning combined with low-level geometric reasoning using the action description language $\mathcal{C}^+$ (Erdem, Haspalamutgil, Palaz, Patoglu, & Uras, 2011), and answer-set programming (Erdem, Aker, & Patoglu, 2012). Narrative-based reasoning has also been used to postdict explanations that describe what may have caused deviations from the planned motions (Eppe & Bhatt, 2013). Other approaches have used geometric volumes in order to provide an intermediate representation that captures both the discrete symbolic actions and the continuous robot motions (Gaschler et al., 2013). Another line of research developed an interface aiming to link geometric reasoning with symbolic planning by incorporating geometric entities into HTN



(a) (b) (c)

Figure 1. (a) Example of a pick-and-place task where the robot is required to transfer each object labeled with i to the dropoff area labeled with G_i. The robot can pick up only one object at a time, and, while carrying an object, is allowed to travel only to an adjacent empty room. Figure is best viewed in color, on screen, and zoomed in. (b) Snapshot of the motion-tree expanded by INTERACT when guided by the action plan that seeks to transfer object 1 to G_1 by passing from room R_5 to R_2 via door D_4. (c) INTERACT penalizes the action `move(R5, R2, D4)` since it fails to make progress, and, discovers an alternative plan that it is able to implement.

planning (de Silva, Pandey, & Alami, 2013).

1.2. *Contribution*

In contrast to `aSyMov` (Cambon et al., 2009) and other geometric reasoners, INTERACT plans dynamically-feasible motions. Considering the combined task- and motion-planning problem only in a geometric setting could lead to infeasible motion plans. A geometric solution, for example, may require the vehicle to instantaneously stop, turn in place, or move sideways, which could be infeasible due to the differential constraints imposed by the dynamics. As an illustration, referring to the pick-and-place task shown in Fig. 1, the robot is required to transfer the object labeled with i to the dropoff area labeled with G_i. An action plan could require transferring O_1 to G_1 by passing from room R_5 to R_2, and then moving from R_2 to R_3 to R_4 to pickup O_2, and finally transferring O_2 to G_2 by passing from R_4 to R_5. Although such action plan could be implemented geometrically (as the robot fits through the narrow passages that connect room R_5 to R_2), it is difficult to implement when considering the dynamics due to the inability of the vehicle to move sideways. When INTERACT is run with this problem instance, it quickly penalizes the above action plan (since it fails to make progress), and discovers an alternative action plan that it is able to implement while taking the dynamics into account.

INTERACT draws from earlier work on using discrete search to guide sampling-based motion planning (Le & Plaku, 2014; Plaku, 2013, 2015; Plaku et al., 2010). It also took into account specifications given in a fragment of Linear Temporal Logic (LTL) (McMahon & Plaku, 2014). INTERACT leverages from this work the idea of tightly coupling the layers of planning and shows how to effectively combine sampling-based motion planning with symbolic action planning in order to account for PDDL task specifications.

PDDL task specifications were also taken into account by SMAP (Plaku & Hager, 2010). SMAP offered a proof-of-concept that sampling-based motion planning can be used in conjunction with action planning. INTERACT makes significant improvements to the interplay between the planning layers and offers several contributions: (i) partitioning the motion tree into equivalence classes based on task predicates; (ii) using action planning to determine which and how to expand each equivalence class; (iii) using a workspace decomposition to facilitate expansion of an equivalence class along regions compatible with its action plan; (iv) using information gathered during motion-tree expansion to update action costs in order to find more suitable action plans. Comparisons to SMAP show speedups of one order of magnitude.

2. Problem Formulation

INTERACT, as `aSyMov`, numerous geometric reasoners, and SMAP, assumes a fully-known environment consisting of static obstacles and movable objects. INTERACT is geared for problems involving a mobile robot. Since INTERACT couples sampling-based motion planning with action planning, it is best suited for PDDL predicates and actions whose semantics are related to the robot motion.

This section describes the task domain, the robot model, the world model, the interpretation of the task domain in the world model, and the problem statement. The task domain defines the discrete space and actions associated with the PDDL task planner. The robot model defines the geometry, control inputs, and the differential equations of motions of the robot. The world, which is modeled in a continuous setting, is comprised of the robot, movable objects, and static obstacles. The world gives meaning to the discrete predicates and actions. Each world state is mapped to a discrete state according to the predicates that it satisfies. Robot motions, which are defined as sequences of world states, make it possible to implement the discrete actions in the continuous world.

2.1. *Discrete Task Specification*

This work uses PDDL (McDermott et al., 1998) to allow for sophisticated task specifications. It assumes a closed-world formulation and requires a PDDL planner that supports action costs and seeks to minimize the plan cost. The domain is defined as a tuple $\mathcal{D} = (\mathcal{O}, \mathcal{P}, \mathcal{C}, \mathcal{A})$ comprised of objects and schemas for predicates, costs, and actions. Predicates express relations among objects. A predicate schema $P \in \mathcal{P}$ with arity n can be considered as a Boolean function $P : \mathcal{O}^n \to \{\top, \bot\}$. An instantiation $P(o_1, \ldots, o_n)$ is referred to as a grounded literal, which is considered positive when $P(o_1, \ldots, o_n) = \top$. Cost schemas provide a mechanism to associate costs with actions. Conceptually, a cost schema $C \in \mathcal{C}$ with arity n corresponds to a function of the form $C : \mathcal{O}^n \to \mathrm{R}$. An instantiation of C is given as a pair $(C(o_1, \ldots, o_n), r)$ where $o_1, \ldots, o_n$ denote the objects and r denotes the cost. Action schemas define the available actions. Each action schema is of the form $\langle A(v_1, \ldots, v_n), A_{\text{pre}}, A_{\text{effect}} \rangle$, where A is the action name, each v_i is an object variable, A_{pre} defines the conditions that must hold in order for A to be applicable, and A_{effect} defines the effect of applying A in terms of the conditions that become true, the conditions that become false, and the changes that occur to the total cost.

A planning instance for a domain $\mathcal{D} = (\mathcal{O}, \mathcal{P}, \mathcal{C}, \mathcal{A})$ is defined by an initial discrete state q and a goal ϕ_{goal}. Each discrete state is of the form $(g_1, \ldots, g_m, c_1, \ldots, c_k)$ where each g_i denotes a positive grounded literal and each c_j denotes an instantiation of a cost schema. This gives rise to the notion of the discrete space, denoted by $\mathcal{Q}$, as the set containing all the discrete states. The goal, ϕ_{goal}, which is specified as a conjunction of positive and negative grounded literals, indicates the conditions that must become true and those that must become false. Discrete states are transformed via actions. An action $a \in A$ corresponds to an instantiation of the action schema A with objects $o_1, \ldots, o_n$ as its arguments, i.e., $A(o_1, \ldots, o_n)$. Given an action $a \in A$ and a discrete state $q \in \mathcal{Q}$ that satisfies a's precondition, the notation $q_{\text{new}} \leftarrow \textsc{apply}(q, a)$ is used to denote the discrete state $q_{\text{new}} \in \mathcal{Q}$ obtained by applying the effects of a to q. The objective of the PDDL planner is then to compute an action plan $a_1, \ldots, a_\ell$ which transforms the discrete state q to a discrete state that satisfies ϕ_{goal} while seeking to minimize the total cost of the action plan. In the context of the overall approach, the PDDL planner will be invoked numerous times starting from different discrete states.

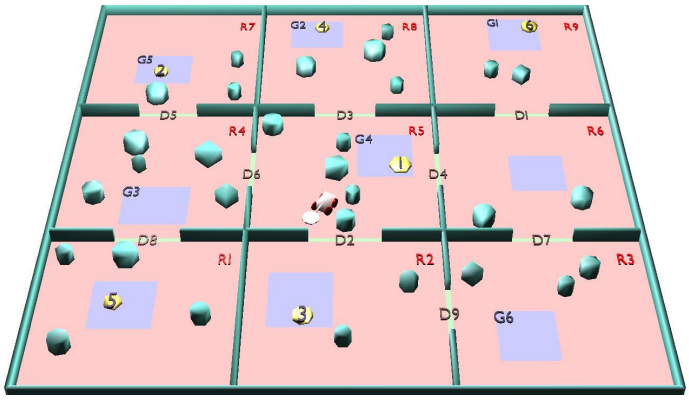


Figure 2. An example of the pick-and-place task domain. Videos of solutions obtained by the approach can be found in the supplementary material. Figure best viewed in color and on screen.

Example: Pick-and-Place Task

In the pick-and-place task, as shown in Fig. 1 and 2, the environment consists of several rooms connected by doors. The task requires the robot to pickup and place each object O_i (shown in yellow) to the corresponding dropoff area G_i (shown as a blue rectangle) while avoiding collisions. The robot can pick up only one object at a time. While carrying an object, the robot is allowed to travel only to an adjacent empty room. The robot is allowed to release the object only in the dropoff area of an empty room (it does not matter whether or not the dropoff area is labeled as a goal).

The planning domain has objects $\mathcal{O} = \{O_1, \ldots, O_n, R_1, \ldots, R_m, D_1, \ldots, D_d\}$, where O_i, R_j, and D_k denote the i-th movable object, j-th room, and k-th door, respectively. The predicate schemas $\mathcal{P}$ consists of `(robotInRoom ?Ri)`, `(objInRoom ?Oi ?Rj)`, `(connects ?Ri ?Rj ?Dk)`, `(empty ?Ri)`, `(carry ?Oi)`, and `(robotEmpty)` which indicate whether or not the robot is in room R_i, the object O_i is in room R_j, the door D_k connects rooms R_i and R_j, the room R_i is empty, the robot is carrying object O_i, and the robot is not carrying any object, respectively. The action schemas $\mathcal{A}$ consist of `move(?Ri ?Rj ?Dk)`, `pickup(?Ri ?Oj)`, `moveWithObject(?Ri ?Rj ?Dk ?Ol)`, and `release(?Ri ?Oj)`, which indicate that the robot moves from room R_i to R_j via door D_k, robot picks up object O_j in room R_i, robot moves from room R_i to R_j via door D_k while carrying object O_ℓ, and robot releases object O_j in room R_i, respectively. There are also costs associated with each of these actions. Definitions of the `move` and `pickup` action schemas are shown below. The full domain definition is provided in the supplementary material.

```
(:action move :parameters (?Ri ?Rj - room ?d - door)
 :precondition (and (robotInRoom ?Ri) (connects ?Ri ?Rj ?d) (robotEmpty))
 :effect (and (not(robotInRoom ?Ri)) (robotInRoom ?Rj) (increase (total-cost) (moveCost ?Ri ?Rj ?d))))

(:action pickup :parameters (?Ri - room ?o - movable)
 :precondition (and (robotInRoom ?Ri) (objInRoom ?o ?Ri) (robotEmpty))
 :effect (and (carry ?o) (not(robotEmpty)) (increase (total-cost) (pickupCost ?Ri ?o))))
```

2.2. *Task Interpretation in the Continuous World*

The world $\mathcal{W}$, which gives meaning to the predicates and actions, is modeled in a continuous setting which describes the robot, movable objects, and static obstacles.

2.2.1. Robot Model

The robot model is defined by specifying the geometry, control inputs, and equations of motions. The robot state $s \in \mathcal{S}$, where $\mathcal{S}$ is the robot state space, defines the position, orientation, steering angle, velocity, and other components that change as a result of motion. A function $u : [0, T] \to \mathcal{U}$ indicates the control inputs, e.g., acceleration, steering velocity, that are applied to the robot at each time step. As a result of applying u, the robot state changes according to its motion equations giving rise to a motion trajectory $\zeta : [0, T] \to \mathcal{S}$. The equations of motions $f : \mathcal{S} \times U \to \dot{\mathcal{S}}$ are specified as a set of differential equations. Hence, ζ is obtained by numerically integrating f, i.e.,

$$\zeta(t) = s + \int_{h=0}^{t} f(\zeta(h), u(h))dh. \tag{1}$$

As an example, for a car-like vehicle, the robot state is defined as $s = (x, y, \theta, v, \psi)$, which consists of the position (x, y), orientation (θ), velocity (v), and steering angle (ψ). The robot is controlled by setting the acceleration (u_a) and the rotational velocity of the steering angle (u_ω). The differential equations of motion are defined as

$$\dot{x} = v\cos(\theta)\cos(\psi), \quad \dot{y} = v\sin(\theta)\cos(\psi), \quad \dot{\theta} = v\sin(\psi)/L, \quad \dot{v} = u_a, \quad \dot{\psi} = u_\omega, \tag{2}$$

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

2.2.2. Mapping Discrete States to World States

For the pick-and-place task, the world state $w \in \mathcal{W}$ is defined as $w = (s, \text{cfg}_1, \ldots, \text{cfg}_n, \text{carry}_1, \ldots, \text{carry}_n)$, where s denotes the robot state, cfg_i denotes the position and orientation of the movable object O_i, and carry_i indicates whether or not the robot is carrying O_i (at most one of these variables can be true since the robot is not allowed to carry more than one object at a time). The notations $w_{[s]}, w_{[\text{cfg}_i]}, w_{[\text{carry}_i]}$ denote the robot state, configuration, and carry components associated with w.

A function $\tau : \mathcal{W} \to \mathcal{Q}$ provides the predicate semantics by mapping a world state to the corresponding discrete state. For example, `(objInRoom O1 R5)` is satisfied by $w \in \mathcal{W}$ if O_1 is in R_5 when placed according to $w_{[\text{cfg}_1]}$. As another example, `(carry O2)` is satisfied when $w_{[\text{carry}_2]}$ is true, which would indicate that the robot is carrying O_2.

A sequence of world states $[w_1, \ldots, w_\ell]$ is said to implement an action a iff each $\tau(w_1), \ldots, \tau(w_{\ell-1})$ satisfies a's precondition and $\tau(w_\ell) = \text{APPLY}(\tau(w_{\ell-1}), a)$. As an example, `move(R5, R2, D2)` is implemented when $w_1, \ldots, w_{\ell-1}$ places the robot in room R_5 and w_ℓ places the robot in room R_2 where the transition from R_5 to R_2 occurs via door D_2. As another example, `pickup(R2, O3)` is implemented when $w_1, \ldots, w_{\ell-1}$ places the robot in R_5 and w_ℓ places the robot in R_5 in a position where it has picked up O_3. A sequence of world states $[w_1, \ldots, w_{n_1}, \ldots w_{n_2}, \ldots, w_{n_\ell}]$ is said to accomplish an action plan $a_1, \ldots, a_\ell$ if $[w_{n_{i-1}}, \ldots, w_{n_i}]$ implements a_i for each $i \in \{1, \ldots, \ell\}$, where $n_0 = 1$.

Note that this definition requires the intermediate world states $w_{n_{i-1}}, \ldots, w_{n_i-1}$ to satisfy a_i's precondition as opposed to requiring only w_{n_i-1}. In this way, the robot motions adhere to the action plan. As an example, referring to Fig. 2, consider the action plan `move(R5 R2 D2), pickup(R2 O3)`, and suppose the robot is in room R_5. By requiring the intermediate world states to satisfy the preconditions of the actions, the robot can implement the action plan only by moving from room R_5 to R_2 and then picking up O_3. If only w_{n_i-1} was required to satisfy a_i's precondition, then the action plan could be satisfied by a sequence of world states which does not adhere to the intent of the action

plan, i.e., robot moves from R_5 to R_2, then to R_3, back to R_2, and finally picks up O_3.

2.2.3. World Simulator

For the pick-and-place task, the robot, as shown in Fig. 2, is equipped with a circular magnetic handle. The robot can pick up an object O_i by activating the magnet when the centroid of O_i is inside the circular handle. Once picked up, the robot carries the object until it decides to release it inside an empty dropoff area.

From a motion-planning perspective, a function

$$w_{\text{new}} \leftarrow \text{SIMULATE}(w, u, dt, a) \tag{3}$$

encapsulates how the world state changes as the result of the robot motions and actions. In this formulation, w_{new} is the new world state obtained when the input control u is applied to the world state w for one time step dt, where a is the action that needs to be implemented. For example, if a is a move action, $w_{\text{new}[s]}$ is obtained by applying u to $w_{[s]}$ and numerically integrating the equations of motions for one time step. The other components of the world state remain unchanged. If a is a pickup action, e.g., `pickup(R5 O1)`, w_{new} is first updated as in a move action. The simulator then checks if the centroid of the object to be picked up (O_1 in the example) is inside the circular handle of the robot. If so, then $w_{\text{new}[\text{carry}_1]}$ is set to true to signify that the robot activated the magnet and picked up the object. If a is a move-with-object action, the configuration of the object being carried is updated based on the robot state to account for the fact that it is attached to the robot. If a is a release action, the robot state and the configuration of the object being carried are first updated as in the move-with-object action. The simulator then checks if the object being carried is inside the empty dropoff area associated with the release action. If so, the object is released and each $w_{\text{new}[\text{carry}_i]}$ is set to false.

Note that w_{new}, obtained by $\text{SIMULATE}(w, u, dt, a)$, does not have to correspond to a world state where the action a is accomplished. In fact, w_{new} could correspond to some intermediate state. As an illustration, if $a =$ `move(R5,R2,D2)` and w places the robot in room R_5, then w_{new} can still place the robot in room R_5. As another example, if $a =$ `pickup(R2, O3)` and w places the robot in room R_2 in a position where it cannot pickup O_3, then w_{new} can still place the robot in a position where it cannot pickup O_3. Subsequent states, obtained by applying the same or different control inputs for additional time steps, could result in a world state where a is accomplished.

A world state w is considered valid if (i) the values of the robot state for the steering angle, velocity, and other components are within desired bounds, and (ii) the robot, when placed according to $w_{[s]}$, is not in collision with the obstacles or movable objects that it is not carrying. This is encapsulated by a function $\text{VALID} : \mathcal{W} \rightarrow \{\top, \bot\}$ which is used by the approach to check the validity of the world states.

2.3. *Problem Statement*

Given the world $\mathcal{W}$ (including descriptions of the robot model, movable objects, static obstacles, SIMULATE, and VALID), a planning domain $(\mathcal{O}, \mathcal{P}, \mathcal{C}, \mathcal{A})$, a mapping $\tau : \mathcal{W} \rightarrow \mathcal{Q}$, an initial world state $w_{\text{init}} \in \mathcal{W}$, a goal ϕ_{goal} as a conjunction of positive and negative grounded literals, compute an action plan $a_1, \ldots, a_\ell$ and a sequence of control inputs $[u_1, \ldots, u_{n_1}, \ldots, u_{n_\ell - 1}]$ such that the sequence of world states $[w_1, \ldots, w_{n_1}, \ldots w_{n_2}, \ldots, w_{n_\ell}]$ with $w_1 = w_{\text{init}}$ resulting from applying the control inputs in succession satisfies the following:

- $[w_1, \ldots, w_{n_1}, \ldots w_{n_2}, \ldots, w_{n_\ell}]$ implements the action plan $a_1, \ldots, a_\ell$,
- $\tau(w_{n_\ell})$ satisfies ϕ_{goal}, and
- $\textsc{valid}(w_i) = \top$ for each w_i in the sequence,

where dt is the simulation time step and $w_j \leftarrow \textsc{simulate}(w_{j-1}, u_{j-1}, dt, a_i)$ for $i \in \{1, \ldots, \ell\}$ and $j \in \{n_{i-1} + 1, \ldots, n_i\}$ (with $n_0 = 0$).

3. Preliminaries

Before presenting INTERACT, we first describe the workspace decomposition and the motion tree which are used by INTERACT to facilitate the overall search.

3.1. *Workspace Decomposition*

The environment in which the robot operates is decomposed into nonoverlapping regions. This work uses triangulations as it ensures that regions in the decomposition do not overlap with obstacles (except at the boundaries). The Triangle package (Shewchuk, 2002) is used to compute the triangulation. Examples are shown in Fig. 3.

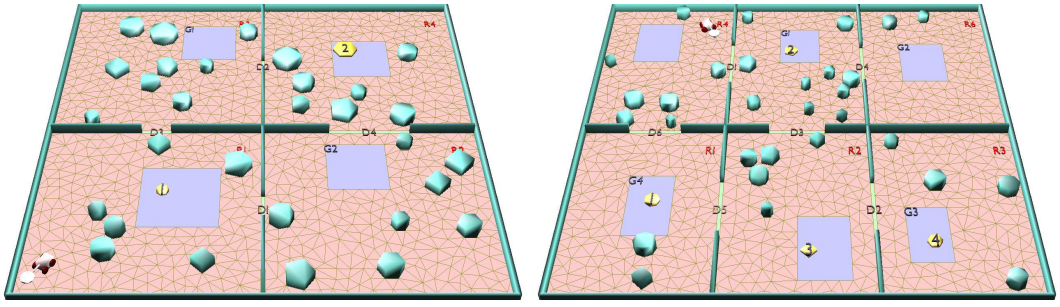


Figure 3. Examples of workspace triangulations.

The workspace decomposition is maintained as a graph $D = (R, E)$, where R denotes the regions and $E = \{(r_i, r_j \in R \text{ and } r_i, r_j \text{ are adjacent }\}$ denotes the edges. The approach relies on a function $\textsc{LocateRegion}(p)$ which determines the region $r \in R$ that contains the point p. Efficient implementations of $\textsc{LocateRegion}$ are available that run in poly-logarithmic time (de Berg, Cheong, van Kreveld, & Overmars, 2008). For convenience, the notation $\textsc{LocateRegion}(s)$ with $s \in \mathcal{S}$ is used as a shorthand to denote invoking $\textsc{LocateRegion}$ with the position of the center of the circular handle when the robot is placed according to the position and orientation specified by the robot state s.

3.2. *Motion Tree*

The search over $\mathcal{W}$ is conducted by expanding a motion tree $\mathcal{T}$. The motion tree $\mathcal{T}$ is maintained as a directed acyclic graph. Each vertex $v \in \mathcal{T}$ is associated with a collision-free world state, a discrete state, a control input, an action, and its vertex parent, denoted by $v.w$, $v.q$, $v.u$, $v.a$, $v.\text{parent}$, respectively, where $v.q = \tau(v.w)$. The edge $(v.\text{parent}, v)$ denotes the fact that $v.w$ was obtained by applying the control input to its parent, i.e.,

$$v.w \leftarrow \textsc{simulate}(v.\text{parent}.w, v.u, dt, v.a). \tag{4}$$

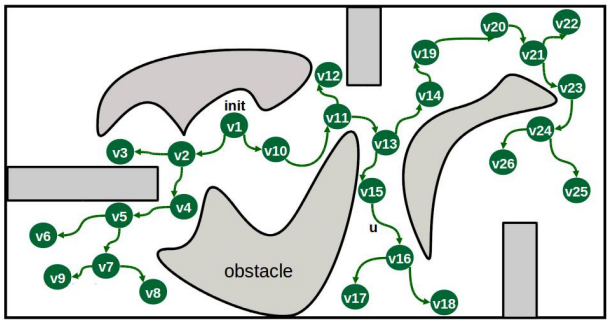


Figure 4. Example of a motion tree.

The motion tree $\mathcal{T}$ initially contains only the root vertex which is associated with the initial world state w_{init} (its control input and action are set to `null`). The motion tree is incrementally expanded by adding new vertices. Fig. 4 provides an illustration. A solution is found when a vertex v_{new} is added to $\mathcal{T}$ such that $\tau(v_{\text{new}}.w)$ satisfies ϕ_{goal}. The solution is then obtained as the sequence of the world states associated with the vertices that connect the root of $\mathcal{T}$ to v_{new}.

4. Method

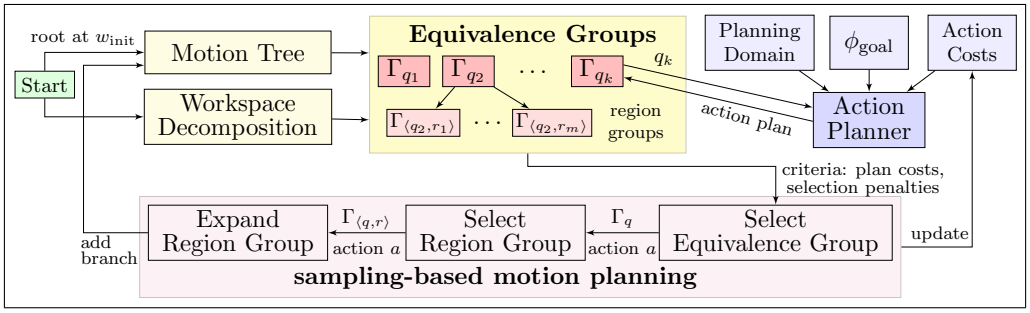


Figure 5. A schematic representation of INTERACT.

Before describing the algorithmic details of INTERACT, we provide an overview that focuses on the planning layers and their interplay. A schematic representation is shown in Fig. 5. Due to the intertwined dependencies between task and motion planning, the success of the search depends on the ability of INTERACT to effectively explore the search space. For this reason, INTERACT leverages τ, which maps a world state to its corresponding discrete state, to partition the motion tree $\mathcal{T}$ into equivalent groups. In fact, from an action-planning perspective, world states w_i and w_j carry the same information when $\tau(w_i) = \tau(w_j)$. In this way, vertices of $\mathcal{T}$ are grouped together based on the discrete states. More precisely, each discrete state $q \in \mathcal{Q}$ defines an equivalence class, denoted as Γ_q, which contains all the vertices in $\mathcal{T}$ that map to q, i.e.,

$$\Gamma_q.\text{vertices} = \{v : v \in \mathcal{T} \text{ and } \tau(v.w) = q\}. \tag{5}$$

As a result, the vertices in $\mathcal{T}$ are partitioned into several groups according to their

respective equivalence class, i.e.,

$$\Gamma = \{\Gamma_q : q \in \mathcal{Q} \text{ and } \Gamma_q.\text{vertices} \neq \emptyset\}. \tag{6}$$

The vertices in Γ_q are further partitioned based on their corresponding region in the workspace decomposition. This is done to leverage recent advances in sampling-based motion planning which have shown that workspace decompositions are useful in guiding the motion-tree expansion (Plaku, 2013, 2015; Plaku et al., 2010). Thus, $\Gamma_{\langle q,r \rangle}$ contains all the vertices in $\mathcal{T}$ that map to the discrete state $q \in \mathcal{Q}$ and to the region $r \in R$, i.e.,

$$\Gamma_{\langle q,r \rangle}.\text{vertices} = \{v : v \in \Gamma_q.\text{vertices and } r = \textsc{LocateRegion}(v.w_{[s]})\}. \tag{7}$$

As a result, the vertices in Γ_q are partitioned into several groups according to the regions in the decomposition, i.e.,

$$\Gamma_q.\text{regions} = \{\Gamma_{\langle q,r \rangle} : r \in R \text{ and } \Gamma_{\langle q,r \rangle}.\text{vertices} \neq \emptyset\}. \tag{8}$$

As shown in Fig. 5, INTERACT leverages action planning to compute action plans for each group Γ_q. INTERACT relies on sampling-based motion planning to expand the motion tree along promising action plans. This is accomplished by first selecting a group Γ_q based on the cost of its action plan. Afterwards, INTERACT leverages the workspace decomposition to select a region $\Gamma_{\langle q,r \rangle}$ which is likely to lead to motion-tree expansions in accordance with the action plan associated with Γ_q. The motion tree is then expanded from vertices $\Gamma_{\langle q,r \rangle}$, seeking to implement the action plan. Action costs and plans are updated to reflect the progress made by sampling-based motion planning. This interplay between sampling-based motion planning and action planning is repeated until a solution is found or an upper bound on the running time is reached.

A critical aspect of INTERACT is determining the group $\Gamma_q \in \Gamma$ from which to expand $\mathcal{T}$. INTERACT leverages action planning to maintain an action plan for each Γ_q, denoted by Γ_q.actionPlan. The action plan is computed using q as the initial discrete state and ϕ_{goal} as the goal. The cost of the action plan serves as a heuristic to estimate the feasibility of satisfying ϕ_{goal}. During the selection process, preference is given to groups with low action-plan costs. This constitutes the greedy aspect of the selection. Since an action plan may be difficult or impossible to implement due to constraints imposed by the obstacles and robot dynamics, a penalty is applied to Γ_q each time it is selected for expansion. This provides the methodical aspect of the selection process which enables INTERACT to abandon an infeasible action plan and instead expand the search from new groups.

After selecting Γ_q, INTERACT seeks to expand $\mathcal{T}$ so that it can implement the first action a in Γ_q.actionPlan. INTERACT leverages the workspace decomposition to determine appropriate regions along which to expand $\mathcal{T}$ in order to implement a. As an example, if a corresponds to `move(R1, R2)`, then the workspace decomposition can be searched for a sequence of regions which allows the robot to move from room R_1 to R_2. More specifically, among the region groups $\Gamma_{\langle q,r \rangle}$ in Γ_q.regions, preference is given to those that are close to room R_2. As another example, when considering a pickup or release action, preference is given to those region groups that are close to the dropoff area. More formally, INTERACT maintains a cost with each $(\Gamma_{\langle q,r \rangle}, a)$ based on the distance along the shortest-path in the decomposition graph $D = (R, E)$ from r to the destination room or dropoff area associated with the action a. The heuristic cost is complemented with selection penalties to counteract the greediness of the selection process. After selecting $\Gamma_{\langle q,r \rangle}$, INTERACT attempts to expand $\mathcal{T}$ along the shortest-path in the decomposition

from r toward the destination associated with a.

If the expansion successfully implements a, then a new group $\Gamma_{q_{\text{new}}}$ is created. This new group will inherit the action plan from Γ_q (minus the first action) as there is no reason to abandon it since the expansion is successful. If, after several attempts, the expansion fails to implement a, then the action plan associated with Γ_q is abandoned. The cost associated with a is increased and a new action plan is computed for Γ_q. If action plans associated with other groups contain a, then their costs are also updated. This provides an effective way to account for changes in the cost of a without resorting to recomputing the action plans for all the groups, which would significantly increase the runtime.

This process of selecting a group Γ_q from Γ, expanding $\mathcal{T}$ from vertices in Γ_q to implement the first action in Γ_q.actionPlan, and updating the action costs and plans to reflect the progress made is repeated until a solution is found or the runtime limit is reached. Details of the group selection and motion-tree expansion in INTERACT follow.

4.1. *Group Selection*

INTERACT maintains an overall cost for each group Γ_q, denoted by $\text{cost}(\Gamma_q)$, based on the cost of the action plan and the number of times Γ_q has been selected previously, i.e.,

$$\text{cost}(\Gamma_q) = \text{cost}(\Gamma_q.\text{actionPlan})\, \beta^{\text{nrSel}(\Gamma_q)}. \tag{9}$$

In this way, preference is given to groups associated with low-cost action plans. The parameter $\beta > 1$ serves as a penalty factor to avoid overexploration or becoming stuck when expansions from Γ_q are infeasible. The group Γ_q with the lowest overall cost in Γ is then selected for expansion, i.e.,

$$\textsc{SelectGroup}(\Gamma) = \underset{\Gamma_q \in \Gamma}{\text{argmin}}\, \text{cost}(\Gamma_q). \tag{10}$$

After selecting Γ_q, the objective is to expand $\mathcal{T}$ from Γ_q.vertices so that it implements the first action a in Γ_q.actionPlan. To facilitate the expansion, INTERACT also maintains an overall cost for each pair $\langle \Gamma_{\langle q,r \rangle}, a \rangle$, denoted by $\text{cost}(\langle \Gamma_{\langle q,r \rangle}, a \rangle)$, and defined as

$$\text{cost}(\langle \Gamma_{\langle q,r \rangle}, a \rangle) = \text{cost}(\text{path}(D, r, a))\, \beta^{\text{nrSel}(\langle \Gamma_{\langle q,r \rangle}, a \rangle)}, \tag{11}$$

where $\text{path}(D, r, a)$ denotes the shortest path in the decomposition graph $D = (R, E)$ from r to the destination room or dropoff area associated with the action a. As before, β penalizes selections to avoid overexplorations or becoming stuck. The region group $\Gamma_{\langle q,r \rangle}$ with the lowest overall cost in Γ_q.regions is then selected for expansion, i.e.,

$$\textsc{SelectRegionGroup}(\Gamma_q, a) = \underset{\Gamma_{\langle q,r \rangle} \in \Gamma_q.\text{regions}}{\text{argmin}}\, \text{cost}(\langle \Gamma_{\langle q,r \rangle}, a \rangle). \tag{12}$$

Note that it is possible to use two different parameters, β_1 and β_2, to define the group- and region-selection penalties in Eqs. 9 and 11. In this paper, the same selection penalty, β, was used in both cases since it worked well in the experiments and simplified the process of selecting parameter values.

Algorithm 1 INTERACT

Input: problem specification; $t_{\max}$: upper bound on running time
Output: A collision-free and dynamically-feasible trajectory that satisfies the task specification or `null` if no solution is found

1: $D = (R, E) \leftarrow$ WORKSPACEDECOMPOSITION()
2: $[\mathcal{T}, \Gamma] \leftarrow$ INITIALIZE(w_{init})
3: **while** ¬solved **and** TIME $< t_{\max}$ **do**
4: $\quad \Gamma_q \leftarrow$ SELECTGROUP(Γ)
5: $\quad a \leftarrow \Gamma_q$.actionPlan.firstAction
// *Expand motion tree from Γ_q seeking to complete action a*
6: $\quad$ nrIters $\leftarrow$ RAND(minNrIters, maxNrIters); actionCompleted $\leftarrow \perp$
7: $\quad$ **while** nrIters > 0 **and** ¬solved **and** ¬actionCompleted **do**
8: $\quad\quad \Gamma_{\langle q,r \rangle} \leftarrow$ SELECTREGIONGROUP(Γ_q, a)
9: $\quad\quad$ target $\leftarrow$ SELECTTARGET($\Gamma_{\langle q,r \rangle}, a$)
10: $\quad\quad v \leftarrow$ SELECTVERTEX($\Gamma_{\langle q,r \rangle}, a$, target)
11: $\quad\quad$ actionCompleted $\leftarrow$ EXPANDTREE($\mathcal{T}, \Gamma, v$, target, a, ϕ_{goal}); nrIters $\leftarrow$ nrIters $- 1$
12: $\quad$ **if** ¬actionCompleted **then**
13: $\quad\quad$ cost(a) $\leftarrow$ INCREASEACTIONCOST(a)
14: $\quad\quad$ **for each** $\Gamma_{q'} \in \Gamma$ **do** // *update cost of action plan*
15: $\quad\quad\quad$ **if** $a \in \Gamma_{q'}$.actionPlan **then** RECOMPUTEPLANCOST($\Gamma_{q'}$.actionPlan)
16: $\quad\quad \Gamma_q$.actionPlan $\leftarrow$ PDDLPLANNER(q, ϕ_{goal})
17: **if** solved **return** solution **else return** `null`

4.2. *Overall Expansion*

Pseudocode for INTERACT is provided in Alg. 1. INTERACT starts by computing the workspace decomposition (Alg. 1:1) and initializing the motion tree (Alg. 1:2). INTERACT then proceeds iteratively by selecting a group (Alg. 1:4) and then expanding the motion tree (Alg. 1:5–16). The group selection was described in Section 4.1. The rest of this section describes the motion-tree expansion.

Let Γ_q be the group selected for expansion from Γ (Alg. 1:4). Let a be the first action of Γ_q.actionPlan (Alg. 1:5). INTERACT will seek several times to expand $\mathcal{T}$ from Γ_q.vertices to implement a (Alg. 1:6–11). Each iteration consists of selecting a region group $\Gamma_{\langle q,r \rangle}$ from Γ_q.regions, a target point p, a vertex v from $\Gamma_{\langle q,r \rangle}$.vertices, and applying control inputs to expand $\mathcal{T}$ from v toward p. As explained in Section 4.1, the region group $\Gamma_{\langle q,r \rangle}$ with the minimum cost($\langle \Gamma_{\langle q,r \rangle}, a \rangle$) is selected for expansion (Alg. 1:8). The target point p is sampled uniformly at random inside a region selected uniformly at random along the shortest path from r to the destination room or dropoff area associated with the action a (Alg. 1:9). Drawing from RRT (LaValle, 2011; LaValle & Kuffner, 2001), the closest vertex in $\Gamma_{\langle q,r \rangle}$.vertices to p is then selected for expansion (Alg. 1:10). When computing the closest vertex, the distance between a vertex v and the target point p is defined as $||\text{CENTER}(v.s) - p||$, where CENTER($v.s$) denotes the position of the center of the robot handle. Although other distance metrics can be used, this workspace distance worked well in the experiments as it relates to the robot motions.

Pseudocode for the algorithm to expand $\mathcal{T}$ from v toward p is shown in Alg. 2. Control inputs are obtained by using a PID controller which adjusts the steering wheel so that the robot vehicle points toward the target point p. The controller is applied for a number of steps (sampled uniformly at random from some lower and upper bound) (Alg. 2:1-2). After simulating the robot motion, collision checking is performed to ensure that there are no collisions (Alg. 2:3-4). If a collision is found, the expansion terminates. Otherwise, a new vertex v_{new} is added to $\mathcal{T}$ (Alg. 2:5–14). If the discrete state, q_{new}, associated with v_{new} satisfies ϕ_{goal}, then the search terminates successfully (Alg. 2:16). In such a case, the solution consists of the sequence of the world states associated with the vertices

Algorithm 2 ExpandTree($\mathcal{T}, \Gamma, v, \text{target}, a, \phi_{\text{goal}}$)

Input: $\mathcal{T}$: motion tree; Γ: partition of motion tree into groups; v: vertex in $\mathcal{T}$ from which to expand; target: point toward which to expand; a: current action; ϕ_{goal}: goal
Output: It adds new vertices to $\mathcal{T}$ and returns $\top$ iff the tree expansion implements a

1: **for** $i = 1 \ldots$ RAND(minNrSteps, maxNrSteps) **and** $\neg$solved **do**
2: $u \leftarrow$ RobotController($v.w_{[s]}$, target)
3: $w_{\text{new}} \leftarrow$ SIMULATE($v.w, u, dt, a$)
4: **if** $\neg$VALID(w_{new}) **then return** $\bot$
// *add new vertex to $\mathcal{T}$ and to corresponding group in Γ*
5: $v_{\text{new}}.[w, \text{parent}, u, r, q] \leftarrow [w_{\text{new}}, v, u,$ LocateRegion($w_{\text{new}[s]}$), $\tau(w_{\text{new}})]$; INSERT($\mathcal{T}, v_{\text{new}}$)
6: $q_{\text{new}} \leftarrow v_{\text{new}}.q$; $\Gamma_{q_{\text{new}}} \leftarrow$ FIND(Γ, q_{new})
7: **if** $\Gamma_{q_{\text{new}}} =$ `null` **then**
8: $\Gamma_{q_{\text{new}}} \leftarrow$ NewGroup(q_{new}); INSERT($\Gamma, \Gamma_{q_{\text{new}}}$)
9: **if** $q_{\text{new}} =$ APPLY($v.q, a$) **then** $\Gamma_{q_{\text{new}}}$.actionPlan $\leftarrow \Gamma_{v.q}$.actionPlan $\setminus \{a\}$
10: **else** $\Gamma_{q_{\text{new}}}$.actionPlan $\leftarrow$ PDDLPlanner($q_{\text{new}}, \phi_{\text{goal}}$)
11: INSERT($\Gamma_{q_{\text{new}}}$.vertices, v_{new})
12: $r_{\text{new}} \leftarrow v_{\text{new}}.r$; $\Gamma_{\langle q_{\text{new}}, r_{\text{new}}\rangle} \leftarrow$ FIND($\Gamma_{q_{\text{new}}}$.regions, r_{new}) *//add to appropriate region group*
13: **if** $\Gamma_{\langle q_{\text{new}}, r_{\text{new}}\rangle} =$ `null` **then**
14: $\Gamma_{\langle q_{\text{new}}, r_{\text{new}}\rangle} \leftarrow$ NewRegionGroup(r_{new}); INSERT($\Gamma_{q_{\text{new}}}$.regions, $\Gamma_{\langle q_{\text{new}}, r_{\text{new}}\rangle}$)
15: INSERT($\Gamma_{\langle q_{\text{new}}, r_{\text{new}}\rangle}$.vertices, v_{new})
16: **if** GOAL(q_{new}) **then** { solved $\leftarrow \top$; **return** $\top$ }
17: **if** $q_{\text{new}} =$ APPLY($v.q, a$) **then return** $\top$
18: $v \leftarrow v_{\text{new}}$
19: **return** $\bot$

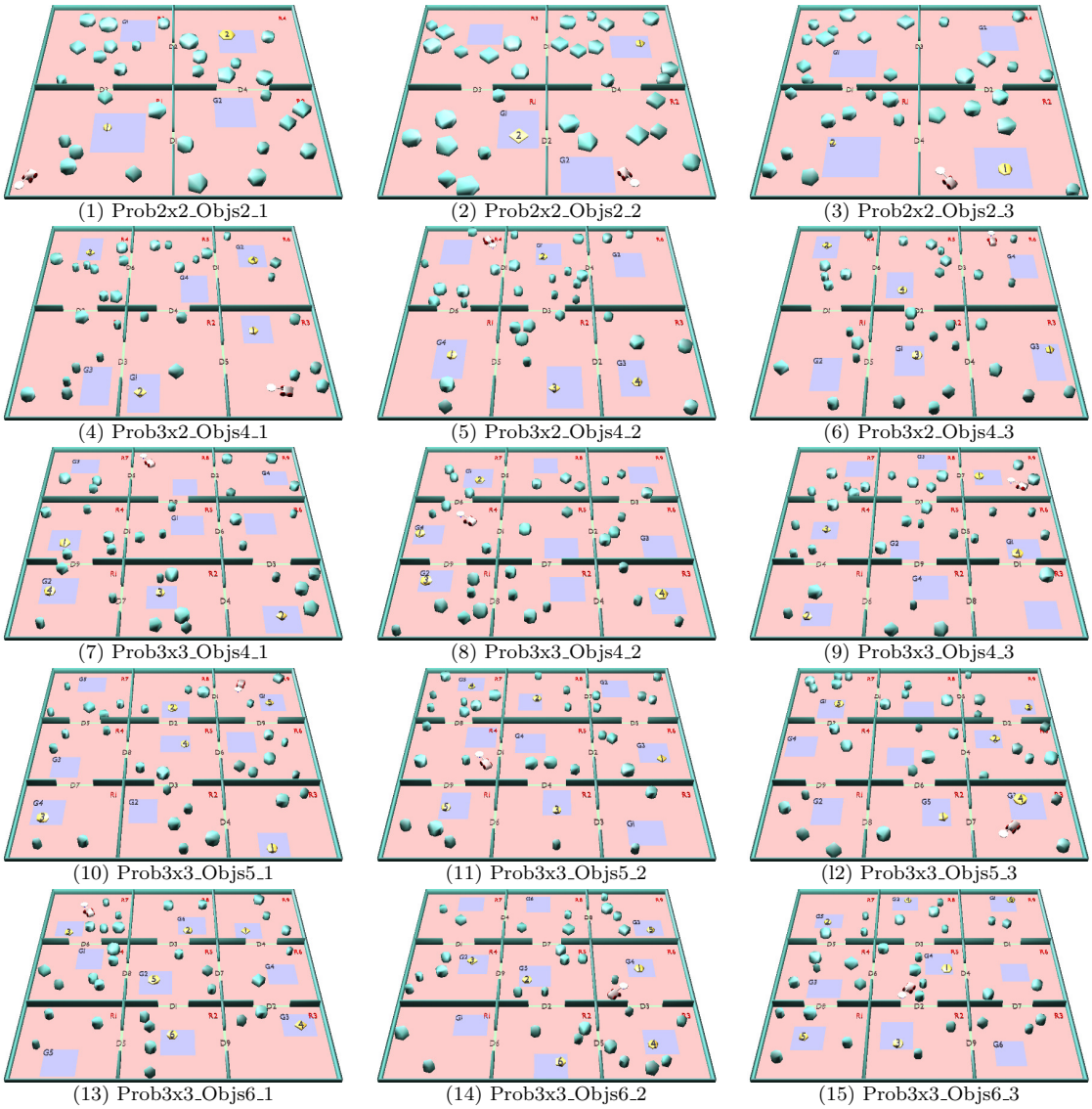
in $\mathcal{T}$ connecting its root to v_{new}. The expansion also terminates successfully when q_{new} satisfies the postcondition of the action a (Alg. 2:17).

When v_{new} is added to $\mathcal{T}$, it is also added to the corresponding group in Γ (Alg. 2:6–15). As an implementation note, Γ is maintained as a hash map. A find operation is performed with q_{new} as the key (Alg. 2:6). If $\Gamma_{q_{\text{new}}}$ is not found in Γ, it is created and inserted into the hash map; otherwise, it is retrieved from the hash map (Alg. 2:7–8). In both cases, v_{new} is inserted into $\Gamma_{q_{\text{new}}}$.vertices. A similar procedure is followed for finding $\Gamma_{\langle q_{\text{new}}, r\rangle}$ in $\Gamma_{q_{\text{new}}}$.regions, and inserting v_{new} to $\Gamma_{\langle q_{\text{new}}, r\rangle}$.vertices (Alg. 2:11-15).

An action plan is associated with $\Gamma_{q_{\text{new}}}$ when it is first created (Alg. 2:9–10). If the action a is completed, $\Gamma_{q_{\text{new}}}$ inherits the action plan (minus the first action) from its parent, $\Gamma_{v.q}$, as there is no reason to abandon it. Otherwise, the PDDL planner is invoked with q_{new} as the initial discrete state to compute an action plan for $\Gamma_{q_{\text{new}}}$ (Alg. 2:10).

As shown in Alg. 1:6–11, several attempts are made to implement the action a by expanding $\mathcal{T}$ from the selected group Γ_q. If these expansions fail to implement the action a, then its cost is doubled (Alg. 1:13). The cost doubling worked well in the experiments, but, of course, other ways of increasing the cost can also be used. The cost increase serves as a penalty to reduce the likelihood of including a in the action plans of new groups. Since the action a could be part of the action plans already computed for the other groups in Γ, their costs are recomputed (by adding the costs associated with the plan actions) to reflect the increase in the cost of the action a (Alg. 1:14–15). Also, a new action plan is computed for Γ_q since the expansion of a was not successful (Alg. 1:16).

In this way, by penalizing unsuccessful actions, INTERACT avoids becoming stuck following what could be an infeasible action. This allows INTERACT to explore alternative action plans, which, although initially had a higher cost, are now more promising. As an example, as shown in Fig. 1, INTERACT penalizes the action `move(R5, R2, D4)` when it fails to expand the motion tree from room R_5 to R_2. After the cost increase, INTERACT considers an alternative plan from R_5 to R_4, which it can easily implement.



(1) Prob2x2_Objs2_1 (2) Prob2x2_Objs2_2 (3) Prob2x2_Objs2_3

(4) Prob3x2_Objs4_1 (5) Prob3x2_Objs4_2 (6) Prob3x2_Objs4_3

(7) Prob3x3_Objs4_1 (8) Prob3x3_Objs4_2 (9) Prob3x3_Objs4_3

(10) Prob3x3_Objs5_1 (11) Prob3x3_Objs5_2 (l2) Prob3x3_Objs5_3

(13) Prob3x3_Objs6_1 (14) Prob3x3_Objs6_2 (15) Prob3x3_Objs6_3

Figure 6. Problem instances 1–15 used in the experiments (problem 0 is shown in Fig. 1). Each case is labeled as ProbAxB_ObjsN_i where $A \times B$ denotes the arrangement of rooms in an $A \times B$ grid, N denotes the number of objects, and $i \in \{1, 2, 3\}$ denotes an instance. Figures may be best viewed in color, on screen, and zoomed in.

5. Experiments and Results

Experimental validation is provided in simulation using various instances of the pick-and-place task domain (Section 2). Comparisons to related work show significant improvements. The impact of the action planner, workspace decomposition, and the selection penalty factor on the overall performance of INTERACT are also studied.

5.1. *Experimental Setup*

5.1.1. Problem Instances

Problem instances are generated by specifying the number of rooms, movable objects, and obstacles. Rooms are configured in a grid. Door placements are determined by running

Table 1. The number of actions in the optimal action plan for each of the problem instances as computed by an optimal PDDL planner starting from the initial discrete state, i.e., $\tau(w_{\text{init}})$.

problem	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
plan length	8	7	9	9	29	31	31	61	32	59	49	39	55	77	76	46

a maze generator on the grid graph. The dropoff areas, robot, movable objects, and obstacles are placed in random locations, using rejection sampling to ensure that there are no collisions. The problem-instance generator will be made publicly available.

The problem instances used in the experiments are shown in Fig. 1 and 6. Table 1 shows the length of the optimal action plan for each problem instance as computed by an optimal PDDL planner starting from the initial discrete state, i.e., $\tau(w_{\text{init}})$. These are difficult problems that even in a discrete setting require tens of actions to be solved.

5.1.2. PDDL Action Planners

As discussed in the introduction, INTERACT works with PDDL planners that take into account action costs and seek to minimize the total cost. For the experiments, we used the popular Fast Downward package (Helmert, 2006). Among the available optimization planners, Fast Downward Stone Soup 1 and 2 yielded the best results in our setting. Among the satisficing planners, a modified version of Lama worked best. The modified version ran the search as lazy weighted A* with a weight factor set to 10. When using these PDDL planners, INTERACT is referred to as INTERACT[fdss-1], INTERACT[fdss-2], and INTERACT[sat] respectively.

5.1.3. Runtime and Solution Length

The performance of a method on a problem instance is measured by running it 30 times (since it is probabilistic). A timeout of 60s is set for each run. Results report mean running time and standard deviation. The runtime includes everything from reading the input file to reporting that a solution is found. Solution length is measured as the distance traveled by the robot. Experiments were run on an Intel Core i7 (CPU: 1.90GHz, RAM: 4GB) using Ubuntu 14.10 and GNU g++-4.8.2.

5.2. *Results*

This section presents results on the anytime performance of INTERACT, comparisons with previous work, and the impact of the workspace decomposition, selection penalty factor (β), and the PDDL action planner on the overall performance of INTERACT.

5.2.1. Anytime Performance

Fig. 7(a,b) shows the percentage of successful runs of INTERACT as a function of runtime on various problem instances. Fig. 7(c) shows the anytime results when considering all the 480 runs (30 for each of the 16 problem instances). INTERACT[fdss-1], INTERACT[fdss-2], and INTERACT[sat] always found a solution. The worst runs for INTERACT[fdss-1], INTERACT[fdss-2], and INTERACT[sat] were 12.4s, 13.6s, and 29.7s, respectively. Overall, INTERACT[fdss-1] and INTERACT[fdss-2] perform equally well while INTERACT[sat] is less efficient. Since INTERACT[sat] uses a satisficing PDDL planner, the action plans can be longer which makes the motion-tree expansion computationally more demanding.

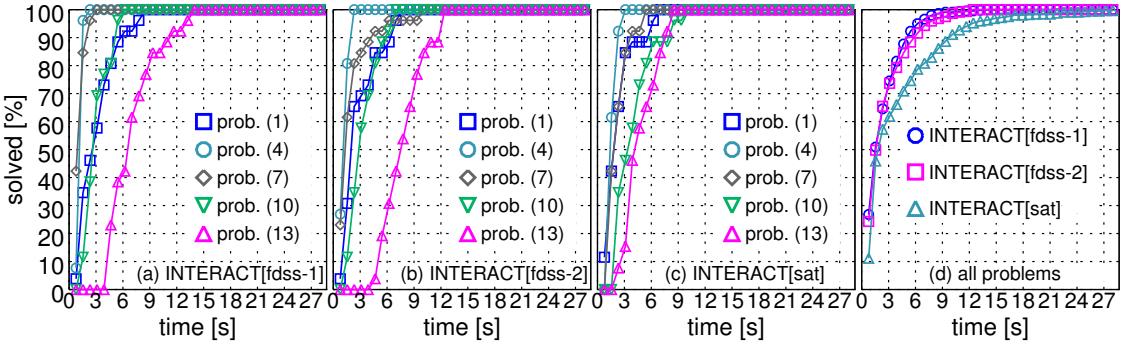


Figure 7. (a)–(b) Anytime results for INTERACT[fdss-1], INTERACT[fdss-2], and INTERACT[sat] on various problem instances. (c) Anytime results for INTERACT[fdss-1], INTERACT[fdss-2], and INTERACT[sat] over all the sixteen problem instances. In each case, the graph shows the percentage of successful runs (out of 30) that had a runtime of less than or equal to t seconds, as t varies in increments of 0.5s.

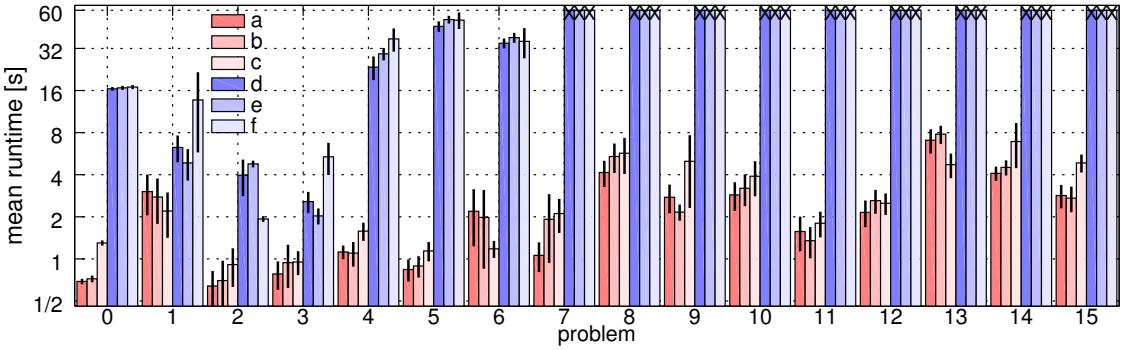


Figure 8. Mean runtime when comparing INTERACT to SMAP: (a) INTERACT[fdss-1], (b) INTERACT[fdss-2], (c) INTERACT[sat], (d) SMAP[fdss-1], (e) SMAP[fdss-2], (f) SMAP[sat]. Each bar indicates the standard deviation. Due to the significant differences in runtime, logscale is used for the y-axis with the label showing the actual value rather than its logarithm. A problem entry marked with X denotes failure by the indicated planner to obtain a solution in at least 20 out of the 30 runs within the time limit of 60s per run.

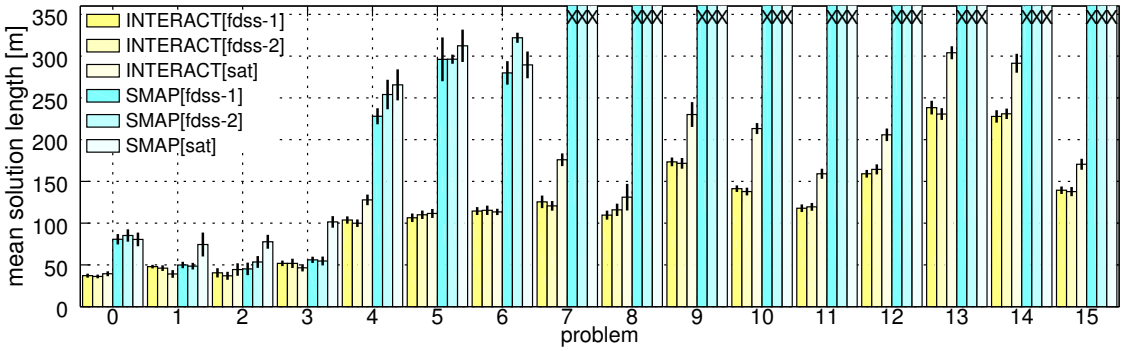


Figure 9. Mean solution length, measured as the distance traveled by the robot, when comparing INTERACT to SMAP. Each bar indicates the standard deviation. A problem entry marked with X denotes failure by the indicated planner to obtain a solution in at least 20 out of the 30 runs within the time limit of 60s per run.

5.2.2. Comparisons to Related Work

INTERACT is compared to SMAP (Plaku & Hager, 2010). As discussed in Section 1.1, other approaches, such as `aSyMov`, which combine motion and PDDL planning, do not incorporate vehicle dynamics into motion planning and so cannot be applied to our problem setting. INTERACT and SMAP have been implemented using the same simulator, collision checking, and general utilities.

Fig. 8 shows the runtime results on all problem instances. Results indicate that INTERACT is significantly faster than SMAP. The efficiency of INTERACT derives from the tight coupling of sampling-based tree search with action planning. By using action-plan costs as heuristics, INTERACT is able to expand the motion tree from groups that are estimated to be close to reaching the goal. At the same time, INTERACT imposes penalties on failed attempts to implement certain actions which makes it possible to explore new actions. The workspace decomposition effectively guides the motion-tree expansions to move the robot to the desired room or to a dropoff area where it can pickup or release an object as indicated by the action plan. This combination of greedy and methodical exploration enables INTERACT to quickly progress toward the goal while avoiding becoming stuck exploring difficult or infeasible actions.

Fig. 8 shows that INTERACT[fdss-1] and INTERACT[fdss-2] often are faster than INTERACT[sat]. Since the PDDL planner in INTERACT[sat] does not compute optimal action plans, it may take more time to implement the action plans.

Fig 9 shows the results with respect to the solution length, measured as the distance traveled by the robot. Results indicate that INTERACT produces shorter solutions than SMAP. The differences are more pronounced on the problem instances that require long sequences of actions to achieve the goal. The effective combination of sampling-based tree search and action planning enables INTERACT to find short solution trajectories.

Fig. 9 also shows that, as the problems become more complex, INTERACT[sat] generates longer solution trajectories than INTERACT[fdss-1] and INTERACT[fdss-2]. This is due to the use of a satisficing PDDL planner, which does not necessarily compute optimal action plans.

5.2.3. Runtime Distribution

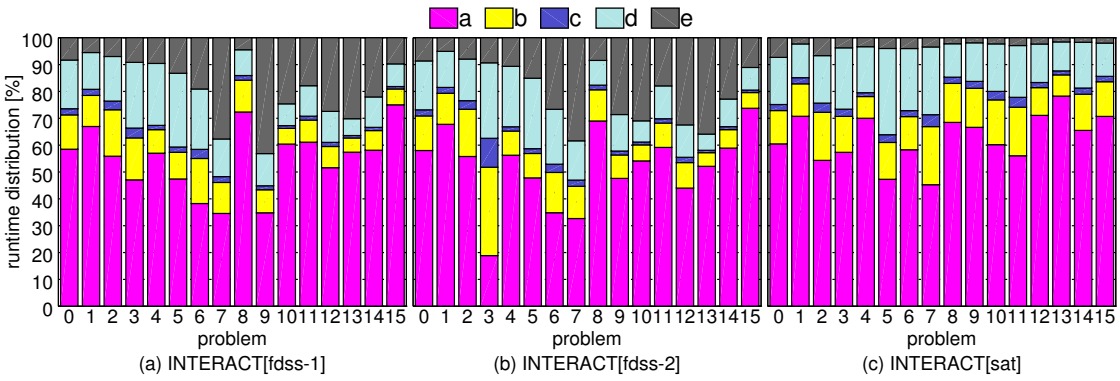
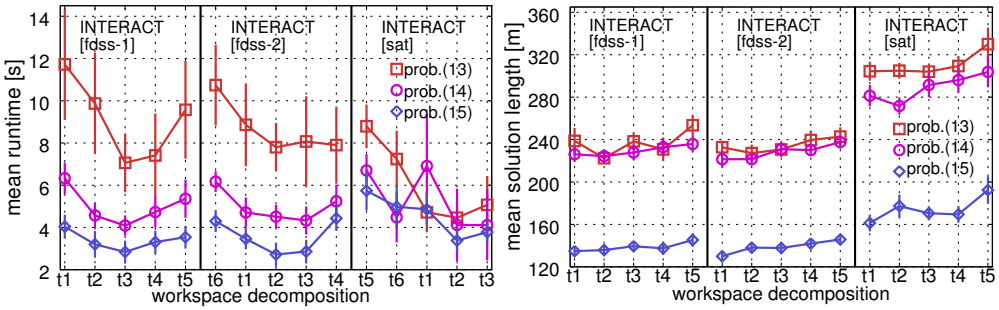


Figure 10. Runtime percentage for various components of INTERACT: (a) PDDL planner (Alg. 1:16, Alg. 2:10); (b) collision checking (Alg. 2:4); (c) simulate (Alg. 2:3); (d) add vertex (Alg. 2:5–9,11–15); (e) other.

Fig. 10 shows the runtime percentage for various components of INTERACT. Results indicate that action planning dominates the running time. Indeed, INTERACT invokes the action planner numerous times using different initial discrete states and different action costs. By doing so, INTERACT is able to expand the motion tree along relevant regions, enabling rapid progress toward achieving the goal. In other words, rather than blindly exploring the vast search space, INTERACT relies on action planning to effectively guide the motion-tree expansion. Although this increases the running time for action planning, it dramatically reduces the overall runtime of INTERACT.

5.2.4. *Impact of the Workspace Decomposition*



(a) results on runtime (b) results on solution length

Figure 11. Runtime and solution-length results when varying the average triangle area in the workspace decomposition. The workspace decompositions `t1`, . . . , `t5` correspond to triangulations where the average triangle area is 0.008%, 0.017%, 0.035%, 0.05%, and 0.08% of the overall workspace area, respectively.

Fig. 11 shows the impact of the workspace decomposition on the overall performance of INTERACT. These results are obtained by varying the average triangle area in the workspace decomposition. The default triangulation, used in all other experiments, is `t3` where the average triangle area is 0.035% of the total workspace area. Referring to Fig. 11, when the decomposition is too fine-grained, the running time increases since it becomes computationally more expensive to compute $\text{path}(D, r, a)$ which is used in the estimation of $\text{cost}(\Gamma_{\langle q,r\rangle,a})$ (Eqn. 11). Another reason is that the partition of the motion-tree into groups $\Gamma_{\langle q,r\rangle}$ also becomes too fine-grained so it loses its effectiveness. At the other end, there is also a runtime increase when the partition is coarse-grained since fewer region groups are created, and, thus, it becomes difficult to find alternative routes along which to expand the motion tree. Nevertheless, the results show that INTERACT works well for a wide range of workspace decompositions. This makes it easier to find a suitable workspace decomposition when considering a new problem instance.

When comparing the solution lengths, as shown in Fig. 11(b), INTERACT tends to produce shorter solutions when using fine-grained workspace decompositions. In such cases, $\text{path}(D, r, a)$ provides a more accurate estimate $\text{cost}(\Gamma_{\langle q,r\rangle,a})$ (Eqn. 11).

5.2.5. *Impact of the Selection Penalty Factor*

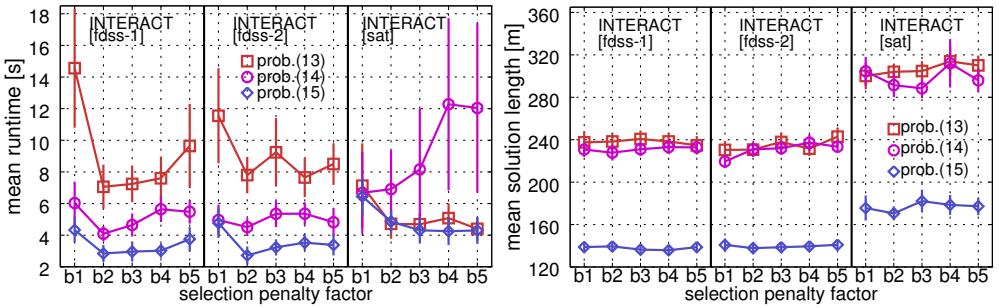


Figure 12. Runtime and solution-length results when varying the selection penalty factor, β (Section 4.1).

Fig. 12 shows the impact of the selection penalty factor, β, which is used by INTERACT to counterbalance the greediness of the selection process toward groups associated with

low-cost action plans (Section 4.1). As β approaches 1, the runtime increases since INTERACT becomes too greedy. The runtime also increases when β becomes large since such values reduce the impact of the PDDL planner. The results show, however, that INTERACT works well for a wide range of values. When starting with a new problem, we recommend using $\beta = 1.5$. This was the default value used in the experiments. When comparing the solution lengths, the variation is small since β does not affect the action-plan costs.

6. Discussion

This paper presented an effective approach to plan collision-free and dynamically-feasible motion trajectories that enable the robot to satisfy task specifications given in PDDL. The key aspect of INTERACT was the tight coupling of sampling-based motion planning with action planning. Indeed, action plans guided the motion-tree expansion, while the latter provided information to adjust the action costs based on the progress made. The interplay among the planning layers is essential to ensure that the search does not become stuck seeking to follow what could be an infeasible action plan. The partition of the motion tree into equivalence classes allowed INTERACT to selectively explore the vast high-dimensional search space, focusing first on the most promising leads. The workspace decomposition facilitated the tree expansion by providing sequences of regions along which to expand the motion tree in order to reach the locations associated with the current action being explored. Experimental results showed the efficiency of the approach. Applying INTERACT in a 3D environment would require a 3D decomposition of the workspace, which can be obtained by using an octree or some other subdivision grid.

A direction for future work is to enhance the action planner to leverage previous solutions to similar problems. In fact, during the course of one run, INTERACT invokes the action planner multiple times, using different initial states. It may be possible to use the previous searches to prune the current search. Another direction is to extend the approach to a team of robots working cooperatively to accomplish a common task. This would require the development of appropriate procedures to assign subtasks to robots while ensuring coordination and effective load balancing. Theoretical analysis can focus on showing probabilistic completeness. This can be achieved by showing that the proposed framework systematically searches the space of discrete actions and continuous motions. Starting from the initial state, the feasible actions will eventually be implemented by the sampling-based motion planning, and as a result the reachable discrete successor states will be eventually reached. The theoretical analysis could also shed light into algorithmic components that can be further improved.

INTERACT also opens up venues for a closer integration with knowledge representation and reasoning. As an example, in a search-and-rescue scenario, additional predicates can be used to classify people found by the robot based on their age, e.g., child, adult, elderly, and their physical and psychological condition. Rules can then be used to define appropriate robot behaviors based on this semantic information, such as aiming to increase safety by avoiding cluttered paths or moving newly found people along previously explored paths.

Funding

This work is supported by NSF IIS-1449505, NSF IIS-1548406, and NSF ACI-1440581.

Supplemental material

The supplemental material contains several videos showing solutions obtained by the proposed approach for various problem instances.

References

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1-2), 5–33.

Botea, A., Enzenberger, M., Müller, M., & Schaeffer, J. (2005). Macro-FF: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, *24*(1), 581–621.

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

Cambon, S., Alami, R., & Gravot, F. (2009). A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, *28*(1), 104–126.

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.

Choi, J., & Amir, E. (2009). Combining planning and motion planning. In *Ieee international conference on robotics and automation* (pp. 238–244).

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.

Coles, A., & Coles, A. (2011). LPRPG-P: Relaxed plan heuristics for planning with preferences. In *International conference on automated planning and scheduling* (Vol. 11, pp. 26–33). Freiburg, Germany.

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

de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. H. (2008). *Computational geometry: Algorithms and applications* (3rd ed.). Springer-Verlag.

de Silva, L., Pandey, A. K., & Alami, R. (2013). An interface for interleaved symbolic-geometric planning and backtracking. In *Ieee/rsj international conference on intelligent robots and systems* (pp. 232–239).

Devaurs, D., Simeon, T., & Cortés, J. (2013). Enhancing the transition-based rrt to deal with complex cost spaces. In *Ieee international conference on robotics and automation* (pp. 4120–4125).

Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., & Nebel, B. (2012). Semantic attachments for domain-independent planning systems. In *Towards service robots for everyday environments* (pp. 99–115). Springer.

Eppe, M., & Bhatt, M. (2013). Narrative based postdictive reasoning for cognitive robotics. In *International symposium on logical formalizations of commonsense reasoning*.

Erdem, E., Aker, E., & Patoglu, V. (2012). Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *Intelligent Service Robotics*, *5*(4), 275–291.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., & Uras, T. (2011). Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Ieee international conference on robotics and automation* (p. 4575-4581).

Fikes, R., & Nilsson, N. (1971). Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3-4), 189-208.

Gaschler, A., Petrick, R. P., Giuliani, M., Rickert, M., Knoll, A., et al. (2013). KVP: A knowledge of volumes approach to robot task planning. In *Ieee/rsj international conference on intelligent robots and systems* (pp. 202–208).

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, *26*(1), 191–246. Retrieved from `http://www.fast-downward.org/`

Hertle, A., Dornhege, C., Keller, T., & Nebel, B. (2012). Planning with semantic attachments: An object-oriented view. In *European conference on artificial intelligence* (Vol. 242, pp. 402–407).

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, *14*, 253–302.

Hsu, D., Kindel, R., Latombe, J. C., & Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, *21*(3), 233–255.

Kaelbling, L., & Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *Ieee international conference on robotics and automation* (pp. 1470–1477).

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.* New York, NY: Dover.

Ladd, A. M., & Kavraki, L. E. (2005). Motion planning in the presence of drift, underactuation and discrete system changes. In *Robotics: Science and systems* (p. 233-241). Boston, MA.

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., & Karlsson, L. (2014). Efficiently combining task and motion planning using geometric constraints. *International Journal of Robotics Research*.

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. (2014). Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions. In *Ieee/rsj international conference on intelligent robots and systems* (pp. 212–217).

Marthi, B., Russell, S., & Wolfe, J. (2007). Angelic semantics for high-level actions. In *International conference on automated planning and scheduling.* Providence, RI.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., ... Wilkins, D. (1998). *PDDL–the planning domain definition language* (Tech. Rep. No. CVC TR-98-003/DCS TR-1165). New Haven, CT: Yale Center for Computational Vision and Control.

McMahon, J., & Plaku, E. (2014). Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic. In *Ieee/rsj international conference on intelligent robots and systems* (pp. 3726–3733).

Nakhost, H., & Müller, M. (2012). A theoretical framework for studying random walk planning. In *Symposium on combinatorial search* (pp. 57–64). Niagara Falls, Canada.

Pednault, E. P. D. (1994). ADL and the state-transition model of action. *Journal of Logic Computation*, *4*(5), 467-512.

Plaku, E. (2013). Robot motion planning with dynamics as hybrid search. In *Aaai conference on artificial intelligence.* (1415–1421)

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

Plaku, E., & Hager, G. D. (2010). Sampling-based motion and symbolic action planning with geometric and differential constraints. In *IEEE International Conference on Robotics and Automation* (pp. 5002–5008).

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.

Richter, S., & Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, *39*(1), 127–177.

Rintanen, J. (2012). Planning as satisfiability: Heuristics. *Artificial Intelligence*, *193*, 45–86.

Shewchuk, J. R. (2002). Delaunay refinement algorithms for triangular mesh generation. *Com-*

putational Geometry: Theory and Applications, *22*(1-3), 21–74.

Sievers, S., Ortlieb, M., & Helmert, M. (2012). Efficient implementation of pattern database heuristics for classical planning. In *Symposium on combinatorial search* (pp. 105–111). Niagara Falls, Canada.

Wells, A., & Plaku, E. (2015). Adaptive sampling-based motion planning for mobile robots with differential constraints. In *Lncs towards autonomous robotic systems* (pp. 283–295).

Wolfe, J., Marthi, B., & Russell, S. (2010). Combined task and motion planning for mobile manipulation. In *International conference on automated planning and scheduling* (pp. 254–258). Toronto, Canada.