Article

The International Journal of Robotics Research
2018, Vol. 37(1) 104–136

DOI: 10.1177/0278364917739114
journals.sagepub.com/home/ijr

FFRob: Leveraging symbolic planning for efficient task and motion planning

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling

Abstract

Mobile manipulation problems involving many objects are challenging to solve due to the high dimensionality and multi-modality of their hybrid configuration spaces. Planners that perform a purely geometric search are prohibitively slow for solving these problems because they are unable to factor the configuration space. Symbolic task planners can efficiently construct plans involving many variables but cannot represent the geometric and kinematic constraints required in manipulation. We present the FFRob *algorithm for solving task and motion planning problems. First, we introduce* extended action specification *(EAS) as a general purpose planning representation that supports arbitrary predicates as conditions. We adapt existing heuristic search ideas for solving* STRIPS *planning problems, particularly delete-relaxations, to solve EAS problem instances. We then apply the EAS representation and planners to manipulation problems resulting in* FFRob. FFRob *iteratively discretizes task and motion planning problems using batch sampling of manipulation primitives and a multi-query roadmap structure that can be conditionalized to evaluate reachability under different placements of movable objects. This structure enables the EAS planner to efficiently compute heuristics that incorporate geometric and kinematic planning constraints to give a tight estimate of the distance to the goal. Additionally, we show* FFRob *is probabilistically complete and has a finite expected runtime. Finally, we empirically demonstrate* FFRob*'s effectiveness on complex and diverse task and motion planning tasks including rearrangement planning and navigation among movable objects.*

Keywords

Task and motion planning, manipulation planning, AI reasoning

1. Introduction

A long-standing goal in robotics is to develop robots that can operate autonomously in unstructured human environments. Recent hardware innovations have made mobile manipulator robots increasingly affordable, and sensing innovations provide unprecedented sensory bandwidth and accuracy. Progress in algorithms for navigation and motion planning has enabled some basic forms of mobile manipulation, which combine actuation of a robot's base and end-effectors to move objects in the world. However, mobile manipulation is primarily restricted to picking and placing objects on relatively uncluttered surfaces. Planning for mobile manipulation problems involving cluttered environments and multiple manipulation primitives still presents substantial challenges.

Researchers in artificial intelligence planning (Ghallab et al., 2004) have been tackling problems that require long sequences of actions and large discrete state-spaces. However, these symbolic *task-level* planners do not naturally encompass the detailed geometric and kinematic considerations that robot motion planning requires. The original Shakey and STRIPS robot system (Fikes and Nilsson, 1971; Nilsson, 1984), from which many of these symbolic planners evolved, managed to plan for an actual robot by working in a domain where all legal symbolic plans were effectively executable. This required the ability to represent symbolically a sufficient set of conditions to guarantee the success of the steps in the plan. Compactly encoding success conditions using typical symbolic representations is not generally possible in realistic manipulation domains because the geometrical and kinematic constraints are significant.

Consider a simple manipulation domain where a variety of objects are placed on a table and the robot's task is to collect some subset of the objects and pack them in a box. The basic robot operations are to pick up an object and place it somewhere else; in addition, the robot can move its base

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, USA

Corresponding author:
Caelan Reed Garrett, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA 02139, USA.
Email: caelan@csail.mit.edu

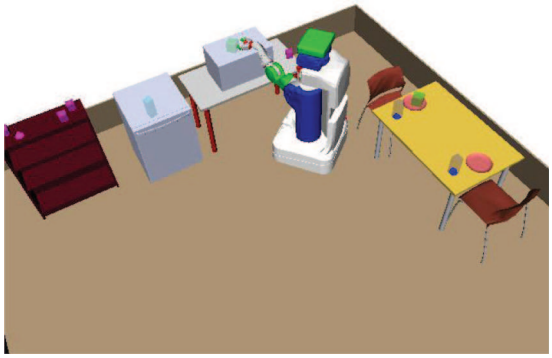

Fig. 1. A task and motion planning problem requiring cooking dinner. The robot must obtain two green cabbages from the shelves, clean them on the dishwasher, cook them on the microwave, and serve them. Additionally, the robot must organize the dirty cups on the table, clean them, and set the table.

in order to reach a distant object. Note that, in general, to pick a distant object or place an object at a distant location, the robot will have to move other objects out of the way. Which objects need moving depends on their shapes, the shape of the robot, where the robot's base is placed and what path it follows to the object. When an object is moved, the choice of where to place it requires similar considerations. The key observation is that constructing a valid symbolic plan requires access to a characterization of the connectivity of the underlying free configuration space (for the robot and all the movable objects). We cannot efficiently maintain a representation of this connectivity with a set of static assertions updated by symbolic actions; determining how the connectivity of the underlying free space changes requires geometric computation.

Whereas classic robot motion planning requires a search in the robot configuration space, manipulation planning requires a search in the combined configuration space of the robot and all the movable objects in the world. Achieving a manipulation goal requires choosing which object to move when, which grasps and intermediate placements to use, etc.

Manipulation planning remains challenging because it is notoriously difficult to work in a high-dimensional space and make a long sequence of intertwined decisions. Existing manipulation planning algorithms (Barry et al., 2013; Cambon et al., 2009; Dogar and Srinivasa, 2012; Hauser and Latombe, 2009; Hauser and Ng-Thow-Hing, 2011; Sim´eon et al., 2004) take substantial time to plan operations involving relatively few objects. Without any search guidance, these algorithms must explore a large fraction of the configuration space to find a plan. And the size of the configuration space grows exponentially with the number of moveable objects in the world. Constructing such plans generally requires methods for partitioning the problem and for effective search guidance. Therefore we seek to integrate the capabilities of a task planner and a manipulation planner to produce an efficient mobile manipulation planning algorithm.

1.1. Approach

The primary contribution of this paper is FFRob, an efficient and probabilistically complete algorithm for fully integrated task and motion planning. This paper is an extended and revised version of a conference paper by Garrett et al. (2014). We model task and motion planning as symbolic planning where the conditions of actions are complex predicates involving geometric and kinematic constraints. We adapt efficient existing heuristic search algorithms for solving traditional symbolic planning problems to solve task and motion planning problems. The key computational benefit of the approach is that it is able to incorporate geometric and kinematic constraints in the heuristic to strongly guide the search.

To start, we formally identify a subclass of task and motion planning problems, pick–place–move (PPM) problems, which will be our focus. We later show how FFRob can be easily extended to solve more general task and motion planning problems involving additional symbolic inferences or manipulation primitives.

We introduce *extended action specification* (EAS), a new symbolic planing representation that supports complex conditions. Although this representation is not specific to task and motion planning or even robotics problems, our primary application of it will be to PPM problems discretized using sampling. EAS is able to represent actions with complex conditions much more concisely than a traditional symbolic planning representation.

Additionally, EAS allows specification of predicate evaluation functions to efficiently test conditions. In the context of PPM problems, we give a method for quickly evaluating reachability predicates using dynamic programming and collision check caching.

Following this, we give our extension of relaxed planning heuristics, particularly the *FastForward* (FF) heuristic (Hoffmann and Nebel, 2001), to the EAS planning representation.

In order to frame task and motion planning as symbolic planning in a finite domain, we repeatedly discretize the planning problem. This involves batch sampling a set of placement poses and grasp transforms to identify the pick and place actions. Then, we construct a roadmap of robot configurations to give an approximation of the robot's free configuration space. This roadmap is instrumental in enabling efficient evaluation of reachability predicates that arise when the robot seeks to move to a new configuration.

We prove completeness results for FFRob by identifying a class of non-degenerate PPM problems and proving FFRob will solve them with finite expected runtime. Finally, we perform experiments on challenging manipulation problems and explore the effect of various planner configurations on their performance. We demonstrate that FFRob can solve a broad class of feasible task and motion planning problems that involve navigating among and rearranging moveable objects.

2. Related work

This work draws from existing approaches to manipulation planning and to task and motion planning well as ideas from the artificial intelligence symbolic planning literature. Our focus will be on showing how ideas originally developed for symbolic planning can be adapted to continuous-action domains to more efficiently solve high-dimensional task and motion planning problems.

2.1. Manipulation planning

In manipulation planning, the goal is not just to move the robot without collision, as in classical motion planning, but also to operate on the objects in the world. This problem was addressed from the earliest days of algorithmic motion planning, for example by Lozano-Pérez (1981), Lozano-Pérez et al. (1987), and Wilfong (1988). The modern treatment of this problem, involving continuous grasps as well as continuous placements, was pioneered by Alami et al. (1990, 1994) who introduced the *manipulation graph*. This graph breaks the problem of one robot moving one object in a potentially complex environment into several problems of moving between connected components of the combined configuration space where each component shares the same grasp. A solution is an alternating sequence of transit paths, in which the robot is not grasping an object, and transfer paths, in which it is. Siméon et al. (2004) expanded this work to more realistic settings by using probabilistic roadmaps. They looked at manipulations necessitating multiple regrasps. Their approach uses the manipulation graph to identify a high-level sequence of transit and transfer paths then performs the motion planning required to achieve them.

Stilman and Kuffner (2006) and Stilman et al. (2007) address a version of manipulation planning called *navigation among movable obstacles* (NAMO), where the robot must reach a specified location among a field of movable obstacles. In order to solve monotonic NAMO instances, instances requiring at most one pick and place for each object, they plan backwards from the goal and use swept volumes to determine, recursively, which additional objects must be moved. Van Den Berg et al. (2009) developed a probabilistically complete algorithm for NAMO. However, this algorithm assumes that one can fully characterize the connected components of the configuration space of the robot at each planning step; this is computationally prohibitive for robotic configuration spaces with more than two dimensions.

Hauser and Latombe (2009) and Hauser and Ng-Thow-Hing (2011) identified a generalization of manipulation planning as *hybrid planning*, that is, planning for systems with multiple (possibly infinitely many) *modes*, representing different constraint submanifolds of the configuration space. In a robotics domain, for example, modes are characterized by a grasp on a particular object and the placements of movable objects. The key insight is that, as in the manipulation graph, one can conceptualize the planning process as alternating between moving in a single mode, where the constraints are constant (e.g. moving through free space with a grasped object), and switching between modes (e.g. grasping a new object). So, solving these problems requires planning within a single mode and identifying configurations where modes can change, which is generally specific to the task. Hauser provided a probabilistically complete algorithm that solves problems of this type assuming that effective single-mode planners and mode-transition samplers are available. However, a pure unidirectional sampling-based algorithm has trouble solving high-dimensional problems.

Barry et al. (2013) defined a bidirectional *rapidly-exploring random tree* (RRT) search of the combined configuration space. Importantly, the individual moves of this algorithm consider complete plans to reach the goal, ignoring obstacles, ensuring that suggested motions have some chance of being on the path to the goal. They also investigated a two-layer hierarchy, where the higher level plans only for the manipulated object (without the robot), with the objective of identifying relevant mode transitions to guide the full planner. This planner was limited to domains with one movable object and had running times on the order of minutes.

Krontiris and Bekris (2015, 2016) provided an algorithm for rearrangement planning: a special instance of pick-and-place planning where all objects have explicit goal poses. Their method constructs a *probabilistic roadmap* (PRM) (Kavraki and Latombe, 1998) in the combined configuration space similar to that of Barry et al. (2013). It samples arrangements of the objects and uses a greedy local planner based on the NAMO algorithm of Stilman and Kuffner (2006) to connect an existing PRM vertex with the sampled object arrangement. The use of the PRM was able to recover completeness for problems that could not be solved by just the greedy planner. However, the lack of search guidance forces the planner to explore a large number of object arrangements.

Garrett et al. (2015) introduced the *hybrid backward-forward* (HBF) algorithm for hybrid planning problems. HBF uses a backward search to produce successors and distance estimates for states in a forward search. HBF was applied to manipulation problems involving robot primitives for picking, placing, and pushing. In contrast with FFRob's batch action sampling, HBF samples action primitives while simultaneously searching through the state-space.

King et al. (2016) investigated rearrangement planning with both *object-centric motions*, actions involving a particular object, and *robot-centric motions*, actions not involving any particular objects. Most of the presented literature involves only object-centric motions. Planning with robot-centric motions can enable complex manipulations such as multi-object pushing and whole arm manipulation. Using these primitives can result in much shorter and more natural

plans than using object-centric motions alone. Like Barry et al. (2013), they generalize the RRT algorithm to plan with both of these motions. Their algorithm also lacks the strong search guidance needed to effectively plan for problems with many objects and long horizons.

2.2. Symbolic planning

The artificial intelligence (AI) planning community has largely adopted heuristic state-space search methods for solving symbolic planning problems. These planning problems are, by and large, discrete problems that are represented using *planning domain definition language* (PDDL) as proposed by McDermott et al. (1998).

Bonet and Geffner (1999, 2001) popularized these state-space search methods by showing how *domain-independent* heuristics could be derived automatically, by manipulating the conditions and effects of the actions. The key idea is to define a relaxed version of the planning problem where the actions do not have any 'delete' effects, that is, no previously achieved result becomes false. This is an easier problem to solve than the original problem and the solution can be used to provide an approximation to the actual cost of solving a problem. They identified two heuristics, h_{add} and $h_{\max}$, that derive their estimates from a relaxed version of the *plan graph* (Blum and Furst, 1997). They each can be computed in polynomial time by taking the sum or max of the costs of achieving individual terms in a conjunctive goal respectively. The $h_{\max}$ heuristic is admissible, while h_{add} is inadmissible but more informed (it tends to be closer to the true cost in practice, providing more effective guidance).

The FF planning system of Hoffmann and Nebel (2001) introduced the h_{ff} heuristic, which explicitly extracts a plan from the relaxed plan graph and uses the plan's cost as its value. By avoiding double-counting actions that achieve several conditions, h_{ff} is generally a tighter estimate of the cost to reach the goal than h_{add} and $h_{\max}$. Importantly, the resulting relaxed plans can also be used to suggest useful actions to consider at the current state, which can reduce the branching factor of the search.

The AI planning community has also investigated planning in hybrid domains with simple continuous dynamics. Coles et al. (2013) gave a heuristic for numerical planning that combines a mixed integer program with a relaxed plan graph to create a hybrid heuristic that is able to more strongly guide the search by using fewer approximations. This adaptation of a relaxed plan graph, albeit in a different way and for a different problem, is similar in spirit to the inclusion of geometric inferences in FFRob's relaxed plan graph.

More generally, *planning modulo theories* (PMT) (Gregory et al., 2012) is a framework for using arbitrary first-order logical theories in planning problems. This formulation, inspired by *SAT modulo theories* (SMT), was designed to have wide expressivity and unify the representation for many existing planning types. Gregory et al. also gave a heuristic search algorithm for solving PMT problems. Its heuristic is an extension of $h_{\max}$.

The resulting planner is able to solve many problems that cannot be modeled with PDDL. It even outperforms some algorithms operating on PDDL problems because it plans using a more compact representation by allowing complex conditions. Our planning representation and algorithms are similar to PMT applied to problems with arbitrary propositional conditions. However, our framework allows for custom evaluation of complex conditions and supports additional heuristics that are more effective than $h_{\max}$.

2.3. Task and motion planning

There have been a number of approaches to integrating discrete task planning and continuous motion planning in recent years. The pioneering Asymov system (Cambon et al., 2009) conducts an interleaved search at the symbolic and geometric levels. They carefully consider the consequences of using non-terminating probabilistic algorithms for the geometric planning, allocating computation time among the multiple geometric planning problems that are generated by the symbolic planner. The process can be viewed as using the task planner as a heuristic to guide the motion planning search. However, since the task-level planner is ignoring geometry, its value as a heuristic is quite limited. The work of Plaku and Hager (2010) is similar in approach.

A natural extension to the classic symbolic planning paradigm is to introduce *computed predicates* (also known as *semantic attachments*); that is, predicates whose truth value is established not via assertion but by calling an external program that operates on a geometric representation of the state (Dornhege et al., 2009, 2013). A motion planner can serve to implement such a predicate, determining the reachability of one configuration from another.

A difficulty with this approach, however, is that calling a motion planner is generally expensive. This leads to a desire to minimize the set of object placements considered to limit the branching factor of the search. Considering only a sparse set of placements may limit the generality of the planner. Additionally, computed predicates are ignored during heuristic computation. This leads to a heuristic that is uninformed about geometric considerations and may result in considerable inefficiency due to heuristic plateaus. The work of Erdem et al. (2011), is similar in approach to Dornhege et al. (2009), augmenting a task planner that is based on explicit causal reasoning with the ability to check for the existence of paths for the robot.

Lagriffoul et al. (2012, 2014) interleave the symbolic and geometric searches and focus on limiting the amount of geometric backtracking. They generate a set of approximate linear constraints imposed by the program under consideration, e.g. from grasp and placement choices, and use linear programming to compute a valid assignment or

determine that one does not exist. This method is particularly successful in domains such as stacking objects in which constraints from many steps of the plan affect geometric choices. Although their approach is able to efficiently decide if a task-level plan is geometrically feasible, it is unable to inform the task-level search, which may result in attempting many infeasible plans.

Pandey et al. (2012) and de Silva et al. (2013) use Hierarchical Task Networks (HTNs) instead of generative task planning. Their system can backtrack over choices made by the geometric module, allowing more freedom to the geometric planning than in the approach of Dornhege et al. (2009). In addition, they use a cascaded approach to computing difficult applicability conditions: they first test quick-to-evaluate approximations of accessibility predicates, so that the planning is only attempted in situations in which it might plausibly succeed.

In the Hierarchical Planning in the Now (HPN) approach of Kaelbling and Lozano-Pérez (2011), a regression-based symbolic planner uses *generators*, which perform fast approximate motion planning, to select geometric parameters, such as configurations and paths, for the actions. Reasoning backward using regression allows the goal to significantly bias the actions that are considered.

Srivastava et al. (2014) offer a novel control structure that avoids computing expensive condition values in many cases by assuming a favorable default valuation of the condition elements; if those default valuations prove to be erroneous, then it is discovered in the process of performing geometric planning to instantiate the associated geometric action. In that case, symbolic planning is repeated after adding updated valuations. This approach requires the ability to diagnose why a motion plan is not possible in a given state, which can be challenging, in general.

Lozano-Pérez and Kaelbling (2014) leverage *constraint satisfaction problem* (CSP) solvers for task and motion planning. Their approach performs a discrete search in the space of plan skeletons and uses a CSP solver to determine if a valid set of action parameters completes the plan skeleton.

Dantam et al. (2016) extend this approach by more generally formulating task and motion planning as a *satisfiability modulo theories* (SMT) problem. They use an incremental constraint solver to add motion constraints to the task-level logical formula when a candidate task plan is found. Upon failure, they iteratively increase the plan depth and motion planning timeouts, which results in a probabilistically complete algorithm.

Toussaint (2015) formulates task and motion planning as a logic–geometric program, a non-linear constrained optimization problem augmented with a logic and knowledge base. He introduces three approximations that make solving the problem more tractable by sequentially optimizing the final state, transfer configurations, and motion trajectories. His experiments apply the technique to maximizing the height of a stable structure constructed from a set of objects.

All of these approaches, although they have varying degrees of integration of symbolic and geometric planning, generally lack a true integrated search that allows the geometric details to affect the focus of the symbolic planning. FFRob develops such an integrated search, provides methods for performing it efficiently, and shows that it results in significant computational savings.

3. Problem formulation

We start by modeling robotic planning domains that involve a single manipulator on a mobile base in an environment with moveable rigid objects. We focus on this specific domain because it is the subject of our experiments; however, the general formulation has broader applicability and can be extended to different domains involving, for instance, several manipulators or additional symbolic fluents. We call this class of problems PPM problems. We assume that the environment is fully observable and that actions have deterministic effects.

Definition 1. A *PPM domain* $\mathcal{D} = \langle \mathcal{Q}, \{(\mathcal{P}^{o_1}, \mathcal{G}^{o_1}) ..., (\mathcal{P}^{o_m}, \mathcal{G}^{o_m})\}\rangle$ is specified by a robot configuration space $\mathcal{Q}$ as well as a space of placement surfaces $\mathcal{P}^{o_i}$ and a space of grasps $\mathcal{G}^{o_i}$ for each of the m moveable objects o_i.

$\mathcal{P}^{o_i}$ is the union of poses where object o_i can legally be placed such as poses supported by tops of tables or floors. $\mathcal{G}^{o_i}$ contains a set of grasps, which may be discrete or continuously infinite depending on the geometry of the robot and o_i. We assume that $\mathcal{Q}$ and each $\mathcal{P}^{o_i}$ take into account collisions with any fixed obstacles or joint limits, so values in each space are collision-free when no moveable objects are in the environment. We will not consider stacking domains where $\mathcal{P}^{o_i}$ could contain surfaces on top of other movable objects. FFRob can be extended to solve stacking problems by sampling sets of object poses that form a structurally sound stack. This formulation encompasses pick-and-place planning, rearrangement planning, and NAMO. We will later show that additional symbolic values can be easily incorporated into the domain to plan for tasks like cooking meals.

In a PPM domain $\mathcal{D}$, we can represent the state of the system using a set of *variables* $\mathcal{V} = \{v_{\mathrm{r}}, v_{\mathrm{h}}, v_{o_1}, ..., v_{o_m}\}$. Each variable v_a has a domain of possible values D_a. A *state* $s = \{v_{\mathrm{r}} = q, v_{\mathrm{h}} = g, v_{o_1} = p^{o_1}, ..., v_{o_m} = p^{o_m}\}$ is an assignment of values to the variables. These variables along with their domains and values are as follows:

- v_{r} is the robot configuration variable. The robot configuration domain is just $D_{\mathrm{r}} = \mathcal{Q}$. Each configuration $q \in D_{\mathrm{r}}$ specifies the pose of the base as well as the joint angles of the manipulator;
- v_{h} is the robot holding variable. The robot holding domain is $D_{\mathrm{h}} = \mathcal{G}^{o_1} \cup ... \cup \mathcal{G}^{o_m} \cup \{\textbf{None}\}$. For $g \in D_{\mathrm{h}}$, $g =$ **None** indicates the robot is not holding anything. Otherwise, $g = (o, \gamma)$ indicates the robot is holding object

o with a grasp transform γ relating robot's end-effector pose and the object pose;

- v_{o_i} is the object o_i pose variable for object label $i \in (1, ..., m)$. The object pose domain is $D_{o_i} = \mathcal{P}^{o_i} \cup \{\textbf{None}\}$. For $p^{o_i} \in D_{o_i}$, $p^{o_i} = \textbf{None}$ indicates that object o_i is not placed. Otherwise, $p^{o_i} = (x, y, z, \omega)$ is a four-dimensional pose (we assume that the object is resting on a stable face on a static horizontal surface).

We assume that the world is quasi-static and the robot can only hold a single object. When the robot is holding an object o_i, the object pose can be determined using $p^{o_i} = q \times \text{TRANSFORM}(g)$ where $\text{TRANSFORM}(g) = \gamma$. As such, it is redundant to explicitly update the pose of o_i when it is in the hand, so we let $p^{o_i} = \textbf{None}$ for simplicity. A state is *legal* if there are no collisions among the robot, held object, and the placed objects.

Definition 2. A *PPM problem* $\Pi = \langle s_0, S_* \rangle$ in a PPM domain $\mathcal{D}$ is specified by an initial state s_0 and a set of goal states S_*.

The initial state $s_0 = \{v_\text{r} = q_0, v_\text{h} = g_0^{o_{h_0}}, v_{o_1} = p_0^{o_1}, ..., v_{o_m} = p_0^{o_m}\}$ must be a legal state of the system. For simplicity, we will assume that S_* can be represented as the conjunction of goal sets for individual variables rather than logical predicates. We make this restriction because our manipulation experiments only involve goals that can be expressed in this form, and introducing arbitrary goal predicates will complicate the theoretical analysis. Thus, $S_* = \{v_\text{r} \in Q_*, v_\text{h} \in G_*^{o_{\text{h}*}}, v_{o_1} \in P_*^{o_1}, ..., v_{o_m} \in P_*^{o_m}\}$ defines a set of legal states in the Cartesian product of $Q_* \times G_*^{o_{\text{h}*}} \times P_*^{o_1} \times ... \times P_*^{o_m}$ where $Q_* \subseteq D_\text{r}$, $G_*^{o_{\text{h}*}} \subseteq D_\text{h}$, and $P_*^{o_i} \subseteq D_{o_i}$ for $i \in [m]$. If the goal set is left unspecified for a variable, the goal set defaults to the full variable domain.

4. The FFR$_{\text{OB}}$ algorithm overview

At the highest level of abstraction, FFRob iteratively alternates between a sampling phase and a planning phase until it is able to find a solution. The sampling phase discretizes the PPM problem by creating symbolic actions from a finite sampled set of poses, grasps, and configurations. The planning phase performs a discrete search to decide whether a solution exists. If the discrete search fails to find a solution, the process repeats with a larger set of samples.

The pseudocode for FFRob is presented in Figure 2. FFRob's inputs are a PPM domain $\mathcal{D}$ and problem Π, and its output is a solution *plan*. The procedure begins by initializing a set of sampling parameters θ that govern the number of samples to produce using INITIAL-PARAMETERS. In the sampling phase, SAMPLE-DISCRETIZATION discretizes the PPM problem by sampling a specified number of configurations, poses, and grasps determined by θ. SAMPLE-DISCRETIZATION returns a symbolic planning representation of the goal C_* and actions A in the current discretization of the problem. In the planning phase, SEARCH performs a discrete search using C_* and A. FFRob immediately terminates if it finds a solution. Otherwise, θ is increased using INCREMENT-PARAMETERS, and this process repeats.

```
FFROB(D, Π):
1  θ = INITIAL-PARAMETERS( )
2  while True:
3      C*, A = SAMPLE-DISCRETIZATION(D, Π; θ)
4      plan = SEARCH(⟨s0, C*, A⟩; ...)
5      if plan ≠ None:
6          return plan
7      INCREMENT-PARAMETERS(θ)
```

Fig. 2. The FFRob algorithm.

The majority of this paper is dedicated to the implementation of the two key subroutines: SAMPLE-DISCRETIZATION and SEARCH. Section 8 discusses the discretization created by SAMPLE-DISCRETIZATION. Sections 5, 6, and 7 are concerned with efficient search algorithms that implement SEARCH.

5. Symbolic planning representation

We will encode robot actions that pick up and place objects as well as move the robot in the style of traditional AI planning action descriptions such as those shown in Figure 3. In these actions, q, p, and γ are continuous variables that range over robot configurations, object poses, and grasp transforms, respectively. Because there are infinitely many values of these variables and therefore infinitely many actions, we assume we have sampled a finite set of these values during a pre-processing phase, resulting in a finite set of actions.

5.1. Extended action specification

We model discretized PPM problems using a representation that extends *simple action specification* (SAS+) (Bäckström and Nebel 1995). SAS+ is expressively equivalent to STRIPS (Bäckström and Nebel 1995) without action parameters. The key difference is that it supports variables with discrete domains instead of only propositional domains. A generic state s in SAS+ is an assignment of values to a finite set of variables $\mathcal{V}$. For PPM problems, the SAS+ variables are the same as the variables described in Section 3. Thus, a discretized PPM system state is a legal SAS+ state. For more general task and motion planning problems, the state may have additional variables such as categorical variables, v_{d_i}, for each object, which represent the cleaned or cooked status of object o_i where $D_{d_i} = \{\textbf{None}, \textbf{Cleaned}, \textbf{Cooked}\}$.

SAS+ requires conditions and effects to be simple assignments of individual values to a subset of the variables.

Definition 3. A *simple condition* $c \equiv [v = x]$ is a restriction that a state have value x for variable v.

Definition 4. A *simple effect* $e \equiv v \leftarrow x$ is an assignment of a value x to variable v.

Definition 5. A *partial state* $C = \{c_1, ..., c_k\}$ is a set of conditions.

A partial state defines a set of states that satisfy its conditions. The *goal* of a planning problem is a partial state.

Definition 6. An *action* $a = \langle C, E \rangle$ is a pair where C is a set of simple conditions and E is a set of simple effects.

Even with finite domains for all the variables, there is a difficulty with determining when the robot can perform an action. In particular, MOVE actions have a REACHABLE condition, which is true if the robot can safely move from q to q'. To concisely model and effectively plan for PPM problems, we need a more expressive representation that allows us to evaluate, for example, whether there exists a path between two robot configurations that does not collide with placed objects. We extend SAS+ by allowing conditions to be logical formulas defined on the values of the variables. We call the resulting planning representation the EAS. This representation is also generic; however, we will focus on its application to PPM planning.

Definition 7. A *condition* $c \equiv f(v_{i_1}, ..., v_{i_k})$ is a restriction that a state has values for variables $v_{i_1}, ..., v_{i_k}$ that satisfy a predicate f.

Definition 8. A *predicate* f is a finite Boolean combination of simple conditions.

An example predicate is $f(v_{o_1}, v_{o_2}) \equiv [v_{o_1} = p_1] \vee [v_{o_2} = p_2]$, which is true when o_1 is currently at pose p_1 or o_2 is currently at pose p_2. Let $s(v)$ give the value of variable v in state s.

Definition 9. A condition c *holds* in a state if it evaluates to true given the values of the state's variables

$$\text{HOLDS}(c, s) \equiv f(s(v_{i_1}), ..., s(v_{i_k}))$$

To concisely represent conditions sharing a common template form, we use *parameterized* conditions, functions from a set of parameters to a condition. The following parameterized conditions are relevant in discretized PPM problems. We use $\forall$ and $\exists$ only to compactly denote conjunctions and disjunctions over elements of our finite domains.

The parameterized condition $\text{INREG}(o_i, R)$ has parameters composed of an object o_i and a region $R \subseteq \mathcal{P}^{o_i}$ in its pose space. $\text{INREG}(o_i, R)$ is true if $s(v_{o_i}) \in R$, i.e. the current placement of o_i is contained within R. However, to express $\text{INREG}(o_i, R)$ as a predicate, we evaluate it in the following way

$$\text{HOLDS}(\text{INREG}(o_i, R), s) \equiv \exists p \in D_{o_i} \cap R.\ [s(v_{o_i}) = p]$$

The parameterized condition $\text{REACHABLE}(q, q', (V, E))$ has parameters composed of an initial robot configuration q, a final robot configuration q', and a discretized roadmap of robot movements (V, E). $\text{REACHABLE}(q, q', (V, E))$ is true if there is a collision-free path in (V, E) between q and q', considering the positions of all fixed and movable objects as well as the object the robot might be holding and the grasp in which it is held

$$\begin{aligned}
&\text{HOLDS}(\text{REACHABLE}(q, q', (V, E)), s) \equiv \\
&\quad \exists (e_1, ..., e_k) \in \text{PATHS}(q, q'; (V, E)).\ \forall e \in (e_1, ..., e_k).\\
&\quad \Big([s(v_{\text{h}}) = \textbf{None}] \vee \neg\text{ROBOT-GRASP-C}(e.\tau, s(v_{\text{h}}))\Big) \wedge \\
&\quad \forall i \in [m].\ \Big([s(v_{o_i}) = \textbf{None}] \vee \\
&\quad \big(\neg\text{ROBOT-OBJ-C}(e.\tau, (o_i, s(v_{o_i}))) \wedge ([s(v_{\text{h}}) = \textbf{None}] \vee \\
&\quad \neg\text{GRASP-OBJ-C}(e.\tau, s(v_{\text{h}}), (o_i, s(v_{o_i})))\big)\Big)
\end{aligned}$$

Each path $(e_1, ..., e_k)$ on (V, E) is composed of edges e which will each have their own trajectory $e.\tau$ for moving between the incoming and outgoing vertices. Let the predicate ROBOT-GRASP-C be true if the robot collides with the object it is holding v_{h}, as it moves along configuration trajectory $e.\tau$. Similarly, let ROBOT-OBJ-C be true if o_i at pose v_{o_i} collides with the robot along $e.\tau$, and GRASP-OBJ-C be true if o_i at pose v_{o_i} collides with the grasped object v_{h} along $e.\tau$ We assume that roadmap (V, E) is free of self-collisions or collisions with fixed obstacles as checked during its discretization. Although REACHABLE is rather complicated, it is still a Boolean combination of simple conditions. In Section 5.3, we provide a way to avoid constructing this predicate by instead directly evaluating it using an external procedure.

Definition 10. An action a is *applicable* in a state s if all of a's conditions hold in s

$$\text{APPLICABLE}(a, s) \equiv \forall c \in a.C.\ \text{HOLDS}(c, s)$$

Definition 11. For s, a such that $\text{APPLICABLE}(a, s)$, a can be *applied* to s to produce a successor state

$$\text{APPLY}(a, s) = \begin{cases} v = x & (v \leftarrow x) \in a.E \\ v = s(v) & \text{otherwise} \end{cases}$$

It is often more compact to represent actions in parameterized form as *action schemas*. An action schema is an action with typed parameters, standing for the set of actions arising from all instantiations of the parameters over the appropriate type domains. The PICK and PLACE action schemas in Figure 3 have parameters composed of a pose p, object o_i, grasp γ, and robot configuration q. The MOVE action schema has parameters composed of two configurations q, q' and a roadmap (V, E).

Although we focus on PPM problems using these actions, we could easily define other action schemas to solve more general task and motion planning problems. For example, the CLEAN and COOK action schemas in Figure 4 are useful for modeling a cooking task. The constants $R^{o_i}_{\text{clean}} \subseteq \mathcal{P}^{o_i}$ and

PICK$(p, (o_i, \gamma), q)$:
pre: $[v_\mathrm{r} = q]$, $[v_\mathrm{h} = \textbf{None}]$, $[v_{o_i} = p]$
eff: $v_\mathrm{h} \leftarrow (o_i, \gamma)$, $v_{o_i} \leftarrow \textbf{None}$

PLACE$(p, (o_i, \gamma), q)$:
pre: $[v_\mathrm{r} = q]$, $[v_\mathrm{h} = (o_i, \gamma)]$, $[v_{o_i} = \textbf{None}]$
eff: $v_\mathrm{h} \leftarrow \textbf{None}$, $v_{o_i} \leftarrow p$

MOVE$(q, q', (V, E))$:
pre: $[v_\mathrm{r} = q]$, REACHABLE$(q, q', (V, E))$
eff: $v_\mathrm{r} \leftarrow q'$

Fig. 3. Pick, place, and move action schemas.

CLEAN(o_i):
pre: $[v_{d_i} = \textbf{None}]$, INREG$(o, R^{o_i}_{clean})$
eff: $[v_{d_i} \leftarrow \textbf{Cleaned}]$

COOK(o_i):
pre: $[v_{d_i} = \textbf{Cleaned}]$, INREG$(o, R^{o_i}_{cook})$
eff: $[v_{d_i} \leftarrow \textbf{Cooked}]$

Fig. 4. Additional clean and cook action schemas.

$R^{o_i}_{\text{cook}} \subseteq \mathcal{P}^{o_i}$ are sets of poses where o_i can be cleaned and cooked, respectively.

Definition 12. An *EAS planning problem* $\langle s_0, C_*, A\rangle$ is specified by an initial state s_0, goal partial state C_*, and a set of actions A.

Definition 13. A finite sequence of actions $(a_1, a_2, ..., a_n) \in A \times A \times ...$ is a *solution* to a planning problem if and only if the corresponding sequence of states $(s_0, s_1, ..., s_n)$ starting from s_0 and recursively constructed using $s_i = \text{APPLY}(a_i, s_{i-1})$ satisfies $\forall i \in [n]$, $\text{APPLICABLE}(a_i, s_{i-1})$ and $\forall c \in C_*$. $\text{HOLDS}(c, s_n)$.

5.2. Relaxed evaluation

In Section 7.1, it will be algorithmically advantageous to evaluate conditions in the context of *relaxed planning*. Relaxed planning is an approximation of standard symbolic planning that ignores delete effects (Bonet and Geffner 2001). Central to relaxed SAS+ planning is the notion of a relaxed state.

Definition 14. A *relaxed state* $s_+ = \{v_1 = X_1, ..., v_n = X_n\}$ is a generalized state in which each v_i can simultaneously take on all values in a set $X_i \subseteq D_i$ where $X_i \neq \emptyset$.

A relaxed state represents a set of states formed by all combinations of the relaxed state's values. Specifically, relaxed states can take on simultaneous values because an action in a relaxed planning problem never removes a value from the relaxed state. Thus, instead of replacing the values of variables, an action's effects add the new variable values to the relaxed state. Relaxed states are equivalent to discrete, abstracted states from PMT (Gregory et al. 2012). Every state is a relaxed state; however, the converse is not true.

A condition c *holds* in a relaxed state s_+ if there exists an assignment of values in X_i to each variable v_i such that the condition evaluates to true. Let $\mathrm{T}(c, s_+)$ be true if such an assignment exists; then we define

$$\text{HOLDS}_+(c, s_+) \equiv \mathrm{T}(c, s_+)$$

Similarly, let $\mathrm{F}(c, s_+)$ be true if there exists an assignment of values in X_i to each variable v_i such that c evaluates to false. Because the domain of each variable is finite, any condition can be expressed as a Boolean combination of atomic variable assignments $[v = x]$. This allows us to define $\mathrm{T}(c, s_+)$ by using recursion on its structure. Because it is possible for both $\mathrm{T}(c, s_+)$ and $\mathrm{F}(c, s_+)$ to hold simultaneously, we provide a recursive definition for $\mathrm{F}(c, s_+)$ as well. T and F are related in their respective recursion by negation

$$\mathrm{T}(c, s_+) = \begin{cases} \mathrm{T}(c_1, s_+) \wedge \mathrm{T}(c_2, s_+) & c \equiv c_1 \wedge c_2 \\ \mathrm{T}(c_1, s_+) \vee \mathrm{T}(c_2, s_+) & c \equiv c_1 \vee c_2 \\ \neg \mathrm{F}(c', s_+) & c \equiv \neg c' \\ x_i \in s_+(v_i) & c \equiv [v_i = x_i] \end{cases}$$

$$\mathrm{F}(c, s_+) = \begin{cases} \mathrm{F}(c_1, s_+) \vee \mathrm{F}(c_2, s_+) & c \equiv c_1 \wedge c_2 \\ \mathrm{F}(c_1, s_+) \wedge \mathrm{F}(c_2, s_+) & c \equiv c_1 \vee c_2 \\ \neg \mathrm{T}(c', s_+) & c \equiv \neg c' \\ s_+(v_i) \neq \{x_i\} & c \equiv [v_i = x_i] \end{cases}$$

The difference between the relaxed HOLDS_+ and standard HOLDS is that at the atomic level, the relaxed HOLDS_+ can choose between several values of each X_i to make c true while the standard HOLDS only has a single value of each x_i. As a consequence, when a relaxed state s_+ is also a standard state, $\text{HOLDS}_+(c, s) = \text{HOLDS}(c, s)$. This relaxed state condition evaluation can be seen as implementing the *satisfies* interface in PMT (Gregory et al., 2012) for arbitrary logical theories over discrete variables.

It will be useful for the planning heuristics in Section 7.1 to identify a variable assignment in the relaxed state that *achieves* the condition. When $\text{HOLDS}_+(c, s_+)$ is true, let $\text{ACHIEVERS}_+(c, s_+)$ be a similar recursive function that uses bookkeeping to identify a variable assignment that makes the condition true

$$\text{ACHIEVERS}_+(c, s_+) = \mathrm{TA}(c, s_+)$$

Let $\mathrm{TA}(c, s_+)$ return a set of variable values in s_+ that allow c to be true; and let $\mathrm{FA}(c, s_+)$ be the analogous set of

variable values that allow c to be false

$$\mathrm{TA}(c, s_+) = \begin{cases} \mathrm{TA}(c_1, s_+) \cup \mathrm{TA}(c_2, s_+) & c \equiv c_1 \wedge c_2 \\ \mathrm{ANY}(\mathrm{TA}(c_1, s_+), \mathrm{TA}(c_2, s_+)) & c \equiv c_1 \vee c_2 \\ \mathrm{FA}(c', s_+) & c \equiv \neg c' \\ \{v_i \leftarrow x_i\} & c \equiv [v_i = x_i] \end{cases}$$

$$\mathrm{FA}(c, s_+) = \begin{cases} \mathrm{ANY}(\mathrm{FA}(c_1, s_+), \mathrm{FA}(c_2, s_+)) & c \equiv c_1 \wedge c_2 \\ \mathrm{FA}(c_1, s_+) \cup \mathrm{FA}(c_2, s_+) & c \equiv c_1 \vee c_2 \\ \mathrm{TA}(c', s_+) & c \equiv \neg c' \\ \{v_i \leftarrow \mathrm{ANY}(s_+(v_i) \setminus \{x_i\})\} & c \equiv [v_i = x_i] \end{cases}$$

The procedure ANY(x) arbitrarily selects an element from set x. Although there may be many combinations that satisfy the predicate because of ANY, our algorithms just require that a single, arbitrary satisfying assignment be returned. However, the strength of these heuristics may vary depending on the assignment.

5.3. Condition tests

Some conditions are computationally expensive to evaluate naïvely using HOLDS. Consider the REACHABLE condition. Its truth value is affected by all the state variables, because the connected components of the configuration space change as the grasp and object poses change, thus affecting reachability. Holding an object changes the 'shape' of the robot and therefore what configurations it may move between. Even more significantly, the poses of all the placed objects define additional obstacles that the robot must not collide with. For a roadmap discretization of the configuration space (V, E), the REACHABLE condition involves quantification over the possibly exponential number of simple paths in the discretized configuration space between q and q'. Additionally, some conditions share a significant amount of logical structure. Consider the set of REACHABLE conditions that share the same start configuration q.

Thus, we allow conditions to optionally be evaluated by a *test*, a procedure TEST($\{c_1, ..., c_n\}, s$) which can simultaneously evaluate the predicates of conditions $\{c_1, ..., c_n\}$. For the REACHABLE condition, we specify a test that uses dynamic programming to test if a collision-free path exists given the current state. This test evaluates all REACHABLE conditions with the same start configuration q at once by considering paths from q. If no TEST procedure is specified, the planner defaults to the previously described methods for testing if the condition holds in standard and relaxed states.

In contrast with the semantic attachments strategy of Dornhege et al. (2009), we additionally require that tests support evaluating whether relaxed states satisfy the conditions. This additional requirement allows the planner to include conditions evaluated by tests in heuristics to strongly guide the search.

Once again, a test that is correct for the relaxed states is also correct for standard states, so it is sufficient to implement just $\mathrm{TEST}_+(\{c_1, ..., c_n\}; s)$. This procedure must also replace the function of $\mathrm{ACHIEVERS}_+(c, s_+)$. Thus, in order to evaluate several conditions at once, TEST returns a subset of the conditions that are true paired with their achievers. An example implementation of TEST_+ that simply uses the default HOLDS_+ and $\mathrm{ACHIEVERS}_+$ is the following

$$\mathrm{TEST}_+(\{c_1, ..., c_n\}; s_+) \equiv \{\langle c_i, \mathrm{ACHIEVERS}_+(s_+)\rangle \mid c_i \in \{c_1, ..., c_n\}.\ \mathrm{HOLDS}(c_i; s)\}$$

In the case of REACHABLE, the procedure TEST-REACHABLE uses dynamic programming as well as lazy collision checking to simultaneously evaluate all REACHABLE predicates from the same start configuration q. For each edge, a set of achievers that enable the edge to be traversed without collision is stored. Then, the full set of achievers for each end configuration q' is computed by tracing back a path and taking the union of the edge achievers on the path. As a result, $\mathrm{TEST\text{-}REACHABLE}_+$ is much more efficient than quantifying over the exponential number of paths in the roadmap (V, E) for each end configuration q'. In practice, our TEST-REACHABLE implementation additionally memorizes the last reachable subgraph in order to avoid repeating computation for sequential evaluations in the same relaxed planning problem.

6. Search algorithms

We now review existing search algorithms that can be directly applied to EAS planning problems with no modification. The generic heuristic search procedure SEARCH is given in Figure 5. The SEARCH procedure has as arguments an EAS planning problem $\langle s_0, C_*, A\rangle$, EXTRACT and PROCESS procedures, which alter the search control, and H heuristic and ACTIONS successor procedures that give a heuristic cost and generate the action successors, respectively.

Depending on the behavior of the EXTRACT and PROCESS procedures, SEARCH can implement many standard search control structures, including depth-first, breadth-first, uniform cost, A^*, greedy best-first, and hill-climbing searches. Critical to many of these strategies is a heuristic function, which maps a state s in the search to an estimate of the cost to reach a goal state from s. We will assume that each of our actions has unit cost; however, these procedures can be easily altered when costs can be any non-negative number. Appendix A describes the standard best-first and deferred best-first control structures used to implement EXTRACT and PROCESS.

7. Search heuristics

In this section, we illustrate implementations of the H heuristic and ACTIONS successor procedures of Figure 5

```
SEARCH( ⟨s_0, C_*, A⟩; EXTRACT, PROCESS; H, ACTIONS)
 1  Q = QUEUE( STATEN( s_0, 0, H( s_0) , None) )
 2  while not EMPTY(Q):
 3      n = EXTRACT( Q)
 4      if ∀c ∈ C_*. HOLDS( c, n.s):
 5          return RETRACE-PLAN(n)
 6      for a ∈ ACTIONS( n.s, A):
 7          if APPLICABLE( a, n.s):
 8              s' = APPLY( a, n.s)
 9              PROCESS( Q, s', n; H)
10  return None
```

Fig. 5. The primary search control procedure.

by adapting several existing domain-independent heuristics to EAS planning problems. Because they are domain-independent, these heuristics apply to any EAS planning problem, not just PPM problems. However, we explore the physical interpretation of these heuristics in the context of discretized PPM problems.

The simplest heuristic we consider is UNSATISFIED-GOALS, which counts the number of unsatisfied goal conditions

$$\text{UNSATISFIED-GOALS}(s) = |\{c \in C_* \mid \neg\text{HOLDS}(c, s)\}|$$

Although this can be computed almost instantly, it gives an exceptionally poor estimate of the cost to the goal because the problem may require many actions to satisfy a goal condition. The subsequent heuristics we will discuss are more involved and consequently are able to give a much better cost estimate.

7.1. Relaxed planning

Many modern domain-independent heuristics are based on solving approximate planning problems and using their solution cost as an estimate of the cost to the goal (Helmert, 2006; Hoffmann and Nebel, 2001). Although this requires more computation per search node than UNSATISFIED-GOALS, it pays off in the search because the improved estimates significantly reduce the number of states explored. One influential planning approximation is the delete-relaxation (Hoffmann and Nebel, 2001). As introduced in Section 5.2, in relaxed planning problems, actions add each effect value to the state without removing the previous value. This leads to relaxed states in which a variable can take on multiple values simultaneously.

A solution to a relaxed planning problem is a relaxed plan, an applicable sequence of actions that results in a relaxed state that satisfies the goal conditions. Relaxed planning problems can be solved in polynomial time, making this approximation attractive. However, the problem of producing a minimum length relaxed plan is NP-hard (Hoffmann and Nebel, 2001).

```
COMPUTE-COSTS( s, C_*, A; COMB) :
 1  C = C_* ∪ ⋃_{a∈A} a.C
 2  s_+ = {v : ∅ | v ∈ V}
 3  effs, conds, acts = { }, { }, { }
 4  Q = QUEUE( {EFFN( v ← s( v) , 0, None) | ∀v ∈ V})
 5  while not EMPTY( Q):
 6      n = POP-MIN( Q, n ↦ n.cost)
 7      if n.e ∈ effs: continue
 8      effs[n.e] = n
 9      s_+( n.e.v) = s_+( n.e.v) ∪ {n.e.x}
10      for c ∈ C if HOLDS_+( c, s_+):
11          conds[c] = CONDN( c, ACHIEVERS( c, s_+) )
12          C = C \ {c}
13      if C_* ⊆ conds:
14          acts[C_*] = GOALN( C_*, COMB( C_*; effs,
                                          conds) )
15          return effs, conds, acts
16      for a ∈ A if a.C ⊆ conds:
17          acts[a] = ACTN( a, COMB( a.C; effs, conds) )
18          A = A \ {a}
19          for e ∈ a.E if e ∉ effs:
20              PUSH( Q, EFFN( e, acts[a].cost + 1, a) )
21  return None
```

Fig. 6. Method for computing relaxed planning costs.

Despite this, several relaxed planning heuristics have been shown to give effective estimates of the distance to the goal. We will mention two of them and finally focus on the FF heuristic in particular. Each of these heuristics can be expressed using a common subroutine, COMPUTE-COSTS, which is shown in Figure 6.

The delete-relaxation allows relaxed planning to be understood as a search in the space of effects rather than the space of full states. This is similar to a search on a hyper-graph where vertices are effects and hyper-edges are actions. At heart, COMPUTE-COSTS is a version of Dijkstra's algorithm generalized for these hyper-graphs. But propagating shortest path costs in a hyper-graph differs from the traditional Dijkstra's algorithm because each hyper-edge may require reaching several vertices at a time. And there are several methods to combine the costs of reaching these vertices to produce a single cost for reaching the hyper-edge. Thus, COMPUTE-COSTS requires the meta-parameter COMB which specifies the method for combining these costs. Specifically, COMB(C; *eff*, *con*) combines the costs of satisfying a set of conditions C given the currently satisfiable effects and conditions in *effs* and *conds*. Therefore, costs are not defined as the length of the shortest plan to reach a vertex. However, they are still a measure of the relaxed plan difficulty of reaching an effect, condition, or action. The choice of COMB will tailor COMPUTE-COSTS for each heuristic.

Figure 6 gives the pseudocode for COMPUTE-COSTS. The inputs are a state s, a goal partial state C_*, and a set of action instances A. The outputs are the *effs*, *conds*, and *acts* maps which compose a search tree within the hyper-graph by recording cost and back pointer information. The maps are composed of effect nodes, condition nodes, and action nodes, respectively. Effect nodes EFFN store an effect comprised of a variable and value, the cost at which it was first produced, and the action that achieves them. Condition nodes CONDN store a condition and a set of effects that satisfy the condition. Action nodes ACTN store an action and the cost at which the action's conditions were first satisfied.

The COMPUTE-COSTS procedure starts by computing the set of unachieved conditions C, initializing the relaxed state s_+, and initializing *effs*, *conds*, and *acts*. It maintains a priority queue Q of effects starting with each effect $v \leftarrow s(v)$ in the current state s. On each iteration, it pops an EFFN node n from the priority queue based on its cost $n.cost$ and adds n to *effs* and the effect $n.e = (v \leftarrow x)$ to s_+ if n has not already been reached. Each unachieved condition $c \in C'$ is then tested to see if it can now be satisfied by the addition of $n.e$. Although not displayed in the pseudocode, this is where TEST procedures, which compute the truth value of several conditions at once, are also evaluated. If a condition is not satisfied, a set of effects that satisfy c is returned by ACHIEVERS and recorded. If each goal condition is contained in *conds*, the search terminates because each goal is reachable. Here, the goal is recorded as a goal node GOALN, which is an ACTN with no effects. A heuristic value can then be obtained using $acts[C_*].cost$. If the goal is not reached, we process each newly reachable and unused action a by computing the cost of achieving a's conditions using COMB. Finally, we independently push each of a's unprocessed effects to the queue along with a back pointer to a.

This differs from existing methods for relaxed planning in SAS+ problems because conditions are allowed to be complex Boolean formulas in EAS. There may be many assignments of variables, and therefore combinations of effects that satisfy a condition. Thus, COMPUTE-COSTS explicitly searches over all the unsatisfied conditions upon reaching a new effect to determine whether any condition is now achievable. If so, it records a satisfying assignment using ACHIEVERS as described in Section 5.2.

We now describe the intuition behind COMPUTE-COSTS in discretized PPM problems. As COMPUTE-COSTS progresses, the application of each PICK, PLACE, and MOVE action can be thought of as creating a copy of the manipulated object or robot at the new grasp, pose, or configuration, respectively. This is because the manipulated object and robot can be at many poses, grasps, and configurations simultaneously in a relaxed state. Moreover, these copies do not interfere with each other; i.e. the robot can select which of the currently available values of its configuration, its grasp, and the object poses will allow for it to perform an action. Actions that have the same cost can be viewed as being performable in parallel. Thus, the robot simultaneously tries all actions that can be performed after a relaxed plan of length 0, then a relaxed plan of length 1, and so on. In terms of reachability, it removes geometric constraints by removing an object from the universe when it is first picked up and never putting it back, and by assuming the hand remains empty (if it was not already) after the first PLACE action. Thus, the set of satisfied REACHABLE$(q, q', (V, E))$ conditions becomes increasingly larger as the procedure progresses. Figure 7 shows the progression of objects that have greater than one pose in the relaxed state. As more objects can be picked and essentially removed from consideration, more and more reachable conditions become true.

Finally, although COMPUTE-COSTS runs in polynomial time in the size of the EAS, collision checks typically dominate the runtime in PPM problems. Even if a heuristic significantly reduces the size of the main search, it might result in a net increase in computation time if it is itself too slow to compute. Because we cache the results of collision checks, in practice, executing COMPUTE-COSTS is quite fast and the heuristic functions it enables substantially reduce the number of states explored and, indeed, the total computation time.

7.2. The HSP heuristics

The first two heuristics we can obtain are h_{add} and $h_{\max}$, which are computed by using ADD-COMB and MAX-COMB for COMB, respectively. As a reminder, C is a partial state and *eff*, *con* are maps of effect and condition nodes. The intention of COMB is to score the difficultly of achieving C using the costs already provided in *eff* and *con*. The h_{add} heuristic (Bonet and Geffner, 2001) returns the sum of the costs for the effects that satisfy each condition

$$\text{ADD-COMB}(C; \mathit{eff}, \mathit{con}) = \sum_{c \in C} \sum_{e \in \mathit{con}[c].E} \mathit{eff}[e].\mathit{cost}$$

This heuristic is optimistic, in the sense that if the delete effects were taken into account, it might take more steps to achieve each individual goal from the starting state; it is also pessimistic, in the sense that the actions necessary to achieve multiple goal conditions might be *shared*.

An admissible heuristic, $h_{\max}$ (Bonet and Geffner, 2001), is obtained by taking the maximum of the costs of the goal literals, rather than the sum. But $h_{\max}$ is found in practice to offer weaker guidance

$$\text{MAX-COMB}(C; \mathit{eff}, \mathit{con}) = \max_{c \in C} \max_{e \in \mathit{con}[c].E} \mathit{eff}[e].\mathit{cost}$$

7.3. The FF heuristic

The FF heuristic h_{ff} (Hoffmann and Nebel, 2001) derives its heuristic cost from the length of a relaxed plan. The relaxed plan is computed by first calculating either $h_{\max}$ or h_{add} using COMPUTE-COSTS to produce an ordering of

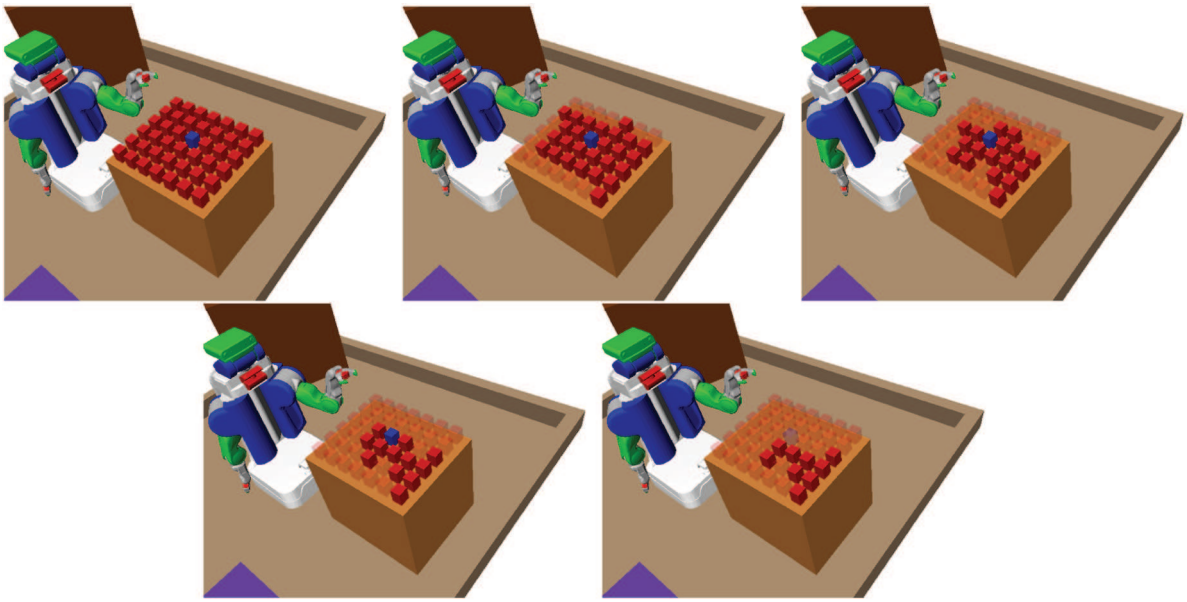

Fig. 7. Visualization of compute-costs for an PPM problem requiring picking up the blue block. Each object o is made transparent at the level when **None** $\in s_+(o)$. RPG levels 0, 1, and 2 are in the top row. RPG levels 3 and 4 are in the bottom row.

effects and actions as represented by *effs*, *conds*, *acts*. Then, EXTRACT-RELAXED-PLAN performs a backwards pass to identify a relaxed plan from *effs*, *conds*, *acts*.

In the event where $h_{\max}$ is used to produce the ordering, *effs*, *conds*, *acts* represent a *relaxed plan graph* (RPG). An RPG is a sequence of alternating *layers* of effects and actions. The first layer consists of all effects that are true in the starting state. Action layer i contains all actions whose conditions are present and simultaneously achievable in effect layer i. Effect layer $i + 1$ contains all effects that are possibly achievable after i actions. The depth of an effect or action in the RPG is equal to its $h_{\max}$ cost (assuming unit action costs). In the event where h_{add} is used to produce the ordering, the resulting structure is not semantically an RPG; however, the resulting structure functions akin to an RPG and can even lead to tighter heuristic estimates.

FF performs an efficient backward-chaining pass in the RPG to determine which actions, if they could be performed in parallel without delete effects, would be necessary to achieve the goal conditions. An important advantage of the FF heuristic is that it does not over-count actions if one action achieves multiple effects.

The EXTRACT-RELAXED-PLAN procedure in Figure 8 extracts a relaxed plan from the RPG.

The plan extraction procedure greedily processes each layer backwards from $i = (n, ..., 1)$, starting with the set of effects that achieve the goal conditions C_*. For each effect $e \in goals[i]$ identified as a goal on the ith layer of the RPG, it seeks the *cheapest* action a^* that can achieve it using EASIEST. The minimizing a^* is added to the relaxed plan *plan*, e' and any other effects achieved by a on the current layer are removed from *goals*, and the conditions $a^*.C$ are processed and their satisfying effects are added to *goals*. This process continues until each layer is processed. Once finished, the FF heuristic returns the number of actions in *plan*.

Our formulation of EXTRACT-RELAXED-PLAN is very similar to the original FF EXTRACT-RELAXED-PLAN algorithm. The modification to support EAS planning problems is simply performed, given that the effect achievers have been computed for each condition, by replacing conditions with the effects that satisfy them. Our metric for choosing the cheapest action is different from the original formulation, though. The original easiest-action metric uses the h_{add} cost of each action which is separately computed before EXTRACT-RELAXED-PLAN. Our EASIEST procedure uses the original cost and additionally discounts the cost of goals that have already been identified by addition to *goals*. The intuition for this change is that actions that do not add many new goals are greedily good choices.

As our results in Section 11 show, h_{ff} has the best performance in our PPM experiments. Our intuition behind why h_{ff} performs well for PPM problems comes from its backwards pass. There are often several ways to achieve a PPM goal. For example, consider the set of grasps and approach trajectories to pick up an object. Many of these approaches involve paths that require moving a different set of objects. And many of these paths may be performable on the same layer of the RPG. This is typically the case when an object is surrounded by an approximately even distribution of objects on several of its sides. The backwards pass, particularly through its greedy discounting, will frequently select the approach that will require moving the fewest of additional objects given the choices it has already made. Thus, the resulting relaxed plan usually reuses goals and gives a better estimate of the optimal cost to the goal.

$$\text{EASIEST}(C, goals; eff, con) = \sum_{c \in C} \sum_{\substack{e \in con[c].E \\ e \notin goals}} eff[e].cost$$

EXTRACT-RELAXED-PLAN($s, C_*, effs, conds, acts$):

```
1  n = acts[⟨C_*, ∅⟩].cost
2  goals = {i : ∅ | ∀i ∈ [n]}
3  for c ∈ C_* for e ∈ conds[c].E:
4      goals[effs[e].cost] ∪= {e}
5  plan = ∅
6  for i ∈(n, ..., 1):
7      for e ∈ goals[i] :
8          A_e = {a ∈ acts | e ∈ a.E
                   acts[a].cost = effs[e].cost − 1}
9          a = argmin_{a∈A_e} EASIEST(a.C, goals; eff, conds)
10         plan ∪= {a*}
11         for c ∈ a*.C for e' ∈ conds[c].E:
12             goals[effs[e'].cost] ∪= {e'}
13         for e' ∈ a*.E:
14             goals[effs[e'].cost] \= {e'}
15 return plan
```

Fig. 8. Method for extracting a relaxed plan.

7.4. Helpful actions

We now describe the implementation of the ACTIONS procedure used by the heuristic search algorithm in Figure 5. The simplest implementation is just to return the full set of actions A. However, we allow the optional specification of ACTIONS to be a function that can more efficiently return the set of applicable actions. For PPM problems, we compute the set of reachable configurations all at the same time using a procedure similar to TEST-REACHABLE in order to determine the applicable move actions.

Additionally, because h_{ff} computes a relaxed plan, we can use the resulting plans to prune and order a set of *helpful actions* for a given state.

Helpful action pruning strategies reduce the choice of the next action to those that were identified to be achieve goals on the relaxed plan via the heuristic computation. Additionally, they can order the application of these actions such that actions deemed more helpful are attempted first. Both of these methods can reduce the effective branching factor and the number of future heuristic computations needed in the search. But these pruning methods sacrifice completeness at the expense of strongly improved search performance. Completeness can be recovered though by using the multiple priority queues as introduced by Helmert (2006). We use a version of the helpful actions strategy that returns the *first actions* followed by the *first goal achievers*. The first actions are actions on a relaxed plan that are immediately performable. The first goal achievers are actions that have an effect on the first layer of a relaxed plan. Thus, the first actions are a subset of the first goal achievers.

SAMPLE-DISCRETIZATION($\mathcal{D}, \Pi; \theta$):

```
1  C_* = CONVERT-GOAL(S_*)
2  A_pick, A_place= S-PICK-PLACE(D, Π; θ)
3  A_move = S-MOVE(Q, q_0, Q_*, A_pick, A_place; θ)
4  A = A_pick ∪ A_place ∪ A_move
5  return C_*, A
```

Fig. 9. Procedure for sampling a discretized PPM problem and producing an EAS problem specification.

8. Discretization

In order to use the EAS representation and planners, we still have to discretize a PPM problem in a pre-processing phase by sampling a finite number of actions from each action schema.

Definition 15. A *discretized PPM domain* $D = \langle (V, E), \{(P^{o_1}, G^{o_1}) ..., (P^{o_m}, G^{o_m})\}\rangle$ is specified by a robot configuration roadmap ($V \subseteq \mathcal{Q}, E \subseteq V \times V$) as well as finite sets of placements $P^{o_i} \subseteq \mathcal{P}^{o_i}$ and grasps $G^{o_1} \subseteq \mathcal{G}^{o_i}$ for each of the m moveable objects o_i.

In Figure 9, the procedure SAMPLE-DISCRETIZATION both samples a discretized PPM problem and produces an EAS specification of the goal and actions for the discretized problem. First, the goal set of states S_* is converted into a set of predicates C_* using CONVERT-GOAL. Sets of PICK and PLACE actions are sampled using S-PICK-PLACE according to the current parameter vector θ. A set of MOVE actions is constructed using S-MOVE by creating a roadmap that connects the start configuration q_0, sampled goal configurations, Q_*, and PICK and PLACE configurations. Finally, SAMPLE-DISCRETIZATION returns C_* as well as the combined set of actions A.

The rest of this section gives the details of S-PICK-PLACE and S-MOVE as well as the procedure to evaluate complex REACHABLE predicates. We start by sampling the PICK and PLACE action schemas. Recall that PICK and PLACE have a condition that the robot be at a specific configuration in order to execute the pick or place. These configurations will serve as target configurations when sampling the roadmap (V, E). MOVE actions are created from pairs of these target configurations.

Sampling PICK and PLACE actions has a deep connection to sampling the modes of the system (Hauser and Latombe 2009; Hauser and Ng-Thow-Hing 2011). Specifically, the collection of poses and grasps will define the set of transit and transfer modes reachable from s_0. The MOVE actions then represent discretized motion plans within a mode. We will later revisit this idea in the theoretical analysis of FFRob in Section 10.

The number of samples chosen for each sampling procedure depends on a parameter vector θ. As previously said, θ will be iteratively increased in the event that not enough samples were chosen to find a solution. Additionally, to test whether this sampled set of poses, grasps, and configurations could possibly contain a plan, we can compute the heuristic value of the starting state s_0 using the actions derived from the samples, as described in Section 7.3. If it is infinite, meaning that the goal is unreachable even under extremely optimistic assumptions, then we return to this procedure and draw a new set of samples. Note that although a finite heuristic value is necessary for a plan to exist, it is not sufficient. Thus, the search algorithm may report that the set of samples is not sufficient to solve the problem although s_0 had a finite heuristic cost. However, in our experiments, this finite heuristic test can save a significant amount of time because computing a heuristic cost is much less expensive than solving the EAS planning problem.

Finally, evaluating REACHABLE still requires performing many expensive collision checks in the context of many different robot grasps and poses of the objects. We address this problem by using a shared *roadmap* data structure called a *conditional reachability graph* (CRG). The CRG is graph (V, E), which is related to a PRM (Kavraki et al. 1996) that answers reachability queries, conditioned on the poses of objects and the robot's grasp, by lazily computing answers on demand and caching results to speed up future queries.

8.1. Pick-and-place actions

Figure 10 gives the procedure for sampling action instances from the PICK and PLACE action schemas. This procedure, for each object, first produces poses and grasps useful for the problem, by using initial poses and grasps and sampling goal poses and grasps (if available) as well as uniformly sampling each space. For our implementation of S-POSES and S-GRASPS, we sample values using uniform rejection sampling (rejecting, for example, poses that collide with fixed obstacles); however, this can be done in other ways such as by choosing evenly spaced samples. Figure 11 shows sampled placements in red, for the blue and green blocks. Additional placements are sampled for the blue and green goal regions.

Once the poses and grasps (p and γ) are sampled, S-PICK-PLACE uses S-IK to sample valid configurations of the robot base and manipulator that reach the end-effector pose of $p \times \gamma^{-1}$. We implement S-IK by first sampling base poses from a pre-computed reachability database that are nearby the end-effector pose as shown in Figure 12. Then, we sample collision-free, analytical *inverse kinematics* (IK) solutions using ikfast (Diankov, 2010). Finally, each pose, grasp, and configuration tuple are used as the arguments of a PICK and PLACE action instance. These actions are added to the respective sets of EAS actions, A_{pick} and A_{place}.

S-PICK-PLACE$(\mathcal{D}, \Pi; \theta)$:

1 $A_{\text{pick}}, A_{\text{place}} = \emptyset, \emptyset$
2 **for** $i \in [m]$:
3 $\quad P^{o_i} = \{p_0^{o_i}\} \cup$ S-POSES$(P_*^{o_i}; \theta) \cup$ S-POSES$(\mathcal{P}^{o_i}; \theta)$
4 $\quad G^{o_i} =$ S-GRASPS$(\mathcal{G}^{o_i}; \theta)$
5 $\quad$ **if** $i = h_0 : G_o \cup= \{g_0^{o_{h_0}}\}$
6 $\quad$ **if** $i = h_* : G_o \cup=$ S-GRASPS$(G_*^{o_{h_*}}; \theta)$
7 $\quad$ **for** $(p, g) \in P^{o_i} \times G^{o_i}$:
8 $\quad\quad \gamma =$ TRANSFORM(g)
9 $\quad\quad$ **for** $q \in$ S-IK$(\mathcal{Q}, p \times \gamma^{-1}; \theta)$:
10 $\quad\quad\quad A_{\text{pick}} \cup= \{\text{PICK}(p, (o_i, \gamma), q)\}$
11 $\quad\quad\quad A_{\text{place}} \cup= \{\text{PLACE}(p, (o_i, \gamma), q)\}$
12 **return** $A_{\text{pick}}, A_{\text{place}}$

Fig. 10. Procedure for sampling the PICK and PLACE actions.

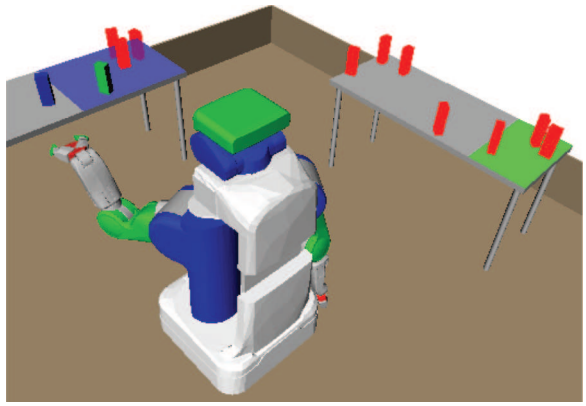

Fig. 11. Sampled placements for the blue and green blocks.

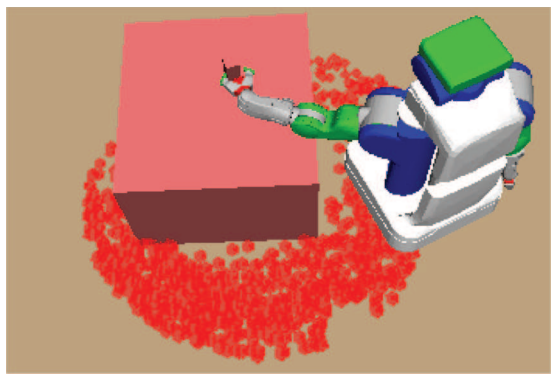

Fig. 12. The inverse reachability database transformed relative to the grasp transform.

8.2. Conditional reachability graph

Using the PICK and PLACE robot configurations as targets, we now sample robot movements and trajectories that will reach these pick and place configurations. In the process, we will create a CRG. The CRG is a partial representation of the connectivity of the space of sampled configurations, conditioned on the placements of movable objects as well

as on what is in the robot's hand. It efficiently allows us to evaluate REACHABLE$(q1, q2, (V, E))$ predicates. It is similar in spirit to the roadmaps of Leven and Hutchinson (2002) in that it is designed to support solving multiple motion planning queries in closely related environments. Formally, it is a graph (V, E) where the *vertices* V are a set of robot configurations, $q \in V$. The *edges* E are triplets $e = \langle q, q', \tau \rangle$ where q, q' are pairs of vertices and τ is a trajectory that connects q and q'. Each edge is also annotated with an initially empty map of *validation* conditions of the form $e.valid[\langle \rho, g \rangle] = b$ where $b =$ **True** if the edge is traversable for a placement of object ρ and grasp g when there are no other objects in the world. Otherwise, $b =$ **False**. There are three cases for a $\langle \rho, g \rangle$ pair:

- $\langle (o, p), \textbf{None} \rangle$: safe for the robot to traverse e when object o is placed at pose p and the robot is not holding any object;
- $\langle \textbf{None}, (o', \gamma) \rangle$: safe for the robot to traverse e when no object is placed and the robot is holding object o' with grasp transform γ;
- $\langle (o, p), (o', \gamma) \rangle$: safe for the robot to traverse e when object o is placed at pose p and the robot is holding object o' with grasp transform γ.

The validation conditions on the edges are not precomputed; they will be computed lazily, on demand, and cached in this data structure. These conditions are separated in this way in order to maximize the amount of collision caching used. Note that the procedure for determining $e.valid[\langle (o, p), (o', \gamma) \rangle]$ does not need to compute whether the robot collides with either o or o' because those conditions will already have been computed and stored in $\langle (o, p), \textbf{None} \rangle$ and $\langle \textbf{None}, (o', \gamma) \rangle$, respectively. However, it will still need need to compute whether (o, p) collides with (o', γ). Figure 13 depicts a cartoon CRG. The bottom figure is conditioned on a moveable object, which temporarily removes three edges from the traversable roadmap.

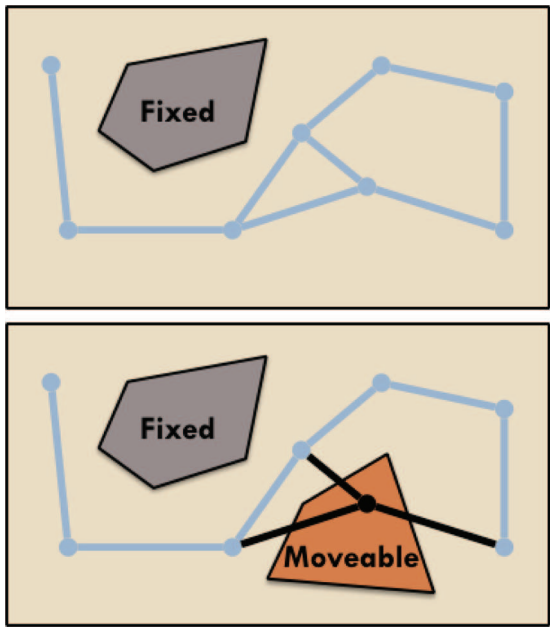

Fig. 13. An unconditioned CRG and the same CRG conditioned on the placement of a single moveable object.

8.3. *Constructing the* CRG

The CRG is initialized in the pre-processing phase when sampling the MOVE actions. The S-MOVE procedure is outlined in Figure 14. The initial configuration and goal configurations sampled using S-CONFIGS are used to initialize the CRG. It then calls the S-APPROACHES procedure, which generates trajectories for moving near each PICK and PLACE configuration.

Let PARAMETERS(a) give the tuple of parameters for action instance a. For each PICK and PLACE configuration, S-APPROACHES samples at a nearby configuration q' using S-NEARBY-CONFIG for the purpose of approaching the PICK and PLACE configuration q. In our implementation, we do this by concatenating the previous base pose with a pre-determined manipulator configuration used when carrying objects. Then it calls S-APPR-TRAJ, which samples trajectories τ between q and q'. For our implementation, we call RRT-Connect (Kuffner and LaValle 2000) between q and q' in the configuration space of just the manipulator (because we use the same base pose). We disallow trajectories that either collide with fixed obstacles while holding o at grasp g or collide with o at pose p when moving with an empty hand in order to use the trajectory for both PICK and PLACE actions. Our implementation assumes the robot is holonomic, so trajectories are reversible. For non-holonomic robots, a separate trajectory from q' and q must be sampled. An edge following the trajectory and an edge following the reversed trajectory are then added to the CRG.

Next, S-MOVE calls the S-ROADMAP procedure, which attempts to connect the configurations in the CRG. The procedure described in Figure 14 is for a fixed-degree PRM. It samples additional roadmap configurations using S-CONFIGS to attempt to connect the roadmap. For each configuration in the roadmap, S-ROADMAP attempts to connect the configuration to its nearest neighbors in V given by NEAREST-NEIGHBORS. It uses S-TRAJ to linearly interpolate between configurations. The number of additional configurations to sample and the desired degree of the PRM are given by the parameter vector θ. Figure 15 shows a sampled CRG for a NAMO problem.

In cases where domain-dependent information is available, it may make sense to use a different S-MOVE procedure. For example, for problems where objects can only be placed on tables, it is wasteful to create a dense roadmap for moving between tables because placed objects cannot possibly affect the validity of edges at a certain distance away from the tables. In our experiments involving objects

S-APPROACHES($\mathcal{Q}, (V,E), A_{\text{pick}}, A_{\text{place}}; \theta$):
1 **for** $a \in (A_{\text{pick}} \cup A_{\text{place}})$:
2 $(p,g,q) =$ PARAMETERS(a)
3 **for** $q' \in$ S-NEARBY-CONFIG($\mathcal{Q}, q$; SCH(a), $p, g; \theta$)
4 $V \cup = \{q, q'\}$
5 **for** $\tau \in$ S-APPR-TRAJ($\mathcal{Q}, q, q'$; SCH(a), $p, g; \theta$)
6 $E \cup = \{(q, q', \tau), (q', q,$ REVERSE(τ))$\}$

S-ROADMAP($\mathcal{Q}, (V,E); \theta$):
1 $V \cup =$ S-CONFIGS($\mathcal{Q}; \theta$)
2 **for** $q \in V$:
3 **for** $q' \in$ NEAREST-NEIGHBORS($V, q; \theta$):
4 **for** $\tau \in$ S-TRAJ($\mathcal{Q}, q, q'; \theta$)
5 $E \cup = \{(q, q', \tau), (q', q,$ REVERSE(τ))$\}$

S-MOVE($\mathcal{Q}, q_0, \mathcal{Q}_*, A_{\text{pick}}, A_{\text{place}}; \theta$):
1 $(V,E) = (\{q_0\} \cup$ S-CONFIGS($\mathcal{Q}_*; \theta$), $\emptyset$)
2 S-APPROACHES($\mathcal{Q}, (V,E), A_{\text{pick}}, A_{\text{place}}; \theta$)
3 S-ROADMAP($\mathcal{Q}, (V,E); \theta$)
4 $A_{\text{move}} = \{$MOVE($q, q', (V,E)$) $|$ $(q,q') \in V \times V$, PATH($q, q'; (V,E)$) $\neq$ **None**$\}$
5 **return** A_{move}

Fig. 14. Procedures for constructing the CRG.

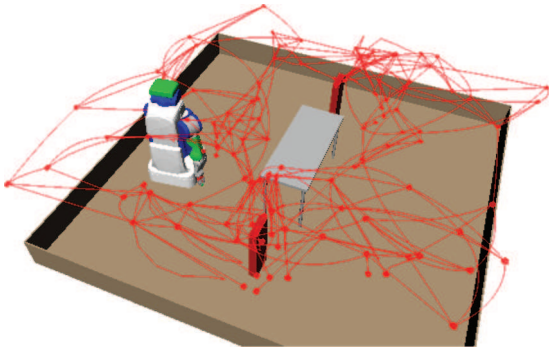

Fig. 15. Full CRG visualized using end-effector poses.

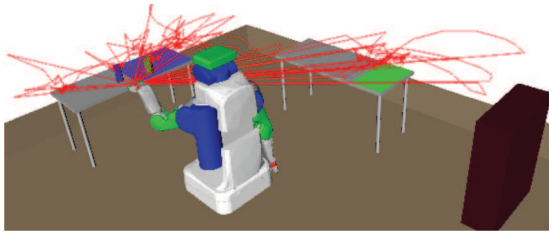

Fig. 16. A star-graph CRG visualized using end-effector poses.

that can only be placed on tables, we use a version of S-ROADMAP that is sparse away from tables, while still dense nearby tables by creating a star-graph of trajectories that connect an arbitrary reference configuration (such as q_0) to configurations near each table, as shown in Figure 16.

Finally, S-MOVE creates a MOVE action instance for pairs of configurations V that are connected in the roadmap as tested by a standard breadth-first search using PATH($q, q'; (V,E)$) (without considering any placed or held objects). In practice, we only create MOVE actions between start and goal configurations as well as configurations at which the robot can perform a PICK or PLACE because the robot will never need to stop at an intermediate configuration. Lastly, S-MOVE returns the set of move actions A_{move}.

8.4. *Querying the* CRG

Now that we have a CRG, we can use it to test whether REACHABLE conditions hold in relaxed state s_+, as shown in TEST-REACHABLE in Figure 17. Recall that in each relaxed state s_+, each object can simultaneously be: missing entirely, in multiple poses, and in multiple grasps. Similarly, the hand can hold several objects while also remaining empty. We need to determine whether there is some simultaneous assignment of all these variables using the values present in s_+, which allows a legal path from a start q to $q' \in V$. The test constructs a subgraph (V,E') of the CRG that consists only of the edges that are each independently valid for some choice of object poses and robot grasps from s_+. Additionally, each edge e is augmented with a temporary set of these pose and grasp achieving values $e.ach$ that collectively allow collision-free traversal of the edge. The test then searches that graph to see if configuration q' is reachable from q. It calls two subprocedures: TEST-GRASP and TEST-OBJ. Figure 18 displays the set of reachable CRG configurations from the current robot configuration given the placements of the movable objects for a NAMO problem.

The procedure TEST-GRASP checks, given an edge e, the relaxed state s_+, and a placement ρ (which may be **None**), whether the edge is valid for some grasp g in the relaxed state. If **None** is a grasp in s_+, the TEST-GRASP can immediately report success because it can navigate the edge without colliding with the fixed obstacles (assuming the robot itself does not collide with ρ). Otherwise, it processes each grasp in s_+. If $\langle \rho, g \rangle$ has not already been cached as a validation condition, it is computed. If $\rho =$ **None**, the procedure computes whether the robot moving along $e.\tau$ with grasp g either causes a self-collision or g collides with fixed obstacles using ROBOT-GRASP-C. Otherwise when $\rho = (o,p)$, the procedure computes whether the grasp g moving along $e.\tau$ causes a collision with object o at pose p using GRASP-OBJ-C. Then, it can check $e.valid$ to obtain the cached answer to the query. If TEST-GRASP finds a satisfying grasp, it returns **True**. Otherwise, it returns **False**.

Similarly, the procedure TEST-OBJ computes, for an edge e, the relaxed state s_+, and an object o, whether the edge is valid for some pose p in the relaxed state. If **None** is a pose for object o in s_+, the procedure can immediately

```
TEST-GRASP(s+, e, ρ) :
1  if None ∈ s+(v_h):
2      e.ach ∪= {v_h ← None}
3      return True
4  for g ∈ s+(v_h):
5      if ⟨ρ, g⟩ ∉ e.valid:
6          e.valid[⟨ρ, g⟩] = (ρ = None and
             not ROBOT-GRASP-C(e.τ, g)) or
             (ρ ≠ None and not GRASP-OBJ-C(e.τ, g, ρ))
7      if e.valid[⟨ρ, g⟩]:
8          e.ach ∪= {v_h ← g}
9          return True
10 return False
```

```
TEST-OBJ(s+, e, o) :
1  if None ∈ s+(v_o):
2      e.ach ∪= {o ← None}
3      return True
4  for p ∈ s+(v_o):
5      if ⟨(o, p), None⟩ ∉ e.valid:
6          e.valid[⟨(o, p), None⟩] =
             not ROBOT-OBJ-C(e.τ, (o, p))
7      if e.valid[⟨(o, p), g⟩] and TEST-GRASP(s+, e, (o, p))
8          e.ach ∪= {o ← p}
9          return True
10 return False
```

```
TEST-REACHABLE({c_1, ..., c_n}, s+; (V, E)) :
1  q = c_1.q
2  E' = E
3  for e ∈ E:
4      if not TEST-GRASP(s+, e, None):
5          E' = E' \ {e}
6          continue
7      for i ∈ [m]:
8          if not TEST-OBJ(s+, e, o_i):
9              E' = E' \ {e}
10             break
11 (N, T) = BFS-TREE(q; (V, E'))
12 return {⟨c_i, TRACE-ACH(c_i.q'; (N, T))⟩ | c_i.q' ∈ N}
```

Fig. 17. Procedure for querying the CRG.

report success because it can navigate the edge when o is not placed. Otherwise, it processes each pose for o in s_+. If $\langle(o,p),g\rangle$ has not already been cached as a validation condition, it is computed. The procedure computes whether the robot moving along $e.\tau$ causes a collision with o at pose p using ROBOT-OBJ-C. It then checks *e.valid* to obtain the cached answer to the query and calls TEST-GRASP with placement $\rho = (o,p)$ to see if this pose admits a legal grasp. If TEST-OBJ finds a satisfying pose, it returns **True**. Otherwise, it returns **False**.

Many of these checks can be done very efficiently using simple bounding-box computations. Additionally, the CRG caches the polyhedral structure of the robot along the trajectory, which can speed up later collision checks along the same edge by not having to place the robot a second time. Within TEST-REACHABLE, if there is not a valid

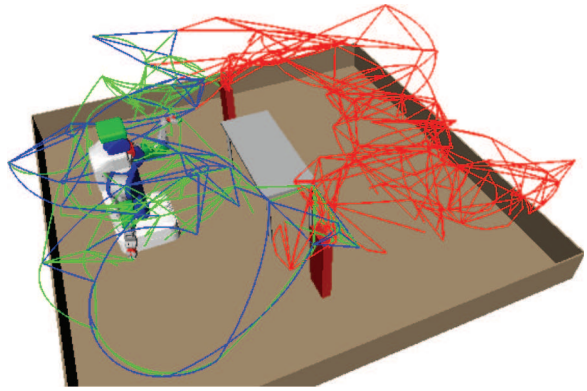

Fig. 18. The reachable CRG. Green edges are on the BFS tree. Blue edges have reachable vertices but were not selected for the BFS tree. Red edges have at least one unreachable vertex.

grasp and pose for each object that allows safe passage across an edge, the edge is removed from the subgraph E'. Finally, BFS-TREE$(q;(V,E'))$ performs a standard breadth-first search through the subgraph to construct a breadth-first search BFS tree (N,T). TEST-REACHABLE then returns the subset of $\{c_1,...,c_n\}$, which have end configuration q' in (N,T). Additionally, a set of achiever variable values is computed using TRACE-ACH$(c_i.q';(N,T))$ by tracing a path to q and taking the union of *e.ach* for visited edges.

Recall that for standard non-relaxed states, each object can only be at one pose at a time, and the robot can hold at most one object, so, the loops over poses $s_+(v_{o_i})$ and grasps $s_+(v_h)$ will only process a single pose or grasp respectively. This test is also efficient for relaxed states that arise during the relaxed planning process. As soon as an object o_i is picked up, it will have **None** $\in s_+(v_{o_i})$, and no further collision checks using object o_i will be required. Additionally, before a new object pose or grasp can be added to the relaxed state, a **None** pose or grasp must be added from a PICK or PLACE action, respectively. Thus, $s_+(v_{o_i})$ and $s_+(v_h)$ will either only contain a single pose or grasp or they will contain **None**, which expedites the test.

Finally, our implementation of TEST-REACHABLE actually tests the validity of the graph lazily while performing the search. We also exploit the fact that we are only interested in the connected component of the CRG, which includes the current robot configuration, which further increases efficiency. Moreover, for sequential evaluations for the same heuristic, TEST-REACHABLE expands on the previously reachable subgraph to avoid re-evaluating traversable edges.

9. Review of sPRM theoretical analysis

To motivate our FFRob theoretical analysis, we review the theoretical analysis for the *simplified probabilistic roadmap* (sPRM) (Kavraki and Latombe, 1998) over the class of

robustly feasible motion planning problems. Two desirable properties for sampling-based motion planning algorithms are probabilistic completeness and exponential convergence. Exponential convergence implies probabilistic completeness.

Definition 16. An algorithm is *probabilistically complete* over a class of problems if and only if the probability that the algorithm halts and returns a solution is one in the limit as the number of time steps goes to infinity.

Definition 17. An algorithm is *exponentially convergent* over a class of problems if and only if the probability that the algorithm has not terminated and returned a solution decreases exponentially in the number of time steps.

The objective of a single-query motion planning problem is to find a collision-free trajectory in a d-dimensional configuration space $\mathcal{Q} \subseteq \mathbb{R}^d$ between a start configuration $q^0 \in \mathcal{Q}$ and a goal configuration $q^* \in \mathcal{Q}$. As is typical in the analysis of motion planning algorithms, we restrict our analysis to Euclidean configuration spaces. We will call configurations and trajectories *collision-free* if all of their configurations are contained within $\mathcal{Q}$.

9.1. Robust feasibility

First, we identify a class of robustly feasible motion planning problems that have a non-zero volume of solutions.

The robustness restriction is necessary because sampling-based algorithms have zero probability of generating samples on any particular lower dimensional submanifold of $\mathcal{Q}$. Having a non-zero volume of solutions ensures that the solutions are not at some segment constrained to such a submanifold.

We give a definition of a robustly feasible motion planning problem that mixes the ideas of clearance (Kavraki et al., 1998) and ϵ-goodness (Kavraki et al., 1995) in order to classify some motion planning problems where q^0 or q^* is on the boundary of $\mathcal{Q}$ as robustly feasible. We will use $||x||$ as the Euclidean norm on points $x \in \mathbb{R}^d$. Let $\tau : [0, L] \to \mathcal{Q}$ be a trajectory of length L from q^0 to q^* such that $q^0 = \tau(0)$ to $q^* = \tau(L)$. Let $\chi(\tau; \mathcal{Q})$ give the *clearance* of τ, the minimum distance from a configuration on τ to the boundary of $\mathcal{Q}$

$$\chi(\tau; \mathcal{Q}) = \inf_{t \in [0,L]} \inf_{x \in \partial \mathcal{Q}} ||\tau(t) - x||$$

Definition 18. A motion planning problem is *robustly feasible* if there exists a trajectory τ from $q^0_+ \in \mathcal{Q}$ to $q^*_+ \in \mathcal{Q}$ and $\delta > 0$ such that

- $\chi(\tau; \mathcal{Q}) \geq \delta$ and:
- $\forall q \in \mathcal{Q}$ such that $||q - q^0_+|| \leq \delta/2$, $(1-t)\,q^0 + tq \in \mathcal{Q}$ and:
- $\forall q' \in \mathcal{Q}$ such that $||q' - q^*_+|| \leq \delta/2$, $(1-t)\,q' + tq^* \in \mathcal{Q}$

This definition asserts that there is a trajectory with non-zero clearance such that the neighborhoods around its start and end configurations can 'see' (admit a linear path between) q^0 and q^*, respectively. This implies that both q^0 and q^* are ϵ-good for some $\epsilon > 0$. This is a weaker condition than the assertion that there exists a direct trajectory between q^0 and q^* with non-zero clearance. The latter assertion would disqualify motion planning problems where the start or goal are themselves on the boundary of $\mathcal{Q}$, even if a non-negligible volume of $\mathcal{Q}$ could see them. These kinds of motion planning problems are prevalent when grasping objects, making our definition useful when analyzing PPM problems. Figure 19 displays a motion planning problem that is robustly feasible under our definition of the term.

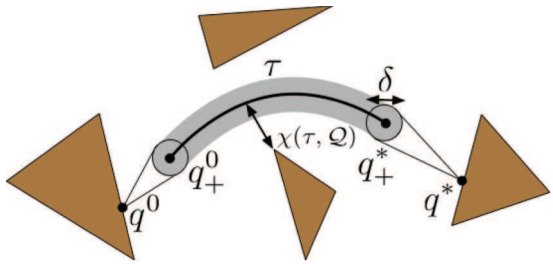

Fig. 19. A robustly feasible motion planning problem.

9.2. Probabilistic completeness

This following theorem states that for any robustly feasible motion planning problem, there exists a finite sequence of d-spheres with non-zero volume for which the linear interpolation of any collection of configurations covering the spheres is a solution to the problem. Let $B(q, r)$ be the d-sphere centered at q with radius r.

Theorem 1. *For any robustly feasible motion planning problem, there exists a sequence of* $k+1$*, where* $k = \lceil 2L/\delta \rceil$*, d-spheres* $(B_0, B_1, ..., B_k)$ *centered at* $\tau(Li/k)$ *for* $i \in \{0, ..., k\}$ *and each with radius* $\delta/2$ *such that any trajectory* τ' *linearly interpolated from* $(q^0, q_0, q_1, ..., q_k, q^*)$*, where* $q_i \in B_i$ *for* $i \in \{0, ..., k\}$*, is a collision-free solution to the motion planning problem.*

We give this proof, which is largely the same as a proof by Kavraki et al. (1998), in Appendix B. This theorem directly reduces motion planning to a sampling problem by identifying these spheres. Thus, sampling-based algorithms, such at PRMs, can produce solutions to robustly feasible motion planning problems by sampling the space until generating samples within these spheres.

Again, we consider completeness properties for the sPRM, a mathematically tractable variant of a PRM. The sPRM starts its roadmap with $V = \{q^0, q^*\}$ and, on each iteration, uniformly at random samples $\mathcal{Q}$ and connects each sampled configuration q to all existing roadmap configurations q' such that $\forall t \in [0, 1], (1-t)\,q + tq' \in \mathcal{Q}$, i.e. the linearly interpolated path between q and q' is collision-free.

Now we prove the main theorem, which is slightly modified from that of Kavraki et al. (1998). Let $\mu(Q)$ be the d-dimensional Lebesgue measure on $Q \subseteq \mathcal{Q}$.

Theorem 2. *The sPRM algorithm is probabilistically complete and exponentially convergent over the class of robustly feasible motion planning problems.*

Proof. Consider any robustly feasible planning problem. By Theorem 1, there exists a sequence of $k+1$, where $k = \lceil 2L/\delta \rceil$, d-spheres with radius $\delta/2$ such that any collection of $k+1$ configuration samples that covers the spheres forms a collision-free, linearly interpolated trajectory. We will construct an upper bound on the probability that, after taking n samples, sPRM has not covered all of the balls. We will assume that the sPRM is able to directly sample $\mathcal{Q}$ by, for instance, sampling a hyper-rectangle subset of $\mathbb{R}^d$ that contains $\mathcal{Q}$ and rejecting samples not containing $\mathcal{Q}$.

We begin by defining σ to be the probability that a random sample is inside some particular sphere (and note that this probability is equal for all the spheres). It is equal to the ratio of the measure of the sphere to the measure of the free configuration space Q

$$\Pr[\text{sample is in } B_i] = \frac{\mu(B(\tau(Li/k), \delta/2))}{\mu(\mathcal{Q})} = \sigma \qquad (1)$$

Note that $\sigma \in [0, 1]$ is a constant with respect to n. Now, the probability that all n samples are outside ball i is $(1-\sigma)^n$ which is bounded above by $e^{-\sigma n}$.

The algorithm will fail if, for any ball, all n samples are outside it; we bound this probability using the union bound

$$\Pr[\bigcup_{i=0}^{k} \text{all } n \text{ samples are outside } B_i] \leq \sum_{i=0}^{k} e^{-\sigma n} \qquad (2)$$

$$= \left\lceil \frac{2L}{\delta} + 1 \right\rceil e^{-\sigma n} \qquad (3)$$

Note that $\lim_{n\to\infty} \lceil 2L/\delta + 1 \rceil e^{-\sigma n} = 0$. Because the probability of failure decreases exponentially in n, the sPRM algorithm is exponentially convergent, which implies it is probabilistically complete. □

10. FFR_{OB} Theoretical analysis

The probabilistic completeness and exponential convergence proofs for FFRob build on ideas from the sPRM proofs. The high-level structure of the argument is the same. We first identify a class of robustly feasible PPM problems that have a non-zero volume of solutions. This is more complicated than in the pure motion planning case because we must reason not only about volumes of trajectories but also volumes of placements, grasps, and inverse kinematic solutions. Then, we describe a simplified version of FFRob. Finally, we show that this version of FFRob is probabilistically complete and exponentially convergent.

Because samples of configurations, poses, and grasps come from different domains, we need a definition of volume relative to each particular space. For the rest of the proof, let $\mu(E; X)$ generically be a measure on subsets E of the set X that assigns finite, non-zero measure to X. This could be for discrete X a counting measure, for Euclidean manifolds a Lebesgue measure of appropriate dimensionality, for rotation groups a Haar measure, and so on. Furthermore, it will be useful to define a normalized version of this measure, $\tilde{\mu}$, where

$$\tilde{\mu}(E; X) = \frac{\mu(E; X)}{\mu(X; X)}$$

When picking and placing objects, we must frequently reason about the set of robot configurations that can reach a particular end-effector transformation. This set of inverse kinematics solutions is frequently a low-dimensional submanifold in $\mathbb{R}^d$, which cannot always be sampled reliably. We denote the set of inverse kinematics solutions for a manipulator transform t as

$$Q_{ik}(t) = \{q \in \mathcal{Q} \mid \text{END-EFFECTOR}(q) = t\}$$

We assume that we can randomly sample from $Q_{ik}(t)$ with probability density bounded away from zero by some $\kappa > 0$. This could be done by, for example, obtaining an analytical representation of the inverse kinematic solutions and sampling free parameters. This, particularly for high-dimensional robots, is an open problem. The probabilistic completeness of FFRob rests on having such a sampler.

10.1. Robust feasibility

We now define a robustly feasible PPM problem by first introducing the components of PPM solutions that must themselves be robust for a PPM problem to be robust.

10.1.1. Mode-constrained motion planning. Many manipulation problems can be thought of as motion planning through multiple *modes*, where each mode ω defines constraints that restrict the system's operable state-space (Hauser and Ng-Thow-Hing, 2011). Specifically, plans in PPM domains can be described as motion plans through an alternating sequence of transit and transfer modes (Siméon et al., 2004). A transit mode ψ corresponds to the robot moving while not holding anything, and a transfer mode ϕ corresponds to the robot moving while holding an object at a fixed grasp. Each mode is defined by a set of parameters, which give rise to its constraints.

Definition 19. A *transit mode* $\psi_i = \langle \textbf{None}, \{p_i^{o_j} \mid \forall j \in [m]\}\rangle$ has parameters consisting of poses for each object $p_i^{o_j} \in \mathcal{P}^{o_j}$. The grasp is **None**. A transit mode is legal if there are no collisions among the placed objects. A transit mode constrains the robot to move in $\mathcal{Q}_{\psi_i} \subseteq \mathcal{Q}$, the subset of $\mathcal{Q}$ that does not collide with the placed objects.

Definition 20. A *transfer mode* $\phi_i = \langle g_i^{o_{h_i}}, \{p_i^{o_j} \mid \forall j \in [m], j \neq h_i\}\rangle$ has parameters consisting of a grasp $g_i^{o_{h_i}} \in \mathcal{G}^{o_{h_i}}$ for an object o_{h_i} and the poses for each other object $p_i^{o_j} \in \mathcal{P}^{o_j}$. A transfer mode is legal if there are no collisions among the placed objects. A transfer mode constrains the robot to move in $\mathcal{Q}_{\phi_i} \subseteq \mathcal{Q}$, the subset of $\mathcal{Q}$ where neither it nor object o_{h_i}, held at grasp $g_i^{o_{h_i}}$, collide with the placed objects nor o_{h_i} intersects with the robot.

In both modes, the poses of the non-held objects are fixed and constrain legal movements of the robot. Additionally, in transfer modes, the held object must remain at the same grasp transform relative to the robot's end-effector, effectively altering the geometry of the robot. The state of the system can always be derived from just the current transit or transfer mode and the current robot configuration. As such, when the current mode is fixed, we can reason about the state of the system by just reasoning in the space of robot configurations subject to the mode.

The problem of planning robot movements subject to a mode is called a *mode-constrained* motion planning problem. This is simply a standard motion planning problem with an additional mode input ω, which defines the *operable* state-space $\mathcal{Q}_\omega \subseteq \mathcal{Q}$ of the problem. A mode-constrained motion planning problem is robustly feasible when the corresponding motion planning problem in the restricted configuration space $\mathcal{Q}_\omega$ is robustly feasible.

Clearly, the sPRM is both probabilistically complete and exponentially convergent for robustly feasible mode-constrained motion planning problems by applying the exact same arguments from Theorems 1 and 2.

10.1.2. Multi-mode-constrained motion planning. A robot must usually move in a sequence of modes in order to solve manipulation problems. Two modes are *adjacent* if the intersection between their operable system state-spaces is non-empty. Two unique transit modes cannot be adjacent because at least one object pose must differ between them, and the robot must grasp the object in order to move it between poses. Similarly, two unique transfer modes cannot be adjacent because an object pose or the grasp differs between them. In either case, the robot must change the current grasp to move the object or obtain a new grasp. Transit and transfer modes, however, can be adjacent.

Definition 21. A transit mode ψ_i and transfer mode ϕ_j are *adjacent* if and only if $p_i^{o_a} = p_j^{o_a}\ \forall a \in [m], h_j \neq a$, and $\mathcal{Q}_{\psi_i} \cap \mathcal{Q}_{\phi_j} \neq \emptyset$.

In adjacent modes, the poses of the objects that are not grasped in the transfer mode are fixed between the two modes (reflecting that the robot can only manipulate a single object at a time) and the set of robot configurations that can move between the modes is non-empty. The system can perform a *mode switch* between two adjacent modes when it is at a state in the intersection of the two operable state-spaces. For transit and transfer modes, this is equivalent to a robot being at a configuration $q \in \mathcal{Q}_{\psi_i} \cap \mathcal{Q}_{\phi_j}$. Mode switches from ψ_i to ϕ_j are PICK actions. Conversely, mode switches from ϕ_j to ψ_i are PLACE actions. Both of these actions involve object o_{h_j} and grasp $g_j^{o_{h_j}}$ from the transfer mode as well as pose $p_i^{o_{h_j}}$ from the transit mode. Precisely, the set of PICK actions $A_{\psi_i}^{\phi_j}$ that can switch from from ψ_i to ϕ_j are

$$A_{\psi_i}^{\phi_j} = \{\text{PICK}(o_{h_j}, p_i^{o_{h_j}}, g_j^{o_{h_j}}, q) \mid q \in \mathcal{Q}_{\psi_i} \cap \mathcal{Q}_{\phi_j}\}$$

and the set of PLACE actions $A_{\phi_j}^{\psi_i}$ that can switch from from ϕ_j to ψ_i are

$$A_{\phi_j}^{\psi_i} = \{\text{PLACE}(o_{h_j}, p_i^{o_{h_j}}, g_j^{o_{h_j}}, q) \mid q \in \mathcal{Q}_{\phi_j} \cap \mathcal{Q}_{\psi_i}\}$$

Define $t(\psi_i, \phi_j) = p_i^{o_{h_j}} \times \text{TRANSFORM}(g_j^{o_{h_j}})^{-1}$ as the end-effector transform for grasping object o_{h_j} at pose $p_i^{o_{h_j}}$ with grasp $g_j^{o_{h_j}}$. For notational simplicity, assume that the arguments to t are unordered so $t(\phi_j, \psi_i) = t(\psi_i, \phi_j)$. $\mathcal{Q}_{\psi_i} \cap \mathcal{Q}_{\phi_j} \subseteq \mathcal{Q}_{ik}(t(\psi_i, \phi_j))$ is the collision-free subset of the inverse kinematic solutions for the end-effector transform at the mode switch. These inverse kinematic configurations will serve as targets for motion planning to move between modes.

A sequence of k legal, adjacent transit and transfer modes $(\omega_1, \omega_2, ..., \omega_k)$ is a *mode sequence*. A PPM mode sequence has $k-1$ mode switches; i.e. $k-1$ PICK and PLACE actions. We will generically refer to the ith mode as ω_i. Given both a mode sequence and whether ω_1 is a transit or transfer mode allows us to determine the type of any other mode ω_i depending on whether i is odd or even.

We will now look at *multi-mode-constrained* motion planning problems from a start configuration $q_0 \in \mathcal{Q}_{\omega_1}$ to a set of end configurations $\mathcal{Q}_* \subseteq \mathcal{Q}_{\omega_k}$ through a fixed sequence of modes. These problems can be reduced to a sequence of k mode-constrained motion planning problems. However, there is a complication that the start q_i^0 and goal q_i^* for the ith mode-constrained motion planning problem (with the exception of the first and last problems) are not given. These must be chosen from the intersection of the neighboring modes such that $q_i^0 \in \mathcal{Q}_{\omega_{i-1}} \cap \mathcal{Q}_{\omega_i}$ and $q_i^* \in \mathcal{Q}_{\omega_i} \cap \mathcal{Q}_{\omega_{i+1}}$. The individual mode motion plans must connect continuously such that $q_i^* = q_{i-1}^0$. Thus, the problem requires choosing these target configurations as well as finding mode motion plans that connect between them. A multi-mode-constrained motion planning problem is robustly feasible when there exists a sequence of sets of target configurations with non-zero measure such that any mode-constrained motion planning problem between pairwise targets is robustly feasible.

Definition 22. A multi-mode-constrained motion planning problem is *robustly feasible* if for a mode sequence $(\omega_1, \omega_2, ..., \omega_k)$ there exists a sequence of sets of configurations $(\mathcal{Q}_0, \mathcal{Q}_1, ..., \mathcal{Q}_k)$ and $\epsilon > 0$ such that

- $Q_0 = \{q_0\}$ and:
- $\forall i \in [k-1]\ Q_i \subseteq Q_{ik}(t(\omega_i, \omega_{i+1}))$ and $\tilde{\mu}(Q_i; Q_{ik}(t(\omega_i, \omega_{i+1}))) \geq \epsilon$ and:
- $Q_k \subseteq Q_*$ and $\tilde{\mu}(Q_k; Q_*) \geq \epsilon$ and:
- $\forall i \in [k]\ \forall q_i^0 \in Q_{i-1}\ \forall q_i^* \in Q_i$ the ω_i mode-constrained motion planning problem from q_i^0 to q_i^* is robustly feasible.

10.1.3. PPM Planning. A multi-mode-constrained motion planning problem is easier than a PPM problem because the mode sequence is given as an input. We now return to full PPM problems that require selection of a mode sequence in addition to finding a multi-mode motion plan. The start mode must be $\omega_1 = \langle g_0^{o_{h_0}}, \{p_0^{o_1}, ..., p_0^{o_m}\}\rangle$. However, the goal mode can be any mode ω_k such that $g_k^{o_{h_k}} \in G_*^{o_{h*}}$ and $\forall j \in [m].p_k^{o_j} \in P_*^{o_j}$.

Suppose now we must determine the sequence of modes. Starting from ω_1 and for each newly chosen mode, each legal mode switch from the last mode ω_i to a new mode ω_{i+1} can be described by a single parameter.

Between a transit mode $\omega_i = \psi_i$ and a new transfer mode, the object poses, with the exception of a grasped object, remain constant. Thus, the new transfer mode $\omega_{i+1} = \phi_{i+1}$ can be inferred from just ψ_i and the specification of a free parameter $g_{i+1}^{o_{h_{i+1}}}$ for the resulting grasp. Similarly, between a transfer and transit mode, the object poses, with the exception of a grasped object, also remain constant leaving a free parameter $p_{i+1}^{o_{h_i}}$ for the new pose of o_{h_i} to give the new mode. Thus, a sequence of modes starting from ω_1 can be completely described by an alternating sequence of $(k-1)$ poses and grasps such as $(g_2^{o_{h_2}}, p_3^{o_{h_2}}, ..., p_k^{o_{k-1}})$. It is sufficient to choose these poses and grasps to identify a mode sequence.

In order for a PPM problem to be robustly feasible, there must be a non-zero volume of mode sequences that admit robust multi-mode motion plans. As previously suggested, the set of length-k mode sequences is contained in the space Θ formed by the Cartesian product of the $k-1$ alternating pose and grasp parameter domains $\mathcal{P}^{o_i}$ and $\mathcal{G}^{o_j}$. Suppose that we are considering mode sequences starting and ending with transit modes where the transfer modes interact with this prescribed sequence of objects $(o_{h_2}, o_{h_4}, ..., o_{h_{k-1}})$. The space containing the set of valid mode sequences is $\Theta = \mathcal{G}^{o_{h_2}} \times \mathcal{P}^{o_{h_2}} \times ... \times \mathcal{P}^{o_{h_{k-1}}}$. We can define a set of mode sequences $\theta = G_2^{o_{h_2}} \times P_3^{o_{h_2}} \times ... \times P_k^{o_{h_{k-1}}}$ by a set of values for each parameter such that $\theta \subseteq \Theta$.

Let $P_i^{o_{h_{i-1}}}$ when $(i-j)$ is even and $G_j^{o_{h_j}}$ when $(i-j)$ is odd refer to these sets of grasps and poses that together define a collection of mode sequences from a prescribed sequence of transit modes and transfer modes. We could measure the volume of these mode sequences by just taking the product of each $\tilde{\mu}(P_i^{o_{h_{i-1}}}; \mathcal{P}^{o_{h_{i-1}}})$ and $\tilde{\mu}(G_j^{o_{h_j}}; \mathcal{G}^{o_{h_j}})$. However, the goal of these mode sequences is to reach a goal mode. This requires choosing grasps and poses within $G_*^{o_{h*}}$ and $P_*^{o_i}$ for $i \in [m]$, respectively. A PPM problem may specify a set of goal poses or grasps that is considerably smaller or has lower dimensionality than the full space of these values. For example, a problem could specify a goal set $P_*^{o_i} = \{p_*^{o_i}\}$ as a single pose, where clearly $\tilde{\mu}(P_*^{o_i}; \mathcal{P}^{o_i}) = 0$. Yet, we still expect some of these problems to be robustly feasible because an algorithm could intentionally sample the goal set $P_*^{o_i}$ in addition to the full domain $\mathcal{P}^{o_i}$. Thus, we will define the measure of $P_i^{o_{h_{i-1}}}$ and $G_j^{o_{h_j}}$ as follows

$$\tilde{\mu}(P_i^{o_{h_{i-1}}}) = \begin{cases} \tilde{\mu}(P_i^{o_{h_{i-1}}}; P_*^{o_{h_{i-1}}}) & P_i^{o_{h_{i-1}}} \subseteq P_*^{o_{h_{i-1}}} \\ \tilde{\mu}(P_i^{o_{h_{i-1}}}; \mathcal{P}^{o_{h_{i-1}}}) & \text{otherwise} \end{cases}$$

$$\tilde{\mu}(G_j^{o_{h_j}}) = \begin{cases} \tilde{\mu}(G_j^{o_{h_j}}; G_*^{o_{h*}}) & o_{h_j} = o_{h*} \text{ and } G_j^{o_{h_j}} \subseteq G_*^{o_{h*}} \\ \tilde{\mu}(G_j^{o_{h_j}}; \mathcal{G}^{o_{h_j}}) & \text{otherwise} \end{cases}$$

Intuitively, these measures are taken with respect to the set of goal values when the set to be measured is a subset of the goal values. Otherwise, the measures are taken with respect to the full domain of values. Finally, we arrive at the definition of a robustly feasible PPM problem.

Definition 23. A PPM problem Π is *robustly feasible* if there exists a set of length-k mode sequences $\theta = \{(\omega_1, \omega_2, ..., \omega_k), (\omega_1', \omega_2', ..., \omega_k'), ...\}$ with transfer modes involving a common sequence of objects $(..., o_{h_i}^*, o_{h_{i+2}}^*, ...)$ such that

- $\forall (\omega_1, \omega_2, ..., \omega_k) \in \theta$:
  - $\omega_1 = \langle g_0^{o_{h_0}}, \{p_0^{o_1}, ..., p_0^{o_m}\}\rangle$.
  - $o_{h_i} = o_{h_i}^*$ if ω_i is a transfer mode.
  - $g_k^{o_{h_k}} \in G_*^{o_{h*}}$ and $\forall j \in [m].p_k^{o_j} \in P_*^{o_j}$.
  - The multi-mode motion planning problem from q_0 to Q_* using $(\omega_1, \omega_2, ..., \omega_k)$ is robustly feasible.
- For the ith set of transfer modes $\{\omega_i, \omega_i', ...\}$, $\tilde{\mu}(P_i^{o_{h_{i-1}}}) > 0$.
- For the jth set of transit modes $\{\omega_j, \omega_j', ...\}$, $\tilde{\mu}(G_j^{o_{h_j}}) > 0$.

Intuitively, a robustly feasible PPM problem has non-negligible volumes of sequential poses and grasps such that any choice of these poses and grasps allows a sequence of robust motion plans that can connect them and solve the problem. The non-negligible volumes of poses relate to the robustness definition given by Van Den Berg et al. (2009). Their definition says a problem is robust if there exists a sequence of poses that could be simultaneously perturbed by a small amount Δ without changing the feasibility of the resulting motion planning problems. Relating this to our definition, Δ is the radius of an n-sphere that is centered at each pose in the sequence where each n-sphere denotes a volume of safe poses.

Our robustness condition forbids overly constrained problems where solution mode sequences are restricted to a lower dimensional submanifold of the mode sequence parameter space.

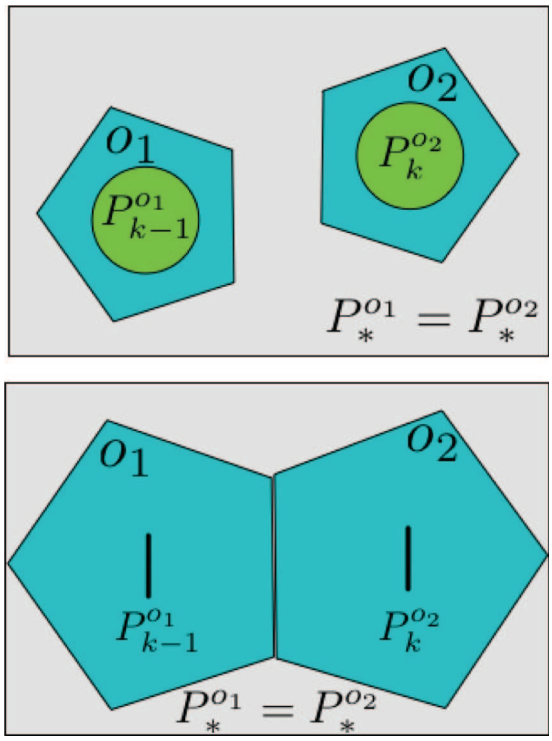

Fig. 20. Illustration of robust feasibility applied to placements. The goal poses $P^{o_1}_*$ and $P^{o_2}_*$ for the blue pentagons o_1, o_2 are poses contained in the grey region. The top scenario shows candidate $P^{o_1}_{k-1}$ and $P^{o_2}_k$ with non-zero areas. The bottom scenario only admits $P^{o_1}_*$ and $P^{o_2}_*$, which are lines giving each zero area.

Such problems require special samplers that are able to produce tuples of values, possibly for different types of parameters, that satisfy a constraint among them. Specifying such a sampler for every combination of parameters is intractable because the number of combinations grows exponentially.

10.1.4. Example. Consider a physical interpretation of robustness for placements. Suppose all modes are chosen from the set of mode sequences apart from the ith transfer mode. This consequently fixes the sequence of placements for all but the ith placement. The robustness condition asserts that for any combination of fixed placements, any choice of the ith placement from $P_i^{o_{h_{i-1}}}$, and therefore any choice of the ith transfer mode, will not collide with the existing placements. This allows for placements to be chosen independently with respect to each other.

Figure 20 gives a example of a robust and non-robust problem involving goal poses. The top problem is robust because $P^{o_1}_{k-1}$ and $P^{o_2}_k$ are both balls of poses with non-zero volume, meaning that o_1 and o_2 could be at any pair of poses from these sets respectively and satisfy the goal without collision. The bottom problem is not robust because $P^{o_1}_{k-1}$ and $P^{o_2}_k$ are lines indicating that the objects can only be moved up and down and result in a collision-free solution.

10.2. Probabilistic completeness

Now that we have defined a robustly feasible PPM problem, we can return to analyzing FFRob. Recall that on each iteration, FFRob generates a finite number of samples of poses, grasps, and configurations and then performs a discrete search to determine if this set of samples is sufficient to solve the PPM problem. Upon the discrete search failing to find a plan, FFRob increases the number of samples on the next iteration. The algorithm terminates if and when the discrete search finds a plan. For the analysis, we assume that samples x are drawn from a space X uniformly at random as denoted by $x \sim X$.

10.2.1. Convergence in iterations. Let n represent the number of sampling iterations. Let $P^{o_i}(n)$, $G^{o_i}(n)$, $Q^{o_i}(n)$ be the sets of sampled poses, grasps, and inverse kinematic configurations involving object o_i after n sampling iterations. For $i \in [m]$, $P^{o_i}(0) = \{p_0^{o_i}\}$ and $G^{o_i}(0) = \emptyset$ if $p_0^{o_i} \neq$ **None**, otherwise, $P^{o_i}(0) = \emptyset$ and $G^{o_i}(0) = \{g_0^{o_i}\}$. However, $Q^{o_i}(0) = \emptyset$ for all $i \in [m]$. Additionally, let $(V(n), E(n))$ define the vertices and edges of the CRG after n sampling iterations where $V(n) = \emptyset$ and $E(n) = \emptyset$. On the nth sampling iteration, for all $i \in [m]$

- $P^{o_i}(n) = P^{o_i}(n-1) \cup \{p^{o_i}, p_*^{o_i}\}$ where $p^{o_i} \sim \mathcal{P}^{o_i}$ and $p_*^{o_i} \sim P_*^{o_i}$.
- $G^{o_i}(n) = G^{o_i}(n-1) \cup \{g^{o_i}, g_*^{o_i}\}$ where $g^{o_i} \sim \mathcal{G}^{o_i}$ and when $o_i = o_{h_*}$, $g_*^{o_i} \sim G_*^{o_{h_*}}$.
- $Q^{o_i}(n) = Q^{o_i}(n-1) \cup \{q \sim Q_{ik}(p \times \text{TRANSFORM}(g)^{-1}) \mid p \in P^{o_i}(n), g \in G^{o_i}(n)\}$

On each iteration and for each object, FFRob samples a pose, goal pose, grasp, and goal grasp (if the object has a holding goal). Additionally, it samples an inverse kinematics solution for each existing pair of poses and grasps. Each pose and grasp pair will continue to have new inverse kinematic solutions sampled as n increases. FFRob will remain probabilistically complete and exponentially convergent as long as the number of samples on iteration n is bounded by some polynomial in n.

To simplify the analysis, we will consider a simplified version of the CRG that is an sPRM in $\mathcal{Q}$. As stated in Section 9, the sPRM is a roadmap that always attempts to connect every pair of sampled configurations, which leads to an exponentially convergent motion planning algorithm. We only construct one roadmap as opposed to growing a separate roadmap for each grasp or combination of object placements. Although grasping an object changes the geometry of the robot, it only decreases the operable $\mathcal{Q}$. The same is true for placing objects. Thus, the CRG grown in the full configuration space $\mathcal{Q}$ with the hand empty and no placed objects will be sufficient to also capture paths in different transit and transfer modes.

On each sampling iteration, the inverse kinematics solutions that were previously sampled are added to the CRG. A single free space robot configuration q and a single goal robot configuration q_* are also sampled and added to the

CRG. Although in practice the CRG is constructed using larger sample batches, sampling individual configurations simplifies the analysis without loss of generality. Then, the set of edges is constructed for every possible pair of vertices that does not have collisions

- $V(n) = V(n-1) \cup Q^{o_1}(n) \cup ... \cup Q^{o_m} \cup \{q, q_*\}$ where $q \sim \mathcal{Q}$ and $q_* \sim \mathcal{Q}_*$.
- $E(n) = \{(v, v'), (v', v) \mid v, v' \in V, \text{COLLISION-FREE}(v, v'; \mathcal{Q})\}$

After the samples are selected for the nth iteration, the actions passed to the discrete planner are the following

- Move actions: $A_{\text{move}} = \{\text{MOVE}(q \in V, q' \in V, (V, E)) \mid q \neq q'\}$.
- Pick actions for $i \in [m]$: $A^i_{\text{pick}} = \{\text{PICK}(o_i, p \in P^{o_i}(n), g \in G^{o_i}(n), q \in Q^{o_i}(n)) \mid q \in Q_{ik}(p \times \text{TRANSFORM.}(g)^{-1})\}$
- Place actions for $i \in [m]$: $A^i_{\text{place}} = \{\text{PLACE}(o_i, p \in P^{o_i}(n), g \in G^{o_i}(n), q \in Q^{o_i}(n)) \mid q \in Q_{ik}(p \times \text{TRANSFORM}(g)^{-1})\}$

Finally, because the heuristic search algorithms that work in practice do not improve symbolic planning's worst case complexity, we simplify FFRob such that it performs an unguided search.

We start by identifying the convergence rate as a function of the number of sampling iterations. We then perform a change of variables to show that it also converges exponentially in the number of time steps.

Theorem 3. *For any robustly feasible PPM problem Π, the probability that FFRob fails to find a solution decreases exponentially as the number of sampling iterations $n \to \infty$.*

Proof. Recall that because FFRob's discrete search is complete, it will find a solution if a sufficient set of pose, grasp, and configuration samples are chosen. FFRob will fail to find a solution by the nth sampling iteration if and only if the set of samples on the nth iteration does not contain a solution. We first divide this sampling failure into the disjunction of three disjoint failure events: an insufficient set of mode samples, an insufficient set of mode switch inverse kinematic samples assuming a sufficient set of mode samples, and an insufficient set of CRG configurations assuming sufficient sets of mode and mode switch samples. Let N_{FFRob} be a non-negative random variable for the number of iterations before FFRob finds a plan given a robustly feasible PPM problem. Let N_{mode}, N_{switch}, and N_{CRG}, be non-negative random variables for the number of sampling iterations before FFRob has produced a sufficient set of mode, mode switch, and CRG samples, respectively. The probability that FFRob fails to find a plan is the sum of the probabilities of the three disjoint failure events

$$\begin{aligned}\Pr[N_{\text{FFRob}} > n] = \Pr[N_{\text{mode}} > n] \\ + \Pr[N_{\text{mode}} \leq n, N_{\text{switch}} > n] \\ + \Pr[N_{\text{mode}}, N_{\text{switch}} \leq n, N_{\text{CRG}} > n]\end{aligned}$$

We start with $\Pr[N_{\text{mode}} > n]$. Recall that a robustly feasible PPM problem has a non-negligible volume of mode sequences, which can be represented by an alternating sequence of $(k-1)$ sets of pose and grasp samples. Let $P_i^{o_{h_{i-1}}}$ and $G_j^{o_{h_j}}$ be these sets of samples. The probability of a mode sampling failure is bounded by the sum of the probabilities of a pose and grasp sampling failure

$$\Pr[N_{\text{mode}} > n] \leq \Pr[N_{\text{pose}} > n] + \Pr[N_{\text{grasp}} > n] \quad (4)$$

Because, on each iteration, FFRob independently, uniformly at random samples a pose from both the full domain and the goal for each object, the probability that a new sample is in a set of poses $P_i^{o_{h_{i-1}}}$ is equal to the measure of $P_i^{o_{h_{i-1}}}$ using the relative measure $\tilde{\mu}$

$$\Pr[p^{o_{h_{i-1}}} \in P_i^{o_{h_{i-1}}}] = \tilde{\mu}(P_i^{o_{h_{i-1}}})$$

For simplicity, define $\rho_p = \min_i \Pr[p^{o_{h_{i-1}}} \in P_i^{o_{h_{i-1}}}]$. Using the union bound, the probability of a pose failure is less than the sum of the probabilities that all n pose samples land outside $P^{o_{h_{i-1}}}(n) \cap P_i^{o_{h_{i-1}}}$

$$\begin{aligned}\Pr[N_{\text{pose}} > n] &= \Pr\Big[\bigvee_i (P^{o_{h_{i-1}}}(n) \cap P_i^{o_{h_{i-1}}} = \emptyset)\Big] \\ &\leq \sum_i \Pr\big[P^{o_{h_{i-1}}}(n) \cap P_i^{o_{h_{i-1}}} = \emptyset\big] \\ &\leq \sum_i \big(1 - \Pr[p^{o_{h_{i-1}}} \in P_i^{o_{h_{i-1}}}]\big)^n \\ &\leq (k-1)(1-\rho_p)^n \\ &\leq k e^{-\rho_p n}\end{aligned}$$

Similarly, for a set of grasps $G_j^{o_{h_j}}$

$$\Pr[g^{o_{h_j}} \in G_j^{o_{h_j}}] = \tilde{\mu}(G_j^{o_{h_j}})$$

Analogously, define $\rho_g = \min_i \Pr[g^{o_{h_j}} \in G_j^{o_{h_j}}]$ as the minimum probability of a pose or grasp from each set in the sequence. Then

$$\begin{aligned}\Pr[N_{\text{grasp}} > n] &= \Pr\Big[\bigvee_j (G^{o_{h_j}}(n) \cap G_j^{o_{h_j}} = \emptyset)\Big] \\ &\leq \sum_j \Pr\big[G^{o_{h_j}}(n) \cap G_j^{o_{h_j}} = \emptyset\big] \\ &\leq \sum_j \big(1 - \Pr[g^{o_{h_j}} \in G_j^{o_{h_j}}]\big)^n \\ &\leq (k-1)(1-\rho_g)^n \\ &\leq k e^{-\rho_g n}\end{aligned}$$

Moving on to $\Pr[N_{\text{mode}} \leq n, N_{\text{switch}} > n]$, recall that FFRob samples an inverse kinematic configuration for all grasp and pose pairs on each iteration. Thus, the events for obtaining successful mode samples and successful mode

switch samples are not independent. Consider the following conditional probability $\Pr[N_{\text{switch}} > n \mid N_{\text{mode}} = i]$. Given $N_{\text{mode}} = i$, FFRob will have $(n - i)$ iterations to generate the mode switches. To simplify the analysis, we upper bound $\Pr[N_{\text{mode}} \leq n, N_{\text{switch}} > n]$ by splitting the sampling into two failure cases, $[N_{\text{mode}} > n/2]$ and $[N_{\text{mode}} \leq n/2, N_{\text{switch}} > n\ ]$, which allows each case to be analyzed independently

$$\begin{aligned}\Pr[N_{\text{mode}} \leq n, N_{\text{switch}} > n] \leq{}& \Pr[N_{\text{mode}} > n/2] \\ &+ \Pr[N_{\text{mode}} \leq n/2, N_{\text{switch}} > n]\end{aligned}$$

We already have a bound for $\Pr[N_{\text{mode}} > n/2]$ from equation (4). What remains is to bound $\Pr[N_{\text{mode}} \leq n/2, N_{\text{switch}} > n]$. Each grasp and pose mode sample may be a seed for up to two inverse kinematic configurations. Without conditioning on the event that a full set of successful mode samples have been generated, it may be the case that what would be a successful inverse kinematics for a particular pose and grasp pair could still be a failure if it turned out that no full mode sequence using the pose and grasp was generated. Let Q_i be the set of inverse kinematics solutions associated with the sampled mode sequence. Because the set of samples induces a robustly feasible multi-mode-constrained motion planning problem, there exists $\epsilon > 0$ such that $\tilde{\mu}(Q_i) \geq \epsilon\ \forall i \in \{2, ..., k\}$. Recall that FFRob samples inverse kinematic configurations with probability density bounded away from zero by $\kappa > 0$. We will assume ϵ and κ are the minimum volumes and densities, respectively across any of the mode sequences in θ. Thus, $\Pr[q \in Q_i] \geq \kappa\epsilon\ \forall i \in \{2, ..., k\}$

$$\begin{aligned}&\Pr[N_{\text{mode}} \leq n/2, N_{\text{switch}} > n] \\ &\quad= \sum_{i=1}^{n/2} \Pr[N_{\text{mode}} = i] \Pr[N_{\text{switch}} > n \mid N_{\text{mode}} = i] \\ &\quad\leq \Pr[N_{\text{switch}} > n \mid N_{\text{mode}} = n/2] \sum_{i=1}^{n/2} \Pr[N_{\text{mode}} = i] \\ &\quad\leq \Pr[N_{\text{switch}} > n \mid N_{\text{mode}} = n/2] \\ &\quad= \Pr\Big[\bigvee_{i=2}^{k} (Q(n) \cap Q_i = \emptyset)\Big] \\ &\quad\leq \sum_{i=2}^{k} \Pr\big[Q(n) \cap Q_i = \emptyset\big] \\ &\quad\leq \sum_{i=2}^{k} \big(1 - \Pr[q \in Q_i]\big)^{n/2} \\ &\quad\leq (k-1)(1 - \kappa\epsilon)^{n/2} \\ &\quad\leq k e^{-\kappa\epsilon n/2} \end{aligned} \tag{5}$$

Finally, when analyzing $\Pr[N_{\text{switch}} \leq n, N_{\text{CRG}} > n]$, recall that the CRG samples a roadmap configuration per iteration, independently of any of the other created samples. Thus, we do not have the dependence complication we faced when sampling mode switches. We can upper bound the probability of the conjunction with the probability of the conditional

$$\begin{aligned}&\Pr[N_{\text{mode}} <= n, N_{\text{switch}} <= n, N_{\text{CRG}} > n] \\ &\quad\leq \Pr[N_{\text{CRG}} > n \mid N_{\text{mode}}, N_{\text{switch}} <= n]\end{aligned}$$

A successful set of mode switches will give rise to $(k-1)$ robustly feasible mode-constrained motion planning problems. Although the CRG does not know *a priori* which motion planning problems its samples will help solve, no part of its sampling is dependent on the targets because the samples are uniformly drawn from the unconstrained configuration space $\mathcal{Q}$. The samples created could be useful for any of the motion planning problems that arise. Let L and $\delta > 0$ be the largest and smallest plan lengths and plan clearances across any of the robust motion planning problems. Using the union bound and Theorem 2

$$\Pr[N_{\text{CRG}} > n \mid N_{\text{mode}}, N_{\text{switch}} \leq n] \leq k\left\lceil \frac{2L}{\delta} + 1 \right\rceil e^{-\sigma n} \tag{6}$$

Combining our bounds on the different types of failures, according to equations (4)–(6), we have

$$\begin{aligned}&\Pr[N_{\text{FFRob}} > n] \\ &\leq \Pr[N_{\text{mode}} > n/2] + \Pr[N_{\text{switch}} > n \mid N_{\text{mode}} = n/2] \\ &\quad + \Pr[N_{CRG} > n \mid N_{\text{mode}}, N_{\text{switch}} <= n] \\ &\leq k e^{-\rho_p n/2} + k e^{-\rho_g n/2} + k e^{-\kappa\epsilon n/2} + k\left\lceil \frac{2L}{\delta} + 1 \right\rceil e^{-\sigma n} \\ &= k\left(e^{-\rho_p n/2} + e^{-\rho_g n/2} + e^{-\kappa\epsilon n/2} + \left\lceil \frac{2L}{\delta} + 1 \right\rceil e^{-\sigma n}\right)\end{aligned}$$

Thus, the probability that FFRob fails decreases exponentially in n. □

10.2.2. Convergence in runtime. While we have proven that FFRob converges exponentially in the number of sampling iterations, this does not necessarily imply that it converges exponentially in runtime. For example, if the number of operations between each sampling iteration grows exponentially in n, an algorithm that converges exponentially in n would only converge polynomially in the runtime t. For an example where $t = e^n$, $\Pr[\text{failure}] \leq e^n = e^{\ln t} = 1/t$.

Recall that symbolic planning is known to be PSPACE-Complete (Bylander, 1994). However, the discrete search will actually run in polynomial time of the number of samples because the addition of samples only increases the size of a variable's domain. The number of variables in the state-space $|\mathcal{V}|$ is fixed per problem based on the number of moveable objects m. Thus, the size of the state-space grows only polynomially in the number of samples. The following lemma gives a very loose upper bound on the runtime of the discrete search on the nth sampling iteration.

Lemma 1. *The running time of the discrete search for a PPM problem with m objects on the nth sampling iteration is* $O(m^4 n^{m+9})$.

Proof. Our symbolic planning problem can be viewed as a directed graph search problem where vertices V' are possible states and edges E' are actions performed from a specific state. A discrete search, such as Dijkstra's algorithm, can find the optimal solution to graph search problems in $O(|E'| + |V'| \log |V'|)$. We compute $|V'|$ by analyzing the size of the state-space $S(n) = V'(n)$ after n iterations. Given our sampling strategy, we have the following quantities of samples after the nth iteration □

- Poses for object o_i: $|P^{o_i}(n)| = O(n)$
- Grasps for object o_i: $|G^{o_i}(n)| = O(n)$
- Robot inverse kinematic configurations: $|Q(n)| \leq \sum_{i=1}^{m} \sum_{j=1}^{n} |P^{o_i}(j)||G^{o_i}(j)| = O(mn^3)$
- Robot configurations: $|V(n)| = O(n) + Q(n) = O(mn^3)$

In each state, the robot either has nothing in its hand with all the objects placed, or the robot is holding an object, so all but one objects are placed. Additionally, each state uses a single robot configuration. The size of S is upper bounded by the combinations of samples for these two types of states

$$\begin{aligned}|S(n)| &\leq |V(n)|\Big(\prod_{i=1}^{m} |P^{o_i}(n)| + \sum_{i=1}^{m} |G^{o_i}(n)| \prod_{j\neq i}^{m} |P^{o_i}(n)|\Big)\\ &= O(mn^3(n^m + m(n \times n^{m-1})))\\ &= O(m^2 n^{m+3})\end{aligned}$$

Next, we compute $|E|$ by first calculating the number of possible actions given these samples:

- Move actions: $|A_{\text{move}}(n)| \leq |V(n)|^2 = O(m^2 n^6)$
- Pick actions: $|A_{\text{pick}}(n)| = |Q(n)| = O(mn^3)$
- Place actions: $|A_{\text{place}}(n)| = |Q(n)| = O(mn^3)$

$$\begin{aligned}|A(n)| &= |A_{\text{move}}(n)| + |A_{\text{pick}}(n)| + |A_{\text{place}}(n)|\\ &= O(m^2 n^6)\end{aligned}$$

From $|S(n)|$ and $|A(n)|$, we can obtain a loose upper bound on the number of edges in the state-space graph

$$\begin{aligned}|E'(n)| &= |S(n)||A(n)|\\ &= O(m^4 n^{m+9})\end{aligned}$$

The number of edges dominates the Dijkstra runtime, thus the discrete search runs in $O(m^4 n^{m+9})$.

This analysis was meant to prove that the discrete search runs in polynomial time in n. The polynomial bound is loose and is not indicative of the runtime of common PPM instances, particularly when the search is guided by a heuristic. Moreover, in practice, sampling is often more expensive than searching due to the geometric overhead from collision checks and inverse kinematics.

Theorem 4. *FFRob is probabilistically complete and exponentially convergent.*

Proof. We can compute the change of variables between iterations n and runtime t using Lemma 1. Each iteration takes $O(m) + O(m) + O(mn^2) = O(mn^2)$ time steps to produce the new samples before running the discrete search. This is clearly dominated by the planning time

$$\begin{aligned}t(n) &= \sum_{i=1}^{n} O(m^4 i^{m+9})\\ &= m^4 \sum_{t=1}^{n} O(i^{m+9})\\ &= O(m^4 n^{m+10})\\ &\leq Cm^4 n^{m+10} \text{ for some } C \text{ and } \forall n \geq n_0\end{aligned}$$

Inverting this mapping gives

$$n(t) \geq \Big(\frac{n}{Cm^4}\Big)^{1/(m+10)}$$

Let $C = \min(\rho_p/2, \rho_g/2, \kappa\epsilon/2, \sigma)$ and T_{FFRob} be a non-negative random variable for the number time steps before FFRob finds a solution

$$\begin{aligned}\Pr[T_{\text{FFRob}} > t] &\leq k\Big\lceil 2L/\delta + 4\Big\rceil e^{-C(\frac{n}{Km^4})^{1/(m+10)}}\\ &= O\Big(e^{-n^{1/(m+10)}}\Big)\end{aligned}$$

Again, m is fixed per the problem which slows but does not stop the exponential convergence. □

10.2.3. Corollaries. We arrive at the following corollaries.

Corollary 1. *FFRob has both a finite expected runtime and finite variance in runtime.*

Proof. This follows from the result that an exponentially convergent algorithm has a finite expected runtime and finite variance in runtime (Hauser and Latombe, 2010). □

If FFRob is modified such that it continues to increase its set of samples even after finding a solution, it will converge to a solution that is in the minimal length robust set of mode sequences. This means it will find a solution that uses the fewest PICK and PLACE mode switches out of the set of solutions that are in a robust set of mode sequences. Moreover, this convergence will also be exponential.

Corollary 2. *The probability that FFRob has not identified a solution contained in the minimal length robust set of mode sequences decreases exponentially in the number of time steps.*

Proof. By Theorem 4, the probability that FFRob will not have samples corresponding to a solution for any robust set of mode sequences decreases exponentially as $t \to \infty$. On each iteration, Dijkstra's algorithm will find the optimal plan in terms of the number of PICK and PLACE actions (we will let MOVE actions have a cost of zero while PICK

and PLACE have unit cost). As soon as samples that admit a minimal length mode sequence have been generated, Dijkstra's algorithm will continue to produce a plan with this mode sequence length forever because additional samples could only produce a solution with a smaller length. □

FFRob will not terminate or declare when it has found an optimal solution, but it with high probability will eventually find a solution with minimal mode-sequence length in finite expected runtime. Additionally, the discrete search could be another optimal search algorithm instead of Dijkstra, such as A* with an admissible heuristic, that may have better practical performance. In symbolic planning, a cost-sensitive version of $h_{\max}$ is guaranteed to be admissible. As an additional practical aside, once a candidate solution is found, the discrete search need not visit states that have summed path cost and admissible heuristic cost that exceeds the cost of the previously optimal plan. While this also does not affect the theoretical analysis, it will prune the search space in practice.

11. Experiments

We experimentally evaluated seven configurations of FFRob using eight problems spanning rearrangement planning, NAMO, non-monotonic planning, and task and motion planning.

11.1. Problems

Problems 1–1 & 1–2 are simple rearrangement problems inspired by Krontiris and Bekris (2015). Each block has a specified goal configuration as represented by the color gradient. The robot may use a single top grasp.

Problem 1–1 has two rows of four blocks. Problem 2-1 has two rows of eight blocks and is displayed in Figure 21.

Problems 2-1 & 2-2 combine NAMO with tabletop pick-and-place. They require the robot to move the two blue blocks from the right table to the left table and return to its initial configuration. In order to reach the blue blocks, the robot must first move several red pillars out of way to clear a path for its base. The wall of pillars is composed of two segments where the top segment is thinner than the bottom segment. This is designed to test whether the heuristic can identify actions that lead to shorter plans that only move the top pillars. Problem 2-1 has a top segment of width one and a bottom segment of width two. Problem 2-2, displayed in Figure 22, has a top segment of width two and a bottom segment of width three.

Problems 3–1 & 3–2 are examples of highly non-monotonic problems, problems that require undoing goal conditions along solutions. The robot must move the green blocks from the left table to a corresponding position on the right table. Both the initial and goal poses are blocked by four blue and cyan blocks, respectively. Critically, the blue and cyan blocks have goal conditions to remain in their initial poses. This is the source of the non-monotonicity as the robot must undo several goals by moving the blue and cyan blocks in order to solve the problem. Problem 3-1 includes only one green block. Problem 3-2, displayed in Figure 23, includes three green blocks.

Problem 4 in Figure 24 is from Srivastava et al. (2014). The robot must retrieve the red cylinder from within the cluttered table of cyan cylinders. The goal conditions are for the robot to be holding the red cylinder at its initial configuration.

Problem 5 in Figures 1 and 25 is a task and motion planning problem in a cooking domain. The robot must retrieve two heads of cabbage (green blocks) from the shelves, clean the heads, cook the heads, and place them on the plates. Turnips (pink blocks) are present on the shelves and must be moved to reach the cabbage. However, the turnips must be returned to the shelves if moved. The robot must also wash the cups (blue and cyan blocks) and set the table using the blue cups. The cyan cup is not needed for dinner. This problem requires the purely symbolic literals **Cleaned** and **Cooked** as well as actions CLEAN and COOK, which are shown in Figure 4. The top of the dishwasher is used to clean food and cups. The top of the microwave is used to cook food. Objects become transparent when cleaned or cooked.

11.2. Heuristics

We experimented with several versions of FFRob using different heuristics. For all heuristics, as previously described, we automatically generate new samples if the FFRob heuristic is infinite before planning because a finite heuristic value is a necessary condition for feasibility. The following heuristics are compared in the experiments:

1. H_0: The heuristic returns 0.
2. H_{Goals}: This calls UNSATISFIED-GOALS to return the number of unsatisfied goal conditions.
3. H_{FF}: This is the original FF heuristic based only on simple conditions. It explicitly ignores complex conditions, namely *reachability conditions*. This heuristic represents the semantic attachments strategy of Dornhege et al. (2009).
4. H_{MaxRob}: This uses MAX-COMB when performing COMPUTE-COSTS to produce its estimate.
5. H_{AddRob}: This uses ADD-COMB when performing COMPUTE-COSTS to produce its estimate.
6. H_{FFRob}: This finds a relaxed plan using the ADD-COMB version of COMPUTE-COSTS and EXTRACT-RELAXED-PLAN.
7. H_{FFRob}, HA: This finds a relaxed plan using the h_{add} version of COMPUTE-COSTS and EXTRACT-RELAXED-PLAN and computes first goal-achieving helpful actions based on that plan.

We use deferred best-first search, described in Appendix 12, as the search control for our experiments.

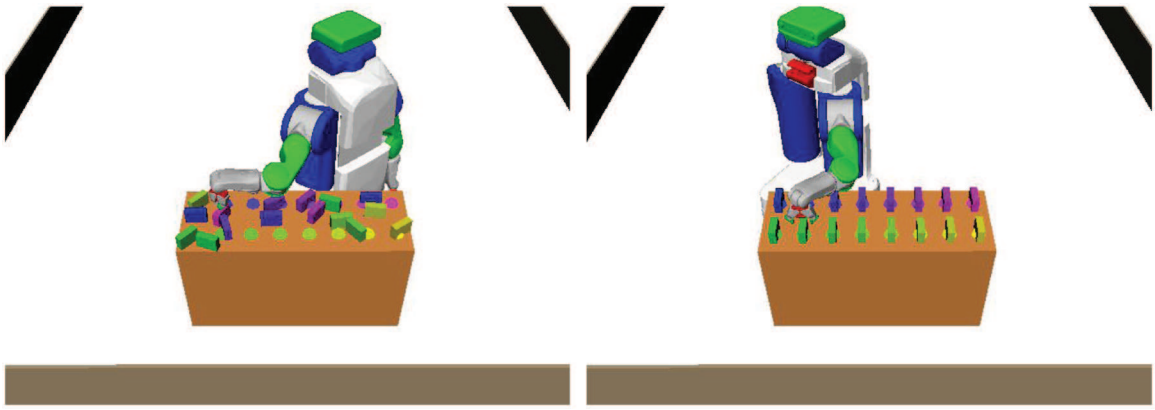

Fig. 21. The second state and last state on a plan for Problem 1–2.

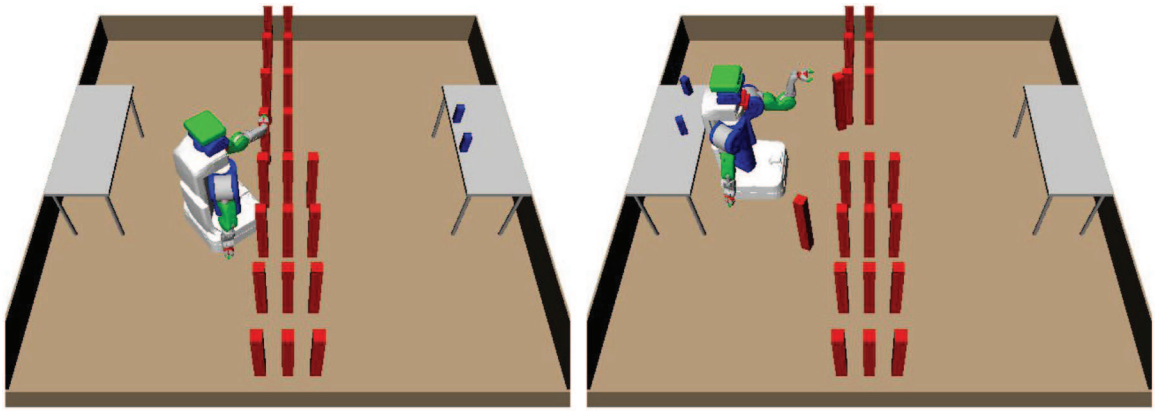

Fig. 22. The second state and last state on a plan for Problem 2–2.

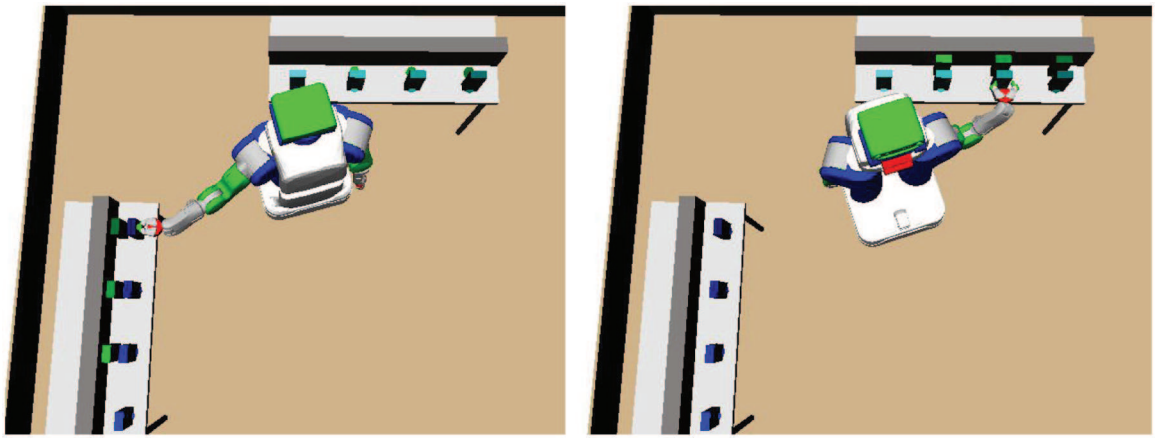

Fig. 23. The second state and last state on a plan for Problem 3-2.

Deferred best-first search typically outperforms standard best-first search because of its lazy evaluation of successors.

11.3. Implementation

We implemented FFRob in Python using the OpenRAVE robotics framework (Diankov and Kuffner, 2008) for simulation. We use a simulated PR2 robot with a mobile base and a single active manipulator. Thus, there are 10 active degrees of freedom. We use Open Dynamics Engine (Smith, 2005) for collision-checking. Additionally, we employ bounding boxes and cache collision results in order to reduce the number of collision checks.

The sampling parameters θ used in our experiments were chosen relatively arbitrarily and are fixed across all problems and heuristics. In practice, we do not increase the sampling parameter sizes upon a sampling failure. We restrict

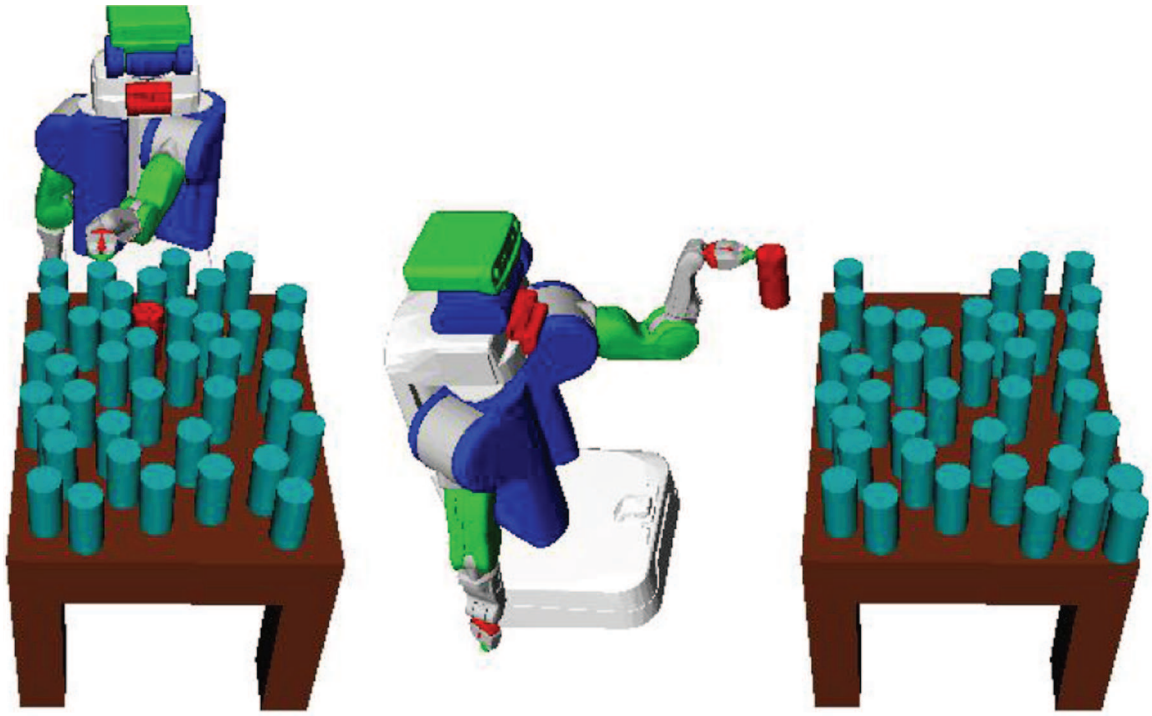

Fig. 24. The second state and last state on a plan for Problem 4.

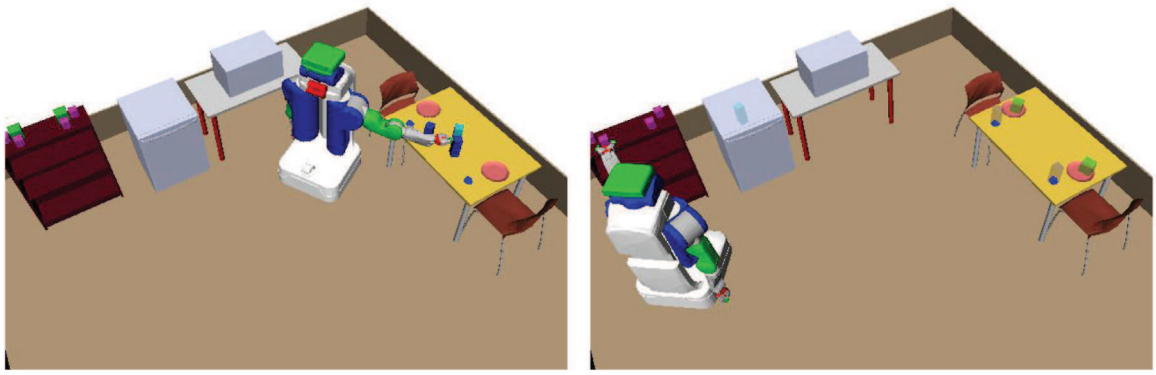

Fig. 25. The second state and last state on a plan for Problem 5.

the robot to four side grasps per objects except on Problems 1–1 & 1–2 where we use a single top grasp.

We randomly sample 25 general poses per object type in addition to 5 poses per specified symbolic region. We bias the general poses to be collision-free in the start state. We attempt to sample a single grasp trajectory per grasp and object pose. We enforce timeouts of 30 iterations for S-PICK-PLACE due to inverse reachability, inverse kinematics, or motion planning failures. We increase this number to 50 for objects that have an explicit goal condition. We reuse placement trajectories for objects with similar geometries. We impose a pruning rule that dynamically limits the number of successor PLACE actions considered that do not achieve a goal to five. This allows a large number of placements to be created for constrained problems without greatly increasing the branching factor.

The approach trajectories sampled using S-APPR-TRAJ control only the robot manipulator, and the roadmap trajectories sampled using S-TRAJ control only the robot base. We use a star-graph CRG for all problems except Problems 2-1 & 2-2 where moveable objects are located on the floor. For these problems the CRG is fixed degree-PRM with a degree of 4, and 50 sampled intermediate roadmap configurations.

11.4. Results

Table 1 displays the results of the experiments. Each problem and algorithm combination was simulated over 50 trials. Each simulation had a timeout of 300 seconds. Simulations were performed on a single Intel Xeon E5 v3 2.5 GHz processor. Each entry in the table reports *success rate* (%) and mean of the *runtimes* (sec) in seconds for

Table 1. Experiment results over 50 trials.

P	H_{Goals}		H_{FF}		H_{MaxRob}		H_{AddRob}		H_{FFRob}		H_{FFRob}, HA	
	%	sec	%	sec	%	sec	%	sec	%	sec	%	sec
1-1	100	5	100	2	0	—	100	4	100	4	100	2
1-2	98	27	96	35	0	—	98	26	100	44	100	23
2-1	40	78	80	73	94	74	98	34	94	33	96	26
2–2	0	—	12	182	46	115	62	91	82	60	76	50
3-1	20	245	92	109	0	—	90	104	88	57	96	19
3-2	0	—	0	—	0	—	0	—	0	—	72	135
4	0	—	0	—	30	97	46	126	78	38	74	31
5	0	—	0	—	0	—	74	56	84	137	76	44

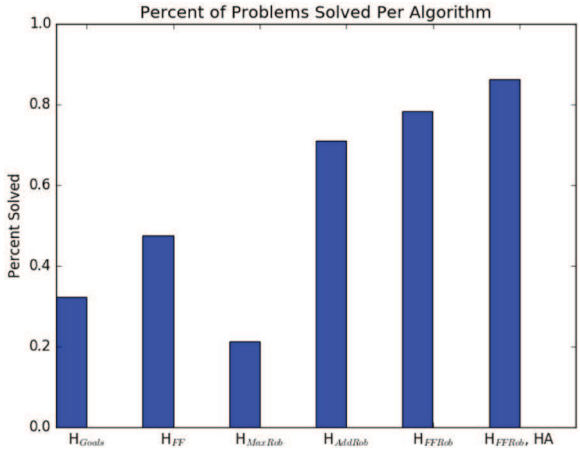

Fig. 26. A bar graph of the overall success rate across all problems per algorithm.

solved instances. The runtimes measure the full algorithm runtime, including the sampling, collision-checking, preprocessing, and discrete planning. We observed that the data often has large runtime outliers causing the mean runtime to be larger than the median runtime. Runtime entries with a dash (-) indicate that the algorithm did not solve any of the simulations for that problem. Figure 26 displays the overall percent of solved instances across all problems for each algorithm. H_0 was unable to solve any of the problems and is excluded from the results.

H_{FFRob}, HA gave the best performance in both success rate and runtime. Helpful actions improved the performance of H_{FFRob}, HA over H_{FFRob}. H_{AddRob} performed worse than H_{Goals} and H_{FF} although it was not strictly worse across all problem instances. In particular, H_{AddRob} performed poorly on Problems 1–1, 1–2, and 3–1 because it requires about the same amount of overhead to evaluate as H_{FFRob} while providing a much less informative heuristic estimate. H_{FF} performed worse than H_{FFRob} indicating that reachability information is vital for producing successor heuristic for geometrically non-trivial task and motion planning problems.

Problem 1–2 can be compared to the *grid @ tabletop (14 objects)* problem of Krontiris and Bekris (2015). All but 1 of our algorithms were able to solve it with an above 95% success ratio in less than 40 s.

The fact that H_{Goals} was able to solve Problems 1-1 and 1-2 indicates that they require little heuristic guidance. Because each object has an explicit goal, H_{Goals} corresponds well with the actual distance to the goal, which is approximately twice H_{Goals}. In the event that the H_{Goals} must move an object more than once along the plan, it will rely on a brute force search to reach a lower heuristic value. Thus, we observed that H_{Goals} required more expansions to solve a problem than H_{FFRob} but was able to perform each expansion more quickly because of the lower heuristic overhead.

Problems 2–1 & 2–2 demonstrate that FFRob can solve both NAMO and traditional pick-and-place problems using the same algorithm. Problems 3–1 & 3–2 proved the most difficult because of their large amount of non-monotonicity. Even in discrete settings, non-monotonicity causes FF to produce poor heuristic estimates because of its delete-relaxation. However, our best algorithm manages to solve Problem 3–2, demonstrating the ability to tackle problems with these elements.

The results of Problem 4 suggest that strong heuristics are particularly necessary in problems with large branching factors resulting from large numbers of movable objects. Problem 5 demonstrates that FFRob is able to quickly solve a long-horizon, real-world problem involving symbolic actions, cluttered environments, and non-monotonic requirements.

For the problems we posed, we noticed that FFRob never performs more than one iteration of sampling and planning. Verifying that the discretized planning problem does not have a solution is computationally expensive because the worst-case complexity grows exponentially in the number of moveable objects. In practice, terminating the search after a finite amount of time to generate new samples will result in better performance. Generally, if the discrete search using H_{FFRob}, HA is given a feasible set of samples, it will identify a solution before a short timeout for non-adversarial problems. Thus, it can be advantageous to restart with a new set of samples even in cases where the original set of samples would have been sufficient.

12. Conclusion

FFRob is a probabilistically complete algorithm for task and motion planning. It uses a EAS, a generalized action representation, to model discretized planning problems with complex conditions. We adapt the FF heuristic to EAS problems in order to provide strong heuristic guidance for these problems. FFRob iteratively discretizes a task and motion planning problem and runs an EAS planner to evaluate if the current set of samples is sufficient for a solution. This leads to a probabilistically complete algorithm as FFRob will, with probability approaching one, generate a set of samples that contains a solution. In our results, we show that including geometric information in a heuristic is critical for the resulting search algorithm to efficiently solve manipulation problems involving interesting geometric constraints. Additionally, we show that a single algorithm can efficiently solve a diverse set of task and motion planning problems.

Future work includes analytically and empirically investigating the quality of solutions returned by FFRob with respect to costs. In Corollary 2, we briefly remark that FFRob, with an optimal implementation SEARCH, will produce a solution no longer than the minimum length robustly feasible solution in finite time. This is the setting where PICK and PLACE actions have unit costs while MOVE actions have zero cost. This can be easily extended to problems in which PICK, PLACE, MOVE, and other actions have arbitrary, non-negative fixed costs. A more interesting case is when MOVE actions have costs dependent on the control effort required to move between their start and end configuration. This requires identifying a class of *robustly optimal* PPM problems and investigating whether sampling-based algorithms are *asymptotically optimal*, i.e. whether they with high probability converge to an optimal solution (Karaman and Frazzoli, 2011). We suspect that FFRob would be asymptotically optimal for a similar set of conditions as those presented in Section 10.

Because FFRob samples continuous values independently of its search, it often produces many unnecessary samples. This is undesirable because sampling can be time intensive and a large number of samples can slow SEARCH. For example, in Figure 24, FFRob samples poses, grasps, configurations, and trajectories for each cylinder on the table, although only a few are required to reach the red cylinder. Future work involves using the planning to guide the sampling such as done by Garrett et al. (2015). Additional future work involves applying FFRob to different manipulation tasks or generally planning domains involving continuous variables. For domains involving stacking, where there are many possible stacking combinations, the combinatorial growth in possible samples may overwhelm FFRob. In which case, a careful sampling strategy would be needed.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the NSF (grant numbers 1122374, 1420927, and 1523767), the ONR (grant number N00014-14-1-0486), and the ARO (grant number W911NF1410433). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

Alami R, Laumond JP and Siméon T (1994) Two manipulation planning algorithms. In: *Workshop on algorithmic foundations of robotics (WAFR 1994)*. Stanford, CA, USA, 12 May 1994, pp. 109–125. Natick, MA, USA: AK Peters, Ltd.

Alami R, Siméon T and Laumond JP (1990) A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps. In: *International symposium of robotic research (ISRR 1990)*. Tokyo, Japan, 28–31 August 1989, pp. 453–463. Cambridge, MA, USA: MIT Press.

Bäckström C and Nebel B (1995) Complexity results for SAS+ planning. *Computational Intelligence* 11: 625–655.

Barry J, Kaelbling LP and Lozano-Pérez T (2013) A hierarchical approach to manipulation with diverse actions. In: *IEEE international conference on robotics and automation (ICRA 2013)*, Karlsruhe, Germany, 6 May 2013, pp. 1799–1806. IEEE Press: Piscataway.

Blum AL and Furst ML (1997) Fast planning through planning graph analysis. *Artificial Intelligence* 90: 281–300.

Bonet B and Geffner H (1999) Planning as heuristic search: New results. In: *5th european conference on planning (ECP)*, Durham, UK, 8–10 September 1999, pp. 360–372. Berlin, Heidelberg: Springer.

Bonet B and Geffner H (2001) Planning as heuristic search. *Artificial Intelligence* 129: 5–33.

Bylander T (1994) The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69: 165–204.

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

Coles A, Coles A, Fox M and Long D (2013) A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research* 46: 343–412.

Dantam NT, Kingston Z, Chaudhuri S and Kavraki LE (2016) Incremental task and motion planning: A constraint-based approach. In: *Robotics: science and systems (RSS 2016)*. Ann Arbor, Michigan, USA, 18–22 June 2016. Available at: http://www.roboticsproceedings.org/rss12/p02.html

de Silva L, Pandey AK, Gharbi M and Alami R (2013) Towards combining HTN planning and geometric task planning. In: *RSS workshop on combined robot motion planning and AI planning for practical applications*. Berkeley, CA, USA, 12–16 July 2014. Available at: https://arxiv.org/abs/1307.1482

Diankov R (2010) *Automated construction of robotic manipulation programs*. PhD Thesis, Robotics Institute, Carnegie Mellon University, USA.

Diankov R and Kuffner J (2008) OpenRAVE: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Carnegie Mellon University, USA, July.

Dogar MR and Srinivasa SS (2012) A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots* 33: 217–236.

Dornhege C, Eyerich P, Keller T, Trüg S, Brenner M and Nebel B (2009) Semantic attachments for domain-independent planning systems. In: *International conference on automated planning and scheduling (ICAPS 2009)*, Thessaloniki, Greece, 19–23 September 2009, pp. 114–121. xxx: AAAI Press. Palo Alto, California, USA:

Dornhege C, Hertle A and Nebel B (2013) Lazy evaluation and subsumption caching for search-based integrated task and motion planning. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS 2013)*. Tokyo, Japan, 3–8 November 2013. IROS Workshop on AI-Based Robotics.

Erdem E, Haspalamutgil K, Palaz C, Patoglu V and 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 (ICRA 2011)*. Shanghai, China, 9 May 2011, pp. 4575–4581. Piscataway, NJ, USA: IEEE

Fikes RE and Nilsson NJ (1971) STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2: 189–208.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2014) FFRob: An efficient heuristic for task and motion planning. In: *Workshop on the algorithmic foundations of robotics (WAFR 2014)*. Istanbul, Turkey, 3–5 August 2014, pp. 179–195. New York, NY, USA: Springer International Publishing.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2015) Backward-forward search for manipulation planning. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS 2015)*. Hamburg, Germany, 28 September 2015, pp. 6366–6373. Piscataway, NJ, USA: IEEE.

Ghallab M, Nau DS and Traverso P (2004) *Automated Planning: Theory and Practice*. Amsterdam, Netherlands: Elsevier.

Gregory P, Long D, Fox M and Beck JC (2012) Planning modulo theories: Extending the planning paradigm. In: *International conference on automated planning and scheduling (ICAPS 2012)*. Atibaia, São Paulo, Brazil, 25–29 June 2012, pp. 65–73. Palo Alto, CA, USA: AAAI.

Hauser K and Latombe J (2009) Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In: *ICAPS Workshop on Bridging the Gap between Task and Motion Planning*. Thessaloniki, Greece, 19–23 September 2009.

Hauser K and Latombe JC (2010) Multi-modal motion planning in non-expansive spaces. *International Journal of Robotics Research* 29: 897–915.

Hauser K and Ng-Thow-Hing V (2011) Randomized multi-modal motion planning for a humanoid robot manipulation task. *International Journal of Robotics Research* 30: 676–698.

Helmert M (2006) The fast downward planning system. *Journal of Artificial Intelligence Research* 26: 191–246.

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

Kaelbling LP and Lozano-Pérez T (2011) Hierarchical planning in the now. In: *IEEE international conference on robotics and automation (ICRA 2011)*. Shanghai, China, 9 May 2011, pp. 1470–1477. Piscataway, NJ, USA: IEEE.

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

Kavraki LE, Kolountzakis MN and Latombe JC (1998) Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation* 14: 166–171.

Kavraki LE and Latombe JC (1998) Probabilistic roadmaps for robot path planning. In: Gupta K and Pobil AP (eds) *Practical Motion Planning in Robotics: Current Approaches and Future Directions*. New York: John Wiley & Sons, pp. 33–53.

Kavraki LE, Latombe JC, Motwani R and Raghavan P (1995) Randomized query processing in robot path planning. In: *27th annual ACM symposium on theory of computing*, Las Vegas, NV, USA, 29 May 1995, pp. 353–362. New York City, NY, USA: ACM Press.

Kavraki LE, Svestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12: 566–580.

King J, Cognetti M and Srinivasa S (2016) Rearrangement planning using object-centric and robot-centric action spaces. In: *IEEE international conference on robotics and automation (ICRA 2016)*. Stockholm, Sweden, 16 May 2016, pp. 3940–3947. Piscataway, NJ, USA: IEEE.

Krontiris A and Bekris KE (2015) Dealing with difficult instances of object rearrangement. In: *Robotics: science and systems (RSS 2015)*, Rome, Italy, 13–17 July 2015. Available at: http://www.roboticsproceedings.org/rss11/p45.html

Krontiris A and Bekris KE (2016) Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In: *IEEE international conference on robotics and automation (ICRA 2016)*, Stockholm, Sweden. 16 May 2016, pp. 3924–3931. Piscataway, NJ, USA: IEEE.

Kuffner JJ Jr and LaValle SM (2000) RRT-Connect: An efficient approach to single-query path planning. In: *IEEE international conference on robotics and automation (ICRA 2000)*. San Francisco, CA USA, 24–28 April 2000, Vol. 2, pp. 995–1001. Piscataway, NJ, USA: IEEE.

Lagriffoul F, Dimitrov D, Bidot J, Saffiotti A and Karlsson L (2014) Efficiently combining task and motion planning using geometric constraints. *International Journal of Robotics Research*: 33: 1726–1747.

Lagriffoul F, Dimitrov D, Saffiotti A and Karlsson L (2012) Constraint propagation on interval bounds for dealing with geometric backtracking. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS 2012)*. Vilamoura, Algarve, Portugal, 7 October 2012, pp. 957–964. Piscataway, NJ, USA: IEEE.

Leven P and Hutchinson S (2002) A framework for real-time path planning in changing environments. *International Journal of Robotics Research* 21: 999–1030.

Lozano-Pérez T (1981) Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man, and Cybernetics* 11: 681–698.

Lozano-Pérez T, Jones JL, Mazer E, O'Donnell PA, Grimson WEL, Tournassoud P et al. (1987) Handey: A robot system that recognizes, plans, and manipulates. In: *IEEE international conference on robotics and automation (ICRA 1987)*. Raleigh, NC, USA, 31 March–3 April 1987, Vol. 4, pp. 843–849. Piscataway, NJ, USA: IEEE.

Lozano-Pérez T and Kaelbling LP (2014) A constraint-based method for solving sequential manipulation planning problems. In: *IEEE/RSJ international conference on intelligent robots*

and systems (IROS 2014), pp. 3684–3691. Piscataway: IEEE Press. Chicago, IL, USA, 14 September 2014, pp. 3684–3691. Piscataway, NJ, USA: IEEE.

McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, Veloso M et al. (1998) PDDL: The planning domain definition language. Technical Report, Yale Center for Computational Vision and Control, USA, October.

Nilsson NJ (1984) Shakey the robot. Technical Report 323, Artificial Intelligence Center, SRI International, USA, April.

Pandey AK, Saut JP, Sidobre D and Alami R (2012) Towards planning human-robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement. In: *RAS/EMBS international conference on biomedical robotics and biomechatronics*. Rome, Italy, 24 June 2012, pp. 1371–1376. Piscataway, NJ, USA: IEEE.

Plaku E and Hager G (2010) Sampling-based motion planning with symbolic, geometric, and differential constraints. In: *IEEE international conference on robotics and automation (ICRA 2010)*. Anchorage, AK USA, 3 May 2010, pp. 5002–5008. Piscataway, NJ, USA: IEEE.

Siméon T, Laumond JP, Cortés J and Sahbani A (2004) Manipulation planning with probabilistic roadmaps. *International Journal of Robotics Research* 23: 729–746.

Smith R (2005) Open dynamics engine. Available at: http://ode.org

Srivastava S, Fang E, Riano L, Chitnis R, Russell S and Abbeel P (2014). Combined task and motion planning through an extensible planner-independent interface layer. In: *IEEE international conference on robotics and automation (ICRA 2014)* Hong Kong, China, 31 May 2014, pp. 639–646. Piscataway, NJ, USA: IEEE.

Stilman M and Kuffner JJ (2006) Planning among movable obstacles with artificial constraints. *International Journal of Robotics Research* 11–12: 1295–1307.

Stilman M, Schamburek JU, Kuffner JJ and Asfour T (2007) Manipulation planning among movable obstacles. In: *IEEE international conference on robotics and automation (ICRA 2007)*. Rome, Italy, 10 April 2007, pp. 3327–3332. Piscataway, NJ, USA: IEEE.

Toussaint M (2015) Logic-geometric programming: An optimization-based approach to combined task and motion planning. In: *AAAI conference on artificial intelligence*, Austin, Texas, USA, 24 June 2015, pp. 1930–1936. Palo Alto, California, USA: AAAI Press.

Van Den Berg J, Stilman M, Kuffner J, Lin M and Manocha D (2009) Path planning among movable obstacles: A probabilistically complete approach. In: Chirikjian GS, Choset H, Morales M and Murphey T (eds) *Algorithmic Foundation of Robotics VIII*. New York: Springer, pp. 599–614.

Wilfong GT (1988) Motion planning in the presence of movable obstacles. In: *Symposium on computational geometry*, pp. 279–288. Urbana-Champaign, IL, USA, 6 June 1988, pp. 279—288. New York City, NY, USA: ACM.

Appendix A: Search

Best-first search

Best-first, shown in Figure 27, search extracts the element in the queue that minimizes a cost function $f(n)$.

BFS-EXTRACT(Q, f)
1 **return** POP-MIN(Q, f)

BFS-PROCESS(Q, s', n; H)
1 PUSH(Q, STATEN($s', n.g + 1$, H(s'), n))

Fig. 27. Best-first search extract and process procedures.

DBFS-PROCESS(Q, s', n; H)
1 PUSH(Q, STATEN($s', n.g + 1$, H($\underline{n.s}$), n))

Fig. 28. Deferred best-first search process procedure.

Common cost functions include the path cost $f(n) \equiv n.g$ (Dijkstra's algorithm), the sum of path cost and heuristic cost (A^*), a weighted sum of path cost and heuristic cost $f(n; w) \equiv (1 - w)\, n.g + (w)\, n.h$ (weighted A^*), and the heuristic cost $f(n) \equiv n.h$ (greedy best-first).

Deferred best-first search

Deferred best-first search (also called *lazy greedy search*) is a variant of standard best-first search (Helmert, 2006). It defers the evaluation of successors states until they are extracted from the queue in order to reduce the number of heuristic evaluations. Successor states temporarily use their parent's heuristic cost while in the queue. The intuition behind this approach is that successors of a state s are often added to the queue in an order given by helpful actions such that states believed to be closer to the goal are processed first. If a extracted state s' has a low heuristic cost, its own successors will temporarily have that cost when they are added to the queue. Thus, the full set of successors of s will likely not be processed because the search greedily proceeds down the subtree rooted at s'. The EXTRACT procedure is the same for standard best-first search but the PROCESS procedure in Figure 28 is slightly modified to use the parent state's heuristic cost.

Appendix B: Review of sPRM analysis

Proof of Theorem 1

Theorem 5. *For any robustly feasible motion planning problem, there exists a sequence of $k + 1$, where $k = \left\lceil \frac{2L}{\delta} \right\rceil$, d-spheres $(B_0, B_1, ..., B_k)$ centered at $\tau(Li/k)$ for $i \in \{0, ..., k\}$, each with radius $\delta/2$, such that any trajectory τ' linearly interpolated from $(q^0, q_0, q_1, ..., q_k, q^*)$, where $q_i \in B_i$ for $i \in \{0, ..., k\}$, is a collision-free solution to the motion planning problem.*

Proof. First consider the following lemma.

Lemma 2. $B_{i+1} \subseteq B(\tau(Li/k), \delta)$.

Proof. By our construction, using $x \leq \lceil x \rceil$

$$\begin{aligned} ||\tau(L(i+1)/k) - \tau(Li/k)|| &\leq L(i+1)/k - Li/k && (7)\\ &= L/k && (8)\\ &\leq L/(2L/\delta) && (9)\\ &= \delta/2 && (10) \end{aligned}$$

Now for any $q_{i+1} \in B_{i+1}$, by the triangle inequality,

$$\begin{aligned} ||q_{i+1} - \tau(Li/k)|| &\leq ||q_{i+1} - \tau(L(i+1)/k)|| && (11)\\ &\quad + ||\tau(L(i+1)/k) - \tau(Li/k)|| \\ &\leq \delta/2 + \delta/2 && (12)\\ &= \delta && (13) \end{aligned}$$

Thus, each q_{i+1} is contained in $B(\tau(Li/k), \delta)$ which implies $B_{i+1} \subseteq B(\tau(Li/k), \delta)$. □

Because τ has at least δ clearance from obstacles, all points within $B(\tau(t), \delta)$ for $t \in [0, L]$ are collision-free. Moreover, line segments between any two points in $B(\tau(t), \delta)$ for $t \in [0, L]$ are collision-free because they are contained in $B(\tau(t), \delta)$ by convexity of the d-sphere. Using Lemma 2, the line segment between any $q_i \in B_i$ and any $q_{i+1} \in B_{i+1}$ is collision-free because both q_i and q_{i+1} are contained within $B(\tau(Li/k), \delta)$. By applying this to all $i, i+1$, we see that the linearly interpolated trajectory $(q_0, q_1, q_2, ..., q_k)$ is thus collision-free. Finally, each segment from q^0 to $q_0 \in B_0$ or from q^* to $q_k \in B_k$ is collision-free by the problem being robustly feasible. So, the linearly interpolated trajectory $(q^0, q_0, q_1, q_2, ..., q_k, q^*)$ is also collision-free. □