FFRob: An efficient heuristic for task and motion planning

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

MIT CSAIL

Abstract. Manipulation problems involving many objects present substantial challenges for motion planning algorithms due to the high dimensionality and multi-modality of the search space. Symbolic task planners can efficiently construct plans involving many entities but cannot incorporate the constraints from geometry and kinematics. In this paper, we show how to extend the heuristic ideas from one of the most successful symbolic planners in recent years, the FastForward (FF) planner, to motion planning, and to compute it efficiently. We use a multi-query roadmap structure that can be conditionalized to model different placements of movable objects. The resulting tightly integrated planner is simple and performs efficiently in a collection of tasks involving manipulation of many objects.

1 Introduction

Mobile manipulation robots are physically capable of solving complex problems involving moving many objects to achieve an ultimate goal. Mobile bases with one or more arms are becoming available and increasingly affordable while RGBD sensors are providing unprecedented sensory bandwidth and accuracy. However, these new capabilities are placing an increasing strain on existing methods for programming robots. Traditional motion-planning algorithms that find paths between fully specified configurations cannot address problems in which the configuration space of interest is not just that of the robot but the configuration space of a kitchen, for example, and the goal is to make dinner and clean the kitchen. We almost certainly do not want to choose whether to get the frying pan or the steak next by sampling configurations of the robot and kitchen and testing for paths between them.

Researchers in artificial intelligence planning have been tackling problems that require long sequences of actions and large discrete state spaces and have had some notable success in recent years. However, these symbolic "task-level" planners do not naturally encompass the detailed geometric and kinematic considerations that motion planning requires. The original Shakey/STRIPS robot system [1, 2], 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. This is not generally possible in realistic manipulation domains because the geometrical and kinematic constraints are significant.

Consider a simple table-top 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, or use them to make a meal, or put them away in their storage bins. The basic robot operations are to pick up an object and place it somewhere else; in addition, the robot can move its base in order to reach a distant object. Note that, in general, to reach some object, we 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 this connectivity with a set of static assertions updated by STRIPS operators; determining how the connectivity of the underlying free space changes requires geometric computation.

A natural extension to the classic symbolic planning paradigm is to introduce "computed predicates" (also know 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. A motion planner can serve to implement such a predicate, determining the reachability of one configuration from another. This approach is currently being pursued, for example, by Dornhege et al. [3, 4], as a way of combining symbolic task-level planners with motion planners to get a planner that can exploit the abstraction strengths of the first and the geometric strengths of the second. 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, and, very importantly, to avoid calling the motion planner during heuristic evaluation. Considering only a sparse set of placements may limit the generality of the planner, while avoiding calling the motion planner in the heuristic leads to a heuristic that is uninformed about geometric considerations and may result in considerable inefficiency due to backtracking during the search for a plan.

An alternative approach to integrating task and motion planning has been to start with a motion planner and use a symbolic planner to provide heuristic guidance to the motion planner, for example in the work of Cambon et al. [5]. However, since the task-level planner is ignoring geometry, its value as a heuristic is quite limited.

In this paper we show how to obtain a fully integrated task and motion planner using a search in which the heuristic takes geometric information into account. We show an extension of the heuristic used in the FastForward (FF) [6] planning system to the FFRob heuristic, which integrates reachability in the robot configuration space with reachability in the symbolic state space. Both the search and the computation of the FFRob heuristic exploit a roadmap [7]

data structure that allows multiple motion-planning queries on the closely related problems that arise during the search to be solved efficiently.

2 Related work

There have been a number of approaches to integrated task and motion planning in recent years. The pioneering Asymov system of Cambon et al. [5] 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 to guide the motion planning search. The work of Plaku and Hager [8] is similar in approach.

The work of Erdem et al. [9], is similar in approach to Dornhege et al. [3], augmenting a task planner that is based on explicit causal reasoning with the ability to check for the existence of paths for the robot.

Pandey et al. [10] and deSilva et al. [11] use 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. [3]. 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.

Lagriffoul et al. [12] also integrate the symbolic and geometric search. 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 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.

In the HPN approach of Kaelbling and Lozano-Pérez [13], 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. This type of backward chaining to identify relevant actions is also present in work on *navigation among movable obstacles*. The work of Stilman et al. [14, 15] also plans backwards from the final goal and uses swept volumes to determine, recursively, which additional objects must be moved and to constrain the system from placing other objects into those volumes.

Srivastava et al. [16, 17] offer a novel control structure that avoids computing expensive precondition values in many cases by assuming a favorable default valuation of the precondition 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 operator. In that case, symbolic planning is repeated. This approach requires the ability to diagnose why a motion plan is not possible in a given state, which can be challenging, in general. Empirically,

their approach is the only one of which we are aware whose performance is competitive with our FFRob method.

All of these approaches, although they have varying degrees of integration of the symbolic and geometric planning, generally lack a true integrated heuristic that allows the geometric details to affect the focus of the symbolic planning. In this paper, we develop such a heuristic, provide methods for computing it efficiently, and show that it results in a significant computational savings.

3 Problem formulation

When we seek to apply the techniques of symbolic planning to domains that involve robot motions, object poses and grasps, we are confronted with a series of technical problems. In this section, we begin by discussing those problems and our solutions to them, and end with a formal problem specification.

We might naturally wish to encode robot operations that pick up and place objects in the style of traditional AI planning operator descriptions such as:

PICK(C_1, O, G, P, C_2):
pre: *HandEmpty*, $Pose(O, P)$, $Robotconf(C_1)$, $CanGrasp(O, P, G, C_2)$, $Reachable(C_1, C_2)$
add: $Holding(O, G)$, $RobotConf(C_2)$
delete: *HandEmpty*, $RobotConf(C_1)$

PLACE(C_1, O, G, P, C_2):
pre: $Holding(O, G)$, $Robotconf(C_1)$, $CanGrasp(O, P, G, C_2)$, $Reachable(C_1, C_2)$
add: *HandEmpty*, $Pose(O, P)$, $RobotConf(C_2)$
delete: $Holding(O, G)$, $RobotConf(C_1)$

In these operations, the C, P, and G variables range over robot configurations, object poses, and grasps, respectively. These are high-dimensional continuous quantities, which means that there are infinitely many possible instantiations of each of these operators. We address this problem by sampling finitely many values for each of these variable domains during a pre-processing phase. The sampling is problem-driven, but may turn out to be inadequate to support a solution. If this happens, it is possible to add samples and re-attempt planning, although that was not done in the empirical results reported in this paper.

Even with finite domains for all the variables, there is a difficulty with explicitly listing all of the positive and negative effects of each operation. The operations of picking up or placing an object may affect a large number of *Reachable* literals: picking up an object changes the "shape" of the robot and therefore what configurations it may move between; placing an object changes the free configuration space of the robot. Even more significant, which *Reachable* literals are affected can depend on the poses of all the other objects (for example, removing any one or two of three obstacles may not render a configuration beyond the obstacles reachable). Encoding this conditional effect structure in typical form in the preconditions of the operators would essentially require us to write one operator description for each possible configuration of movable objects.

We address this problem by maintaining a state representation that consists of both a list of true literals and a data structure, called *details*, that captures the geometric state in a way that allows the truth value of any of those literals to be computed on demand. This is a version of the *semantic attachments* strategy [3].

The last difficulty is in computing the answers to queries in the details, especially about reachability, which requires finding free paths between robot configurations in the context of many different configurations of the objects. We address this problem by using a conditional *roadmap* data structure called a *conditional reachability graph*, related to a PRM [7], for answering all reachability queries, and lazily computing answers on demand and caching results to speed future queries.

More formally, a *state* is a tuple $\langle L, D \rangle$, where L is a set of literals and D is a domain-dependent detailed representation. A *literal* is a predicate applied to arguments, which may optionally have an attached *test*, which maps the arguments and state into a Boolean value. A literal *holds* in a state if it is explicitly represented in the state's literal set, or its test evaluates to true in the state:

$$\textsc{holds}(l, s) \equiv l \in s.L \text{ or } l.test(s) \ .$$

A *goal* is a set of literals; a state *satisfies* a goal if all of the literals in the goal hold in the state:

$$\textsc{satisfies}(s, \Gamma) \equiv \forall l \in \Gamma.\ \textsc{holds}(l, s) \ .$$

An *operator* is a tuple $\langle \phi, e_{pos}, e_{neg}, f \rangle$ where ϕ is a set of literals representing a conjunctive precondition, e_{pos} is a set of literals to be added to the resulting state, e_{neg} is a set of literals to be deleted from the resulting state, and f is a function that maps the detailed state from before the operator is executed to the detailed state afterwards. Thus, the successor of state s under operator a is defined

$$\textsc{successor}(s, a) \equiv \langle s.L \cup a.e_{pos} \setminus a.e_{neg}, a.f(s) \rangle \ .$$

An operator is *applicable* in a state if all of its preconditions hold in that state:

$$\textsc{applicable}(a, s) \equiv \forall l \in a.\phi.\ \textsc{holds}(l, \phi) \ .$$

An *operator schema* is an operator with typed variables, standing for the set of operators arising from all instantiations of the variables over the appropriate type domains.

Our general formulation has broader applicability, but in this paper we restrict our attention to a concrete domain in which a mobile-manipulation robot can move, grasp rigid objects, and place them on a surface. To formalize this domain, we use literals of the following forms:

- $RobotConf(C)$: the robot is in configuration C, where C is a specification of the pose of the base as well as joint angles of the arm;
- $Pose(O, P)$: object O is at pose P, where P is a four-dimensional pose (x, y, z, θ), assuming that the object is resting on a stable face on a horizontal surface;

- *Holding*(O, G): the robot is holding object O with grasp G, where G specifies a transform between the robot's hand and the object;
- *HandEmpty*: the robot is not holding any object;
- *In*(O, R): the object O is placed in such a way that it is completely contained in a region of space R; and
- *Reachable*(C_1, C_2): there is a collision-free path between robot configurations C_1 and C_2, considering the positions of all fixed and movable objects as well as any object the robot might be holding and the grasp in which it is held.

The details of a state consist of the configuration of the robot, the poses of all the objects, and what object is being held in what grasp.

Two of these literals have tests. The first, *In*, has a simple geometric test, to see if object O, at the pose specified in this state, is completely contained in region R. The test for *Reachable* is more difficult to compute; it will be the subject of the next section.

4 Conditional reachability graph

In the mobile manipulation domain, the details contain a *conditional reachability graph* (CRG), which 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 is similar in spirit to the roadmaps of Leven and Hutchinson [18] in that it is designed to support solving multiple motion-planning queries in closely related environments. The CRG has three components:

- **Poses**: For each object o, a set of possible stable poses.
- **Nodes**: A set of robot configurations, c_i, each annotated with a (possibly empty) set $\{\langle g, o, p\rangle\}$ where g is a grasp, o an object, and p a pose, meaning that if the robot is at the configuration c_i, and object o is at pose p, then the robot's hand will be related to the object by the transform associated with grasp g.
- **Edges**: A set of pairs of nodes, with configurations c_1 and c_2, annotated with an initially empty set of *validation* conditions of the form $\langle h, g, o, p, b\rangle$, where b is a Boolean value that is TRUE if the robot moving from c_1 to c_2 along a simple path (using linear interpolation or some other fixed interpolator) while holding object h in grasp g *will not* collide with object o if it is placed at pose p, and FALSE otherwise.

The validation conditions on the edges are not pre-computed; they will be computed lazily, on demand, and cached in this data structure. Note that some of the collision-checking to compute the annotations can be shared, e.g. the same robot base location may be used for multiple configurations and grasps.

Constructing the CRG The CRG is initialized in a pre-processing phase, which concentrates on obtaining a useful set of sampled object poses and robot configurations. Object poses are useful if they are initial poses, or satisfy a goal

condition, or provide places to put objects out of the way. Robot configurations are useful if they allow objects, when placed in useful poses, to be grasped (and thus either picked from or placed at those poses) or if they enable connections to other useful poses via direct paths. We assume that the following components are specified: a workspace W, which is a volume of space that the robot must remain inside; a placement region T, which is a set of static planar surfaces upon which objects may be placed (such as tables and floor, but not (for now) the tops of other objects); a set $\mathcal{O}_f$ of fixed (immovable) objects; a set $\mathcal{O}_m$ of movable objects; and a vector of parameters θ that specify the size of the CRG. It depends, in addition, on the start state s and goal Γ. We assume that each object $o \in \mathcal{O}_m$ has been annotated with a set of feasible grasps. The parameter vector consists of a number n_p of desired sample poses per object (type); a number n_{ik} of grasp configurations per grasp; a number n_n of configurations near each grasp configuration; a number n_c of RRT iterations for connecting configurations, and a number k specifying a desired degree of connectivity.

The ConstructCRG procedure is outlined below.

ConstructCRG$(W, T, s, \Gamma, \mathcal{O}_f, \mathcal{O}_m, \theta)$:

```
1  N = {s.details.robotConf} ∪ {robot configuration in Γ}
2  for o ∈ O_m:
3      P_o = {s.details.pose(o)} ∪ {pose of o in Γ}
4      for i ∈ {1, ..., θ.n_p}:
5          P_o.add(sampleObjPose(o.shape, T))
6          for g ∈ o.grasps:
7              for j ∈ {1, ..., θ.n_ik}: N.add(sampleIK(g, o, p), (g, o, p))
8              for j ∈ {1, ..., θ.n_n}: N.add(sampleConfNear(g, ( )))
9  E = { }
10 for n_1 ∈ N:
11     for n_2 ∈ NearestNeighbors(n_1, k, N):
12         if CFreePath(n_1.c, n_2.c, O_f): E.add(n_1, n_2)
13 N, E = connectTrees(N, E, W, θ.n_c)
14 return ⟨P, N, E⟩
```

We begin by initializing the set of nodes N to contain the initial robot configuration and the configuration specified in the goal, if any. Then, for each object, we generate a set of sample poses, including its initial pose and goal pose, if any, as well as poses sampled on the object placement surfaces. For each object pose and possible grasp of the object, we use the sampleIK procedure to sample one or more robot configurations that satisfy the kinematic constraints that the object be grasped. We sample additional configurations with the hand near the grasp configuration to aid maneuvering among the objects. We then add edges between the k nearest neighbors of each configuration, if a path generated by linear interpolation or another simple fixed interpolator is free of collisions with fixed objects. At this point we generally have a forest of trees of configurations. Finally, we attempt to connect the trees using an RRT algorithm as in the sampling-based roadmap of trees [19].

To test whether this set of poses and configurations is plausible, we use it to compute a heuristic value of the starting state, as described in section 5. 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.

Querying the CRG Now that we have a CRG we can use it to compute the test for the *Reachable* literal, as shown in REACHABLETEST below.

REACHABLETEST$(c_1, c_2, D, \text{CRG})$:

```
1  for (o, p) ∈ D.objects:
2      for e ∈ CRG.E:
3          if not ⟨D.heldObj, D.grasp, o, p, ∗⟩ ∈ e.valid:
4              p = CFREEPATH(e.n1.c, e.n2.c, o@p, D.heldObj, D.grasp)
5              e.valid.add(⟨D.heldObj, D.grasp, o, p, (p != None)⟩)
6  G = {e ∈ CRG.E | ∀(o, p) ∈ D.objects. ⟨D.heldObj, D.grasp, o, p, True⟩ ∈ e.valid}
7  return REACHABLEINGRAPH(c1, c2, G)
```

The main part of the test is in lines 6–7: we construct a subgraph of the CRG that consists only of the edges that are valid given the object that the robot is holding and the current placements of the movable objects and search in that graph to see if configuration c_2 is reachable from c_1. Lines 1–5 check to be sure that the relevant validity conditions have been computed and computes them if they have not. The procedure CFREEPATH$(c_1, c_2, obst, o, g)$ performs collision checking on a straight-line, or other simply interpolated path, between configurations c_1 and c_2, with a single obstacle *obst* and object o held in grasp g.

In addition, the CRG is used to implement APPLICABLEOPS(s, Ω, CRG), which efficiently determines which operator schema instances in Ω are applicable in a given state s. For each schema, we begin by binding variables that have preconditions specifying the robot configuration, object poses, the currently grasped object and/or the grasp to their values in state s. We consider all bindings of variables referring to objects that are not being grasped. For a *pick* operation, P is specified in the current state, so we consider all bindings of G and C_2 such that $(C_2, (G, O, P)) \in \text{CRG}.N$. For a *place* operation, G is specified in the current state, so we consider all bindings of P and C_2 such that $(C_2, (G, O, P)) \in \text{CRG}.N$.

5 Planning algorithms

A *planning problem*, Π, is specified by $\langle s, \Gamma, \mathcal{O}, T, W, \Omega \rangle$, where s is the initial state, including literals and details, Γ is the goal, $\mathcal{O}$ is a set of objects, T is a set of placement surfaces, W is the workspace volume, and Ω is a set of operator schemas.

PLAN, shown below, is a generic heuristic search procedure. Depending on the behavior of the EXTRACT procedure, it can implement any standard search control structure, including depth-first, breadth-first, uniform cost, best-first, A^*, and hill-climbing. Critical to many of these strategies is a heuristic function,

which maps a state in the search to an estimate of the cost to reach a goal state from that state. Many modern domain-independent search heuristics are based on a *relaxed plan graph* (RPG). In the following section, we show how to use the CRG to compute the relaxed plan graph efficiently.

```
PLAN(Π, EXTRACT, HEURISTIC, θ)
 1  ⟨s, Γ, O, T, W, Ω⟩ = Π
 2  CRG = CONSTRUCTCRG(W, T, s, Γ, O, θ)
 3  def H(s): HEURISTIC(RPG(s, Γ, CRG, Ω))
 4  q = QUEUE(SEARCHNODE(s, 0, H(s), None))
 5  while not q.empty():
 6      n = EXTRACT(q)
 7      if SATISFIES(n.s, Γ): return n.path
 8      for a ∈ APPLICABLEOPS(n.s, Ω, CRG):
 9          s' = SUCCESSOR(n.s, a)
10          q.push(SEARCHNODE(s', n.cost + 1, H(s'), n))
```

Computing the relaxed plan graph In classical symbolic planning, a *plan graph* is a sequence of alternating *layers* of literals and actions. The first layer consists of all literals that are true in the starting state. Action layer i contains all operators whose preconditions are present and simultaneously achievable in literal layer i. Literal layer $i+1$ contains all literals that are possibly achievable after i actions, together with a network of *mutual exclusion* relations that indicates in which combinations those literals might possibly be true. This graph is the basis for *GraphPlan* [20] and related planning algorithms.

The *relaxed plan graph* is a simplified plan graph, without mutual exclusion conditions; it is constructed by ignoring the negative effects of the actions. From the RPG, many heuristics can be computed. For example, the H_{Add} heuristic [21] returns the sum of the levels at which each of the literals in the goal appears. It is optimistic, in the sense that if the mutual exclusion conditions 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 fluents might be "shared." An admissible heuristic, H_{Max} [21], is obtained by taking the maximum of the levels of the goal literals, rather than the sum; but it is found in practice to offer weaker guidance. An alternative is the FF heuristic [6], which performs an efficient backward-chaining pass in the plan graph to determine how many actions, if they could be performed in parallel without deletions, would be necessary to achieve the goal and uses that as the heuristic value. An important advantage of the FF heuristic is that it does not over-count actions if one action achieves multiple effects, and it enables additional heuristic strategies that are based on *helpful actions*. We use a version of the helpful-action strategy that reduces the choice of the next action to those that are in the first level of the relaxed plan, and find that it improves search performance.

In order to use heuristics derived from the RPG we have to show how it can be efficiently computed when the add lists of the operators are incomplete and the

truth values of some literals are computed from the CRG in the details. We present a method for computing the RPG that is specialized for mobile manipulation problems. It constitutes a further relaxation of the RPG which allows literals to appear earlier in the structure than they would in an RPG for a traditional symbolic domain. This is necessary, because the highly conditional effects of actions on *Reachable* literals makes them intractable to compute exactly. The consequence of the further relaxation is that the H_{Add} and H_{Max} heuristics computed from this structure have less heuristic force. However, in section 5 we describe a method for computing a version of H_{FF} that recovers the effectiveness of the original.

The intuition behind this computation is that, as we move forward in computing the plan graph, we consider the positive results of all possible actions to be available. In terms of reachability, we are removing geometric constraints from the details; we do so 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. Recall that, in APPLICABLE and SATISFIES, the HOLDS procedure is used to see if a literal is true in a state. It first tests to see if it is contained in the literal set of the state; this set becomes increasingly larger as the RPG is computed. If the literal is not there, then it is tested with respect to the CRG in the details, which becomes increasingly less constrained as objects are removed.

Importantly, since the geometric tests on the CRG are cached, the worst-case number of geometric tests for planning with and without the heuristic is the same. In practice, computing the RPG for the heuristic is quite fast, and using it substantially reduces the number of states that need to be explored.

RELAXEDPLANGRAPH, shown below, outlines the algorithm in more detail. In the second part of line 1, in a standard implementation we would generate all possible instantiations of all actions. However, because of the special properties of reachability, we are able to abstract away from the particular configuration the robot is in when an action occurs; thus, we consider all possible bindings of the non-configuration variables in each operator, but we only consider binding the starting configuration variable to the actual current starting configuration and leave the resulting configuration variable free. In line 2, we initialize *hState*, which is a pseudo-state containing all literals that are possibly true at the layer we are operating on, and a set of details that specifies which objects remain as constraints on the robot's motion at this layer. In line 6, we ask whether a operator schema with all but the resulting configuration variable bound is applicable in the heuristic state. We only seek a single resulting configuration that satisfies the preconditions of *op* in *hState*; even though many such configurations might exist, each of them will ultimately affect the resulting *hState* in the same way. Lines 7–9 constitute the standard computation of the RPG. In lines 10–11 we perform domain-specific updates to the detailed world model: if there is any way to pick up an object, then we assume it is completely removed from the domain for the rest of the computation of the RPG; if there is any way to put down the currently held object, then we assume that there is no object in the hand, when

doing any further computations of reachability in the CRG. Line 14 creates a new *hState*, which consists of all literals possibly achievable up to this level and the details with possibly more objects removed.

RelaxedPlanGraph(s, Γ, CRG, Ω) :

```
1  D = s.D; ops = allNonConfBindings(Ω)
2  literals = [ ] ; actions = [ ] ; hState = s
3  while True
4      layerActions = { } ; layerLiterals = { }
5      for op ∈ ops:
6          if applicable(op, hState):
7              layerActions.add(op)
8              layerLiterals.union(op.e_pos)
9              ops.remove(op)
10             if op.type = pick: D.objects.remove(op.obj)
11             if op.type = place: D.heldObj = None
12     literals.append(layerLiterals)
13     actions.append(layerActions)
14     hState = ⟨⋃_i literals_i, D⟩
15     if satisfies(hState, Γ): return (literals, actions)
16     if layerActions = { }: return None
```

There is one last consideration: the strategy shown above does not make the dependencies of *Reachable* literals at level i on actions from level $i-1$ explicit; the truth of those literals is encoded implicitly in the details of the *hState*. We employ a simple bookkeeping strategy to maintain a causal connection between actions and literals, which will enable a modified version of the FF heuristic to perform the backward pass to find a parallel plan. We observe that, in the relaxed plan, once an object is *pick*ed, it is effectively removed from the domain. So, we add an extra positive effect literal, *Picked*(o) to the positive effects set of the *pick* action, just when it is used in the heuristic computation.

The FFRob heuristic The FF heuristic operates by extracting a *relaxed plan* from the RPG and returning the number of actions it contains. A relaxed plan $\mathcal{P}$ constructed for starting state s and set of goal literals G consists of a set of actions that has the following properties: (1) For each literal $l \in G$ there is an action $a \in \mathcal{P}$ such that $l \in a.e_{pos}$ and (2) For each action $a \in \mathcal{P}$ and each literal $l \in a.\phi$, either $l \in s$ or there exists an action $a' \in \mathcal{P}$ such that $l \in a.e_{pos}$.

That is, the set of actions in the relaxed plan collectively achieve the goal as well as all of the preconditions of the actions in the set that are not satisfied in the initial state. It would be ideal to find the shortest linear plan that satisfied these conditions, however that is NP-hard [6]. Instead, the plan extraction procedure works backwards, starting with the set of literals in the goal G. For each literal $l \in G$, it seeks the "cheapest" action a^* that can achieve it; that is,

$$a^* = \arg \min_{\{a | l \in a.e_{pos}\}} \sum_{l \in a.\phi} \mathcal{L}(l) \ ,$$

where $\mathcal{L}(l)$ is the index of the lowest layer containing l (which is itself a quick estimate of the difficulty of achieving l.)

The minimizing a^* is added to the relaxed plan, l and any other literals achieved by a^* are removed from the goal set, and the preconditions $a^*.\phi$ are added to the goal set unless they are contained in s. This process continues until the goal set is empty.

The RPG computed as in section 5 does not immediately support this computation, because the *Picked* fluents that are positive results of *Pick* actions do not match the *Reachable* fluents that appear in preconditions. In general, there may be many ways to render a robot configuration reachable, by removing different combinations of obstacles. Determining the smallest such set is known as the *minimum constraint removal* problem [22]. Hauser shows it is NP-Hard in the discrete case and provides a greedy algorithm that is optimal if obstacles must not be entered more than once. We have extended this method to handle the case in which objects are *weighted*; in our case, by the level in the RPG at which they can be picked. The weighted MCR algorithm attempts to find a set of obstacles with a minimal sum of weights that makes a configuration reachable.

So, any action precondition of the form *Reachable*(c) is replaced by the set of preconditions *Picked*(o) for all objects o in the solution to the weighted MCR problem for configuration c. This represents the (approximately) least cost way to make c accessible. Having carried out this step, we can use the standard FF method for extracting a relaxed plan. The FFRob heuristic returns the number of actions in this relaxed plan.

Geometric biases It frequently happens that multiple states have the same heuristic value; in such cases, we break ties using geometric biases. These three biases do not affect the overall correctness or completeness of the algorithm. Intuitively, the idea is to select actions that maximize the reachability of configurations in the domain from the current state.

- Choose actions that leave the largest number of configurations corresponding to placements of objects in their goal poses or regions available. This captures the idea that blocking goal regions should be avoided if possible. This is useful because although a heuristic will report when a placement is immediately bad, i.e., already blocking future goals, it will not convey information that the placement may prevent two necessary placements later in the search because it was out in the open. This is because the relaxed plan assumed that a free placement exists, despite objects being placed there, because it does not model negative effects of actions.
- Choose actions that leave the largest total number of configurations corresponding to placements reachable; this ensures that all placements are as tight as possible against the edge of the reachable space.
- If neither of the previous biases breaks the tie, then select actions that maximize the total number of reachable configurations.

These biases experimentally prove to be helpful in giving the search additional guidance in this domain, especially in combination with enforced hill climbing search, which lacks backtracking to undo bad decisions.

6 Results

We have experimented with various versions of this algorithm, differing in the definition of the heuristic, on a variety of tasks; we report the results in this section.

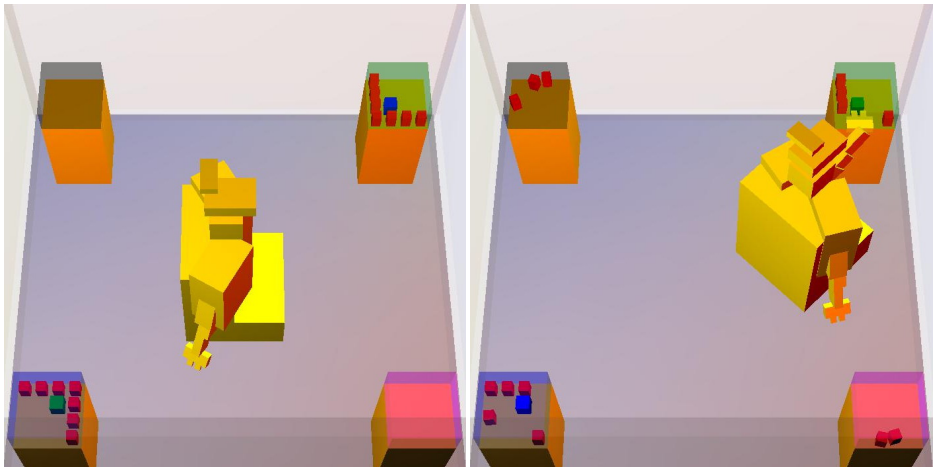
The search strategy in all of our experiments is enforced hill-climbing [6], in which a single path through the state space is explored, always moving to the unvisited successor state with the smallest heuristic value, with ties broken using geometric biases. This search strategy is known not to be complete, but we have found it to be very effective in our domains. If the hill-climbing search were to reach a dead end, one could restart the search (as is done in FastForward), using the best-first strategy or weighted A^*, which are complete. However, even with a complete search and no helpful-action heuristic, the overall planner is not probabilistically complete, since it is limited to the initial set of sample poses and configurations.

The parameters governing the creation of the CRG are: $n_p \in [25 - 50]$ (the number of placements for each object); this varies with the size of the placement regions; $n_{ik} = 1$ (number of robot configurations for each grasp); $n_n = 1$ (number of additional robot configurations near each grasp); $n_c = 250$ (number of RRT iterations); $k = 4$ (number of nearest neighbors).

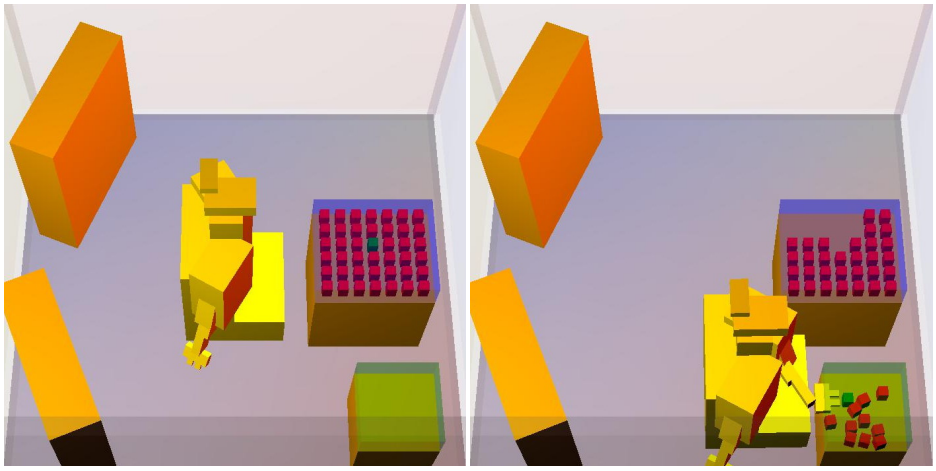
In our experiments, we generate an initial CRG using these parameters during pre-processing and then test whether the value of the heuristic at the initial state is finite. If it is not, we discard it and try again, with the same parameters. Very few retries were necessary to find a CRG with finite heuristic value. This condition was effective: in every case in our experiments, the CRG contained a valid plan.

The following versions of the planner are compared in the experiments:

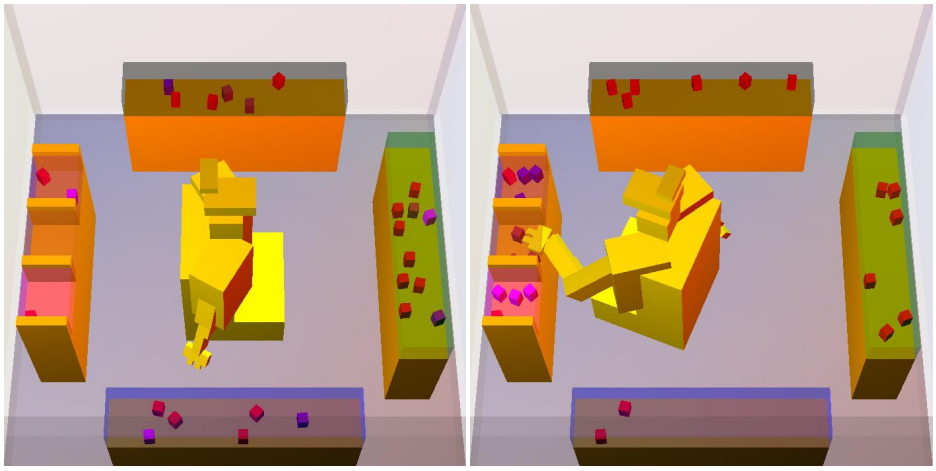
1. No H: The heuristic always returns 0.
2. H_{FF}: This is the original heuristic in FF, based only on the symbolic literals, completely ignoring the reachability conditions when computing the heuristic. Helpful actions are not used.
3. H_{AddR}: This is a version of the original H_{Add} heuristic that returns the sum of the levels of the RPG at which the goal literals are first found. This makes use of the CRG to reason about reachability. It does not build a relaxed plan and, therefore, does not have helpful actions.
4. $H_{\mathrm{FFR,HA}}$: This computes the RPG, does a backward scan to find a relaxed plan and computes helpful actions based on that plan.
5. H_{FFRB}: Like H_{FFR} but using geometric biases to break ties and without using helpful actions.
6. $H_{\mathrm{FFRB,HA}}$: Like H_{FFR} but using geometric biases to break ties and using helpful actions.



(a) Median 18 actions



(b) Median 20 actions



(c) Median 32 actions

Fig. 1: The initial and final state in three of the tasks (3,4,5) in the experiments.

We tested our algorithm on 6 different *tasks*, in which the goals were conjunctions of $In(O_i, R_j)$ for some subset of the objects (the ones not colored red). Other objects were moved as necessary to achieve these goals. The last three tasks are shown in Figure 1; the first three are tasks are simpler variations on task 3 (Figure 1(a)). The table below shows the results of running the algorithms in each of the tasks.

T	Pre	No H			H_{FF}			H_{AddR}			H_{FFR}, HA			H_{FFRB}			H_{FFRB}, HA		
		t	m	s	t	m	s	t	m	s	t	m	s	t	m	s	t	m	s
0	21	265	35	48719	102	72	6123	41	19	536	6	5	78	7	5	87	2	0	23
1	25	300	0	63407	283	17	14300	162	55	2042	3	0	8	16	11	153	4	1	49
2	29	300	0	50903	300	0	8947	300	0	3052	5	1	12	17	13	114	7	2	32
3	23	300	0	39509	300	0	4849	300	0	1767	83	19	464	99	43	523	13	1	69
4	30	300	0	23920	300	0	1574	300	0	1028	300	0	1274	18	3	20	16	3	20
5	51	300	0	9422	300	0	1533	300	0	592	300	1	272	106	17	32	99	14	32

Each entry in the table reports *median time (t)* (in gray), *median absolute deviation, MAD, of the times (m)*, and *states (s) expanded*. Each task also incurs a pre-processing time for building the roadmap; this is reported (in seconds) in the Pre column of the table. The median-based robust statistics are used instead of the usual mean and standard deviation since the data has outliers. Entries with a median time of 300 and MAD of 0 did not successfully complete any of the simulations. There were 20 simulations per task for the first two heuristics and 120 simulations per task for the others. Running times are from a Python implementation running on a 2.6GHz Intel Core i7.

As can be clearly seen, especially in the number of expanded states, exploiting geometric information in the heuristic produces substantial improvements. Introducing geometric biases to settle ties helps in the most cluttered of the examples.

Conclusion We have shown how to combine data structures for multi-query motion planning algorithms with the search and heuristic ideas from the FF planning system to produce a deeply integrated task and motion planning system. The integrated heuristic in this system is quite effective in focusing the search based on geometric information at relatively low cost.

References

1. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2** (1971) 189–208
2. Nilsson, N.J.: Shakey the robot. Technical Report 323, Artificial Intelligence Center, SRI International, Menlo Park, California (1984)
3. Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., Nebel, B.: Semantic attachments for domain-independent planning systems. In: International Conference on Automated Planning and Scheduling (ICAPS), AAAI Press (2009) 114–121
4. Dornhege, C., Hertle, A., Nebel, B.: Lazy evaluation and subsumption caching for search-based integrated task and motion planning. In: IROS workshop on AI-based robotics. (2013)

5. Cambon, S., Alami, R., Gravot, F.: A hybrid approach to intricate motion, manipulation and task planning. International Journal of Robotics Research **28** (2009)
6. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal Artificial Intelligence Research (JAIR) **14** (2001) 253–302
7. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4) (1996) 566–580
8. Plaku, E., Hager, G.: Sampling-based motion planning with symbolic, geometric, and differential constraints. In: IEEE International Conference on Robotics and Automation (ICRA). (2010)
9. Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., Uras, T.: 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)
10. Pandey, A.K., Saut, J.P., Sidobre, D., Alami, R.: 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. (2012)
11. de Silva, L., Pandey, A.K., Gharbi, M., Alami, R.: Towards combining HTN planning and geometric task planning. In: RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications. (2013)
12. Lagriffoul, F., Dimitrov, D., Saffiotti, A., Karlsson, L.: Constraint propagation on interval bounds for dealing with geometric backtracking. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). (2012)
13. Kaelbling, L.P., Lozano-Perez, T.: Hierarchical planning in the now. In: IEEE Conference on Robotics and Automation (ICRA). (2011)
14. Stilman, M., Kuffner, J.J.: Planning among movable obstacles with artificial constraints. In: Workshop on Algorithmic Foundations of Robotics (WAFR). (2006)
15. Stilman, M., Schamburek, J.U., Kuffner, J.J., Asfour, T.: Manipulation planning among movable obstacles. In: IEEE International Conference on Robotics and Automation (ICRA). (2007)
16. Srivastava, S., Riano, L., Russell, S., Abbeel, P.: Using classical planners for tasks with continuous operators in robotics. In: ICAPS Workshop on Planning and Robotics (PlanRob). (2013)
17. Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., Abbeel, P.: Combined task and motion planning through an extensible planner-independent interface layer. In: IEEE Conference on Robotics and Automation (ICRA). (2014)
18. Leven, P., Hutchinson, S.: A framework for real-time path planning in changing environments. I. J. Robotic Res. **21**(12) (2002) 999–1030
19. Plaku, E., Bekris, K.E., Chen, B.Y., Ladd, A.M., Kavraki, L.E.: Sampling-based roadmap of trees for parallel motion planning. IEEE Transactions on Robotics **21**(4) (2005) 597–608
20. Blum, A., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. **90**(1-2) (1997) 281–300
21. Bonet, B., Geffner, H.: Planning as heuristic search. Artificial Intelligence **129**(1) (2001) 5–33
22. Hauser, K.: The minimum constraint removal problem with three robotics applications. I. J. Robotic Res. **33**(1) (2014) 5–17