Heuristic Search for Task and Motion Planning

Caelan Reed Garrett and **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, on the other hand, can efficiently construct plans involving many entities but cannot incorporate the constraints from geometry and kinematics. In recent years, there have been a number of approaches proposed that integrate symbolic task planning with motion planning to attempt to get the best of both worlds. Many of these proposals have attempted to use planners, whether task or motion, as black-boxes, resulting in systems with weak heuristic guidance. In this paper, we show how to generalize the ideas behind one of the most successful symbolic planners in recent years, the FastForward (FF) planner, to the motion planning context. In particular, we show how to incorporate reachability in a configuration-space roadmap into the FF heuristic. The resulting tightly-integrated planner is simple and efficient in a set of representative tasks.

Introduction

Mobile manipulation represents one of the most exciting frontiers in robotics as well as one of its greatest challenges. New robotic platforms with mobile bases and 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 are totally inadequate when 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 (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. 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 our 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 shape, the shape of the robot, where the robot stands 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, Dornhege et al. (2009), Dornhege, Hertle, and Nebel (2013), 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. Requiring the user to specify a sparse set of placements to consider limits the usefulness of the planner, while avoiding the motion planner in the heuristic leads to a heuristic that is uninformed about geometric considerations and results in considerable 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, Alami, and Gravot (2009). 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 application of the heuristic used in the FastForward (FF) (Hoffmann and Nebel 2001) planning system that integrates reachability in the robot configuration space with reachability in the symbolic states. Both the search and the computation of the heuristic exploit a roadmap (Kavraki et al. 1996) data structure that allows multiple motion-planning queries on the closely related problems that arise during the search to be solved efficiently.

Related work

There have been a number of approaches to integrated task and motion planning in recent years. The pioneering Asymov system of Cambon, Alami, and Gravot (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, thinking about how to allocate 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 (2010) is similar in approach.

The work of Erdem et al. (2011), like the work of Dornhege et al. (2009), extends a task planner, based on explicit causal reasoning, with the ability to check for the existence of paths for the robot.

Pandey et al. (2012) and de Silva et al. (2013) use HTNs instead of generative task planning; their approach to integrating the symbolic and geometric levels is similar but more general. 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). Also, they use a cascaded approach to computing difficult applicability conditions: the precondition to picking up an object is that it be accessible, which requires making a geometric plan to determine exactly. They describe conservative, quick-to-evaluate approximations of these predicates, so that the planning is only attempted in situations in which it might plausibly succeed.

Lagriffoul et al. (2012) also integrate the symbolic and geometric search. They generate a set of approximate linear constraints imposed by the program under consideration,

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: $Holding(O, G)$, $RobotConf(C_1)$

Figure 1: Example operator descriptions

e.g., from grasp and placement choices, and use linear programming to compute a valid assignment or determine one does not exist.

The HPN approach of Kaelbling and Lozano-Perez (2011) chooses candidate actions based on *generators* that use geometric considerations based on the current state. In particular, this approach uses *goal regression* to reason from the goal towards the initial state. This enables prioritizing actions that are likely to achieve the goal. The use of 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. (2006; 2007) 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. (2013; 2014) 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.

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.

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 using PDDL-like operator descriptions such those shown in figure 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 preprocessing phase. The sampling is problem-driven but may be, at first, inadequate for any given problem. If planning were to fail, additional samples could be added and planning re-attempted, although this was not necessary for the examples addressed in our experiments.

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. Picking or placing an object might affect a very large number of $Reachable(C_1, C_2)$ or $In(O, R)$ literals, where R is a region of space, and it is impractical to pre-compute and state them explicitly. 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 (Dornhege et al. 2009).

The last difficulty is in computing the answers to queries in the details, especially about reachability, which requires finding a free path between robot configurations. We address this problem by using a shared *roadmap* data structure, related to a PRM (Kavraki et al. 1996), for answering all reachability queries, and taking a lazy approach which computes on demand and caches the answers to speed future queries.

Problem definition

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:

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

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

$$\text{SATISFIES}(s, \Gamma) \equiv \forall l \in \Gamma.\ \text{HOLDS}(l, s) \quad .$$

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

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

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

$$\text{APPLICABLE}(a, s) \equiv \forall l \in a.\phi.\ \text{HOLDS}(l, \phi) \quad .$$

An *operator schema* is an operator with typed variables, and stands 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 (which 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.

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.

Conditional reachability graph

The conditional reachability graph (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 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. 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.

The annotations 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 in figure 2. It begins 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 sample one or more robot configurations that satisfy the kinematic constraints that the object be grasped using the SAMPLEIK procedure. 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 Sampling-Based Roadmap of Trees (Plaku et al. 2005).

To test whether this set of poses and configurations is plausible, we use it to compute a heuristic value of the starting state. If it is infinite, meaning that the goal is unreachable even under extremely optimistic assumptions, then we return to this procedure and add more samples to each component.

```
CONSTRUCTCRG(W, T, s, Γ, O_f, O_m, θ) :
 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 = SAMPLEOBJPOSE(o.shape, T)
 6          P_o.add(p)
 7          for g ∈ o.grasps
 8              for j ∈ {1, ..., θ.n_ik}
 9                  c = SAMPLEIK(g, o, p)
10                  N.add(c, (g, o, p))
11              for j ∈ {1, ..., θ.n_n}
12                  c = SAMPLECONFNEAR(g)
13                  N.add(c)
14  E = { }
15  for n_1 ∈ N
16      for n_2 ∈ NEARESTNEIGHBORS(n_1, k, N)
17          if CFREEPATH(n_1.c, n_2.c, O_f)
18              E.add(n_1, n_2)
19  N, E = CONNECTTREES(N, E, W, θ.n_c)
20  return ⟨P, N, E⟩
```

Figure 2: Procedure for constructing the CRG

```
REACHABLETEST(c_1, c_2, D, 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.n_1.c, e.n_2.c, o@p,
 5                            D.heldObj, D.grasp)
 6              v = not(p == None)
 7              e.valid.add(⟨D.heldObj, D.grasp, o, p, v⟩)
 8  G = {e ∈ CRG.E | ∀(o, p) ∈ D.objects.
 9              ⟨D.heldObj, D.grasp, o, p, True⟩ ∈ e.valid}
10  return REACHABLEINGRAPH(c_1, c_2, G)
```

Figure 3: Using the CRG to determine reachability.

Querying the CRG

Now that we have a CRG we can use it to compute the test for the *reachable* literal, as shown in figure 3. The main part of the test is in lines 8–10: 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, as specified in the details D, and then search in that graph to see if configuration c_2 is reachable from c_1. Lines 1–7 check to be sure that the relevant validity conditions have been computed and computes them if they have not. The procedure CFREEPATH$(c_1, c_2, obst, o, g)$ simply 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 the procedure APPLICABLEOPS(s, Ω, CRG), which efficiently determines

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):
4      HEURISTIC(RPG(s, Γ, CRG, Ω))
5  q = QUEUE(SEARCHNODE(s, 0, H(s), None))
6  while not q.empty():
7      n = EXTRACT(q)
8      if SATISFIES(n.s, Γ)
9          return n.path
10     for a ∈ APPLICABLEOPS(n.s, Ω, CRG)
11         s' = SUCCESSOR(n.s, a)
12         q.push(SEARCHNODE(s', n.cost + 1, H(s'), n))
```

Figure 4: An illustration of the role of the CRG in a generic heuristic search method.

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 $(C_2, (G, O, P)) \in \text{CRG}.N$.

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.

Figure 4 shows 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.

Computing the relaxed plan graph

In classical symbolic planning, a *plan graph* is a sequence of alternating *layers* of literals and actions (operations). 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* (Blum and Furst 1997) and related planning algorithms.

The *relaxed plan graph* is a simplified plan graph, in which the mutual exclusion conditions are ignored; 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 (Bonet and Geffner 2001) 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} (Bonet and Geffner 2001), 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 (Hoffmann and Nebel 2001), which also requires an efficient backward-chaining pass in the plan graph to determine how many operations, 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 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 operation to those that are in the first level of the relaxed plan, and find that it makes the search more efficient.

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 value of some literals are computed from the details (via the CRG). In figure 5 we present a method for computing the RPG that is domain-dependent.

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* operation. 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 details (via the CRG), which become 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 while substantially reducing the number of states that need to be explored.

Figure 6 illustrates the operation of a heuristic based on counting the level of the RPG at which goals appear. In the root state (at the top left), the blue circular objects are reachable from the start state, so pick actions for all of them appear at level 1 in the RPG. Each of those pick actions removes an object from the details and thus increases the accessible

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

Figure 5: Computing the relaxed plan graph.

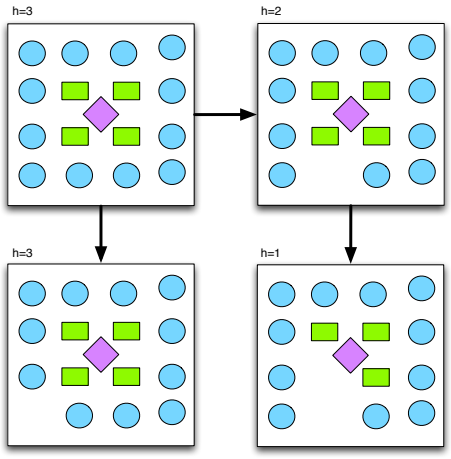


Figure 6: Illustration of the operation of a heuristic based on the RPG; the goal is to pick the central block.

configurations. Then, at level 2 of the RPG, the green rectangular objects become reachable and are removed from the details. Then, the goal purple object is reachable at level 3. Note that different successor states will have different heuristic values depending on whether the contemplated action removes a relevant object. In the lower left state, removing the corner object does not make any of the green objects directly reachable, and so the heuristic value is not reduced. This is in contrast to the state in the upper right.

Figure 5 shows the algorithm in more detail. In line 2, 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 operation 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 4, 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 8, 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 end up affecting the resulting $hState$ in the same way. Lines 9–11 constitute the standard computation of the RPG. In lines 12–14 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, and 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 18 creates a new $hState$, which consists of all literals possibly achievable up to this level and the details with possibly more objects removed.

There is one last consideration: the strategy illustrated in figure 5 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 the following simple bookkeeping strategy to maintain a causal connection between actions and literals, which will enable 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 effects set of the $pick$ operation, just when it is used in the heuristic computation.
- For any action that is added to an action layer during the computation of the RPG, its $Reachable(c_1, c_2)$ precondition is replaced by a conjunction of $Picked(o)$ conditions. We determine this set by planning a path from c_1 to c_2 in the CRG and finding which *already picked* objects obstructed that path.

This condition is too strong: in fact, there may be many different sets of objects which, if moved, would enable c_2 to be reached, and there is the potential that a plan with more than the minimum number of actions in the relaxed graph will be selected. This affects the accuracy and admissibility of the heuristic, but the FF heuristic is already known to be inadmissible. We are interested only in the heuristic's effectiveness in the context of a satisficing planner.

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, but can result, empirically, in improved performance. 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. If an object must be placed in a goal region, it should bias the placement to minimize the number of other placement configurations made unreachable (e.g., by putting the object up against a wall or other region boundary). 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 tight 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.

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.

Algorithms

The search strategy in all of our experiments is enforced hill-climbing (Hoffmann and Nebel 2001), 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 to not be complete. We have experimented with other search strategies but have found that hill-climbing works better for 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.

The parameter 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).
- $n_c = 500$ (number of RRT iterations)
- $k = 4$ (number of nearest neighbors)

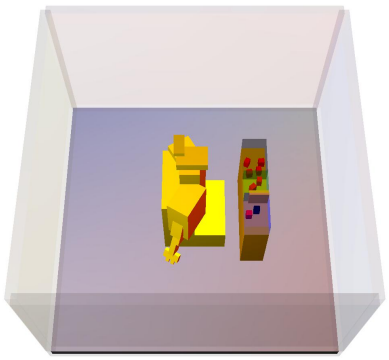
In our experiments, we generate an initial CRG using these parameters during pre-processing and then test whether the value of the heuristic at the initial state is finite. If it is not, we discard it and try again, with the same parameters. Very few retries were necessary to find a CRG with finite heuristic value. This condition was effective: in every case in our experiments, the CRG contained a valid plan. One important intuition about these tasks is that by allowing objects to be moved (and assuming there is a place to move them to), we have actually reduced the need to sample many configurations of the robot. As long as there is one reachable grasp for the object that is feasible relative to the fixed obstacles, we can move the other objects out of the way, enabling the robot to reach it. Of course, puzzle-like situations can be constructed that will require extensive sampling.

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

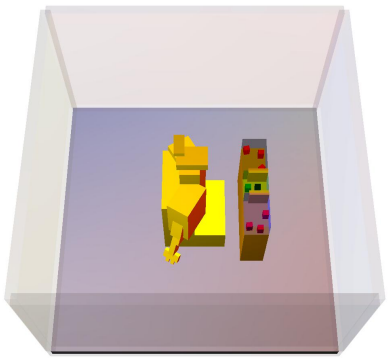
1. No H: The heuristic always returns 0.
2. H_{FF}: This is the original heuristic in FF, based only on the symbolic literals, completely ignoring the reachability conditions when computing the heuristic. Helpful actions are not used.
3. H_{AddRob}: This is a version of the original H_{Add} heuristic 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. $H_{\mathrm{FFRob,HA}}$: This computes the RPG, does a backward scan to find a relaxed plan and computes helpful actions based on that plan.
5. $H_{\mathrm{FFRobBias}}$: Like H_{FFRob} but using geometric biases to break ties and without using helpful actions.
6. $H_{\mathrm{FFRobBias,HA}}$: Like H_{FFRob} but using geometric biases to break ties and using helpful actions.

Test domains

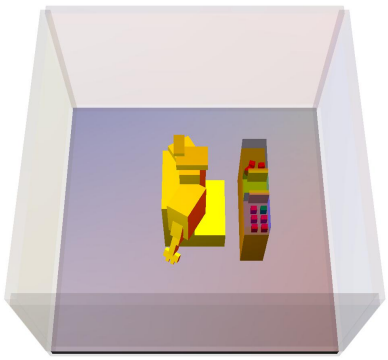
We tested our algorithm on 10 different *tasks*, shown in Figure 7. The Move and Move+Clutter tasks require moving one object out of the way to reach the target object on the right of the table and take it to a target region on the left. In the Regrasp and Regrasp+Clutter tasks, the target object and an obstacle are in the narrow "chute" in the center of the



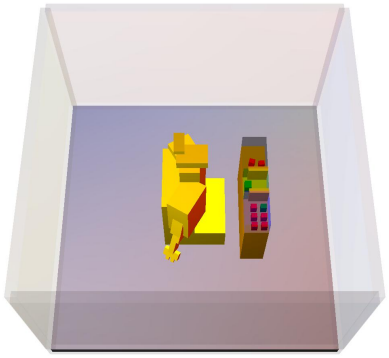
(a) Move+Clutter: Avg 4.2 actions



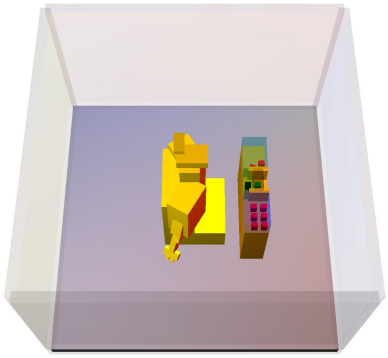
(b) Regrasp+Clutter: Avg 5.8 actions



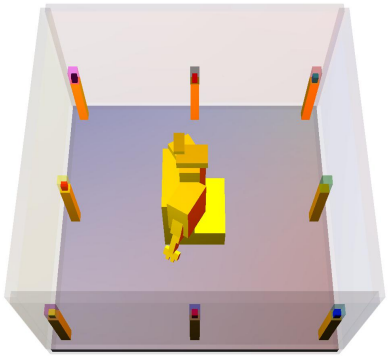
(c) Dig: Avg 8.0 actions



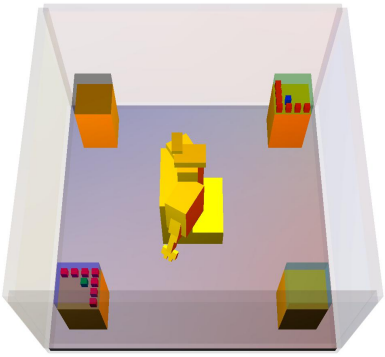
(d) Double Dig: Avg 8.4 actions



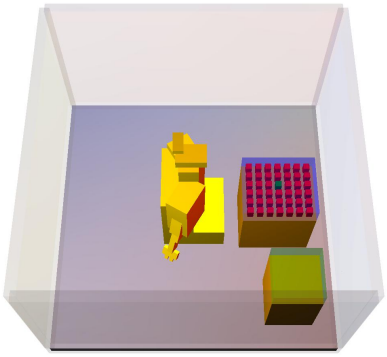
(e) Swap: Avg 10.0 actions



(f) Transport: Avg 7.8 actions



(g) Walls: Avg 18.8 actions



(h) Table: Avg 23.0 actions

Task	Ob	Pre	No H			H_{FF}			H_{AddRob}			H_{FFRob}, HA			$H_{\mathrm{FFRobBias}}$			$H_{\mathrm{FFRobBias}}, HA$		
			f	t	s	f	t	s	f	t	s	f	t	s	f	t	s	f	t	s
a2	2	14	1.0	3	30	1.0	3	4	1.0	3	4	1.0	3	4	1.0	3	4	1.0	4	4
a	2	23	1.0	17	170	1.0	10	29	1.0	6	13	1.0	4	4	1.0	4	4	1.0	4	4
b2	2	15	1.0	7	504	1.0	6	71	1.0	4	33	1.0	4	33	1.0	5	31	1.0	5	32
b	8	24	0.9	125	2304	0.9	70	437	0.9	21	68	1.0	4	15	1.0	31	73	1.0	5	14
c	9	19	0.6	214	4828	0.7	84	624	0.9	22	109	1.0	5	14	1.0	5	7	1.0	6	14
d	9	17	0.1	205	3787	0.8	112	774	0.9	87	329	1.0	7	9	0.9	11	12	1.0	7	9
e	10	16	0.0	—	—	0.4	115	848	0.6	87	352	0.9	9	27	1.0	17	43	1.0	9	26
f	8	23	0.1	245	4784	0.1	17	34	0.1	39	176	1.0	8	67	1.0	5	15	1.0	6	23
g	16	36	0.0	—	—	0.0	—	—	0.0	—	—	1.0	65	34	1.0	85	24	1.0	84	38
h	42	66	0.0	—	—	0.0	—	—	0.0	—	—	0.6	74	45	1.0	121	22	1.0	104	23

Figure 7: (a)-(h) Tasks used in experiments; the (a2) and (b2) tasks are like the (a) and (b) tasks but with only two objects on the table. The Ob column has the number of movable objects and Pre has the processing time (secs). Other entries report: *success fraction* (f) of the problems solved within the 300 sec cutoff; *time* (t) (in gray), average planning time (secs) for the successful trials and *states* (s) the average number of states expanded during successful trials. Entries with — indicate none of the trials succeeded. There were 10 simulations per setting.

table; the goal is to place the target object, currently in front of the chute, to the location in the back where the obstacle is. This requires placing the target object at an intermediate location, moving the obstacle out of the way and then retrieving the target object.

In the Dig task, the target object is at the back of a row of obstacles and it has to be placed in the central chute. Two of the blocking obstacles must be removed in order to reach the target object. The Double Dig task is like Dig except that the chute is blocked by the presence of another object, which is supposed to remain in the chute. This blocking object has to be re-arranged to make room for the new object to be placed in the chute.

The Swap task involves interchanging the locations of two objects in the central chute. In the Transport task there are 8 objects and 9 posts. The goal is to place each of the objects on a previously specified post. The Walls task involves interchanging the location of two objects each of which is behind a "wall" of obstacles. The Table task requires reaching an object, which is near the middle of a of 6×7 array of objects, and placing it on another table. This requires moving quite a few objects out of the way. The final states, showing the moved objects, can be seen in Figure 8.

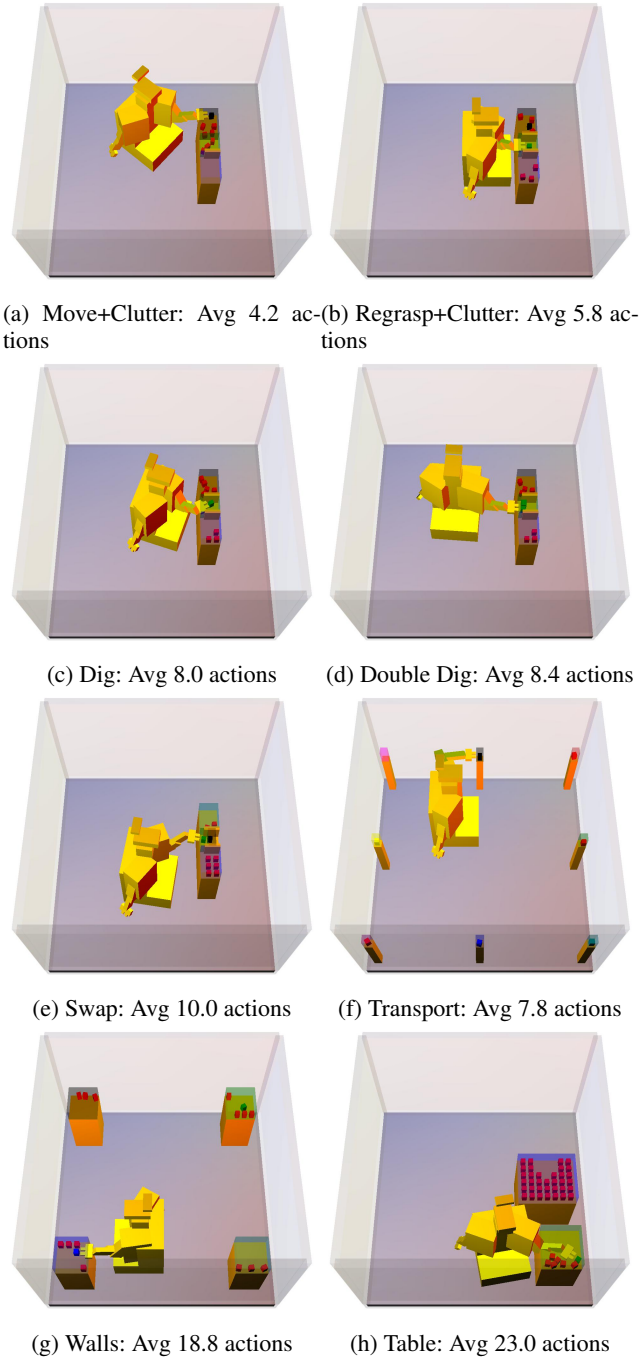
The table in Figure 7 shows the results of running the algorithms in each of the tasks. Each entry in the table reports *success fraction (f), time (t), states (s)*, where *success fraction* is the fraction of the problems solved within the 300 second cutoff, *time* is the average planning time (over 10 simulations) of all the successful trials in seconds and *states* is the average number of states expanded during planning in the successful trials. Each task also incurs a pre-processing time for building the roadmap; this is reported (in seconds) in the Pre column of the table. Average times and states have been rounded to integers.

As can be clearly seen, especially in the number of expanded states, exploiting geometric information in the heuristic produces substantial improvements in all but the simplest tasks. Introducing geometric biases to settle ties helps in the most cluttered of the examples but further experimentation is required to establish its merits.

Time as a function of distractors

To test the effect on the algorithm of adding extra "distractor" objects, we ran the $H_{\mathrm{FFRobBias}}, HA$ version of the algorithm on a task that requires moving three objects from one table to another and varied the number of objects on a nearby table from 0 to 28. The pre-processing times increase substantially (from 13.5s to 37.9s), but the planning time increases very little (from 7.4s to 9.8s). The slow growth in planning time shows that the heuristic is effective at focusing the planner on the relevant part of the space.

All running times are from an implementation running on a 2.8GHz Intel Core i7. All the code is written in Python, so there is substantial room for decreasing all of these times, even without further algorithmic improvements, especially in the low-level geometric processing, such as collision-checking, for which much more efficient algorithms and implementations exist.



(a) Move+Clutter: Avg 4.2 actions

(b) Regrasp+Clutter: Avg 5.8 actions

(c) Dig: Avg 8.0 actions

(d) Double Dig: Avg 8.4 actions

(e) Swap: Avg 10.0 actions

(f) Transport: Avg 7.8 actions

(g) Walls: Avg 18.8 actions

(h) Table: Avg 23.0 actions

Figure 8: Final states for the tasks.

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. Although a number of extensions are required before it can be useful in tackling realistic tasks, the performance of the system is already quite promising even in its pilot implementation.

References

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artif. Intell.* 90(1-2):281–300.

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

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* 28.

de Silva, L.; Pandey, A. K.; 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*.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic attachments for domain-independent planning systems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 114–121. AAAI Press.

Dornhege, C.; Hertle, A.; and Nebel, B. 2013. Lazy evaluation and subsumption caching for search-based integrated task and motion planning. In *Proceedings of the 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 *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

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

Kaelbling, L. P., and Lozano-Perez, T. 2011. Hierarchical planning in the now. In *IEEE Conference on Robotics and Automation (ICRA)*.

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

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)*.

Leven, P., and Hutchinson, S. 2002. A framework for real-time path planning in changing environments. *I. J. Robotic Res.* 21(12):999–1030.

Nilsson, N. J. 1984. Shakey the robot. Technical Report 323, Artificial Intelligence Center, SRI International, Menlo Park, California.

Pandey, A. K.; Saut, J.-P.; 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*.

Plaku, E., and Hager, G. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Plaku, E.; Bekris, K. E.; Chen, B. Y.; Ladd, A. M.; and Kavraki, L. E. 2005. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics* 21(4):597–608.

Srivastava, S.; Riano, L.; Russell, S.; and Abbeel, P. 2013. Using classical planners for tasks with continuous operators in robotics. In *ICAPS Workshop on Planning and Robotics (PlanRob)*.

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 Conference on Robotics and Automation (ICRA)*.

Stilman, M., and Kuffner, J. J. 2006. Planning among movable obstacles with artificial constraints. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR)*.

Stilman, M.; Schamburek, J.-U.; Kuffner, J. J.; and Asfour, T. 2007. Manipulation planning among movable obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.