Using Classical Planners for Tasks with Continuous Operators in Robotics

Siddharth Srivastava and **Lorenzo Riano** and **Stuart Russell** and **Pieter Abbeel**
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720

Abstract

The need for high-level task planning in robotics is well understood. However, interfacing discrete planning with continuous actions often requires extensive engineering of the solution. For instance, picking up an object may require removing many others that obstruct it. Identifying the exact obstructions requires geometric reasoning which is prohibitively expensive to precompute, with results that are difficult to represent efficiently at the level of a discrete planner. We propose a new approach that utilizes representation techniques from first-order logic and provides a method for synchronizing between continuous and discrete planning layers. We evaluate the approach and illustrate its robustness through a number of experiments using a state-of-the-art robotics simulator, accomplishing a variety of challenging tasks like picking objects from cluttered environments, where the planner needs to figure out which other objects need to be moved first to be able to reach the target object, and laying out a table for dinner, where the planner figures out effective tray-loading, navigation and unloading strategies.

1 Introduction

A robot capable of achieving high-level goals was one of the original motivations behind the automated planning problem (Fikes and Nilsson 1971). Since this early work, numerous advances have been made in automated planning. Discrete planners are able to automatically compute effective problem-specific heuristics for solving planning problems specified implicitly in terms of parameterized operators with preconditions and effects (e.g.,(Hoffmann and Nebel 2001; Helmert, Haslum, and Hoffmann 2007)). Continuous state space planners have also been developed for solving the lower level search problems in the configuration space of a robot to achieve desired motion trajectories (LaValle 2006). Techniques from both of these areas are required for a real-world robot to solve a high-level problem like preparing a table for dinner. However, using classical planners to solve planning problems encountered by a robot presents several fundamental challenges.

For instance, consider the task of picking up an object in blocks-world, a domain that is widely regarded as trivial for modern classical planners. For instance, consider the task of picking up an object. If the table is cluttered, a continuous planner will fail to find a solution because other objects will have to be removed first (see Fig. 1). A discrete planner could compute the sequence of operations required to clear a path to the object, but only if it gets the set of obstructions. On the surface this seems to be easily resolved by adding a high-level predicate, *obstructs*(c, b_1, b_2) where c ranges over the robot configurations and b_i over objects. However, any such formulation leads to two major problems. First, the robot configuration is usually a high dimensional real-valued vector (11-dimensional for the PR2 using only one gripper). Representing this problem for a discrete planner requires a discretization of the possible configurations which in itself is infeasible for the high dimensional spaces involved. Moreover, a pre-processing step will need to compute the graspability and obstruction facts specific to the given scenario, for every combination of discretized configurations and objects used as arguments to the corresponding predicates. This will lead to planner input files with thousands of configuration symbols at the least, making planning infeasible. Even if all of these representational steps are carried out, it is difficult to specify in a discrete problem definition how obstructions change on the application of a pick and place operation, due to the spatial reasoning involved.

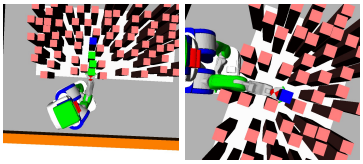

Figure 1: Example scenario with the target object in blue (L) and the desired solution after removing obstructing objects (R)

These issues are symptoms of the following fundamental problems: how to enable classical planners to efficiently utilize information provided by a continuous planner, and how to utilize them in situations where facts and operator effects over continuous variables are not available a priori. Our main contribution is a new approach addressing these problems. We show how this approach solves number of challenging robot planning tasks.

Overview of Our Approach We assume that every high-level action corresponds to a continuous implementation. The continuous implementation may use a variety of techniques for accomplishing the post-conditions of the high-level action. If the preconditions of the high-level action are satisfied, a trajectory is guaranteed to be found by the continuous implementation. However, as discussed in the introduction, the preconditions of high-level actions may involve spatial reasoning and are generally not computable a priori. Thus, the high-level planner may be wrong in selecting an action for execution at a certain step. If a solution trajectory cannot be found, the lower level returns the violated preconditions to the high-level planning layer, which incorporates the new facts and computes a new high-level solution plan. Throughout this paper, we will use the terms "discrete plan-

ning" interchangeably with high-level planning and "lower level" with continuous planning and execution.

While this overall approach is similar to re-planning (Talamadupula et al. 2010; Yoon, Fern, and Givan 2007), our focus is on the problem of continuous operator representations for high-level, discrete planning and communication between the continuous and discrete planning levels. The discrete problem is specified by replacing the domains of real-valued variables (required to express action preconditions and effects, even at the high-level) with finite sets of *Skolem symbols* that are independent of their lower level implementation. This translation is described in Sec. 2 The interface between the low-level and high-level planning layers is described in Sec. 3 The resulting approach allows easy modeling of various tasks. We present many experimental results in a simulated execution scenario developed using OpenRave (Diankov 2010) for the PR2 robot in Sec. 4 A summary of related work is presented in Sec. 5

2 Problem Formulation

We assume that a propositional framework like typed PDDL (Fox and Long 2003) is used for specifying high-level planning domains and problems. We formalize the representation briefly as follows. A *planning domain* $\langle \mathcal{R}, \mathcal{A} \rangle$ consists of a set of relation symbols $\mathcal{R}$ with their arities and type signatures, and a set $\mathcal{A}$ of actions. Let the set of literals and atoms constructed using a set of relations $\mathcal{R}$ and a set of typed variables or constants V be *lit*$(\mathcal{R}, V)$ and *atoms*$(\mathcal{R}, V)$ respectively. An action is defined as $\langle$*Param, Pre, Eff*$\rangle$, where *Param* is a sequence of typed parameter variables; *Pre* is a first-order logic formula and *Eff* is a conjuction, both over *lit*$(\mathcal{R}, V)$ with only the variables in *Param* being free. With some syntactic limitations, such expressions can be represented in the PDDL input language used by most classical planners.

Given a finite set U of typed constants (informally constituting the "objects" in a problem), a state is an assignment of truth values to each atom in *atoms*$(\mathcal{R}, U)$; for compactness, we use the closed world assumption to represent states as sets of true atoms. We define ground actions as $a(\bar{c})$, where $\bar{c}$ denotes a mapping from the parameters of a to appropriately typed constants from U. The preconditions and effects of a ground action $a(\bar{c})$ are obtained using the variable assignments in $\bar{c}$ in the usual way. We consider full observability and deterministic action effects, so that a ground action ga is applicable in a state s iff its precondition is true in s.

Finally, a *planning problem* consists of a planning domain $\langle \mathcal{R}, A \rangle$, a set U of typed constants, an initial state s_0 and a set $g \subseteq$ *atoms*$(\mathcal{R}, U)$ denoting the goal condition. A solution to a planning problem is a sequence of ground actions $ga_1, \ldots ga_k$ such that ga_i is applicable in $ga_{i-1}(\cdots ga_1(s_o)\cdots)$ and $ga_k(\cdots ga_1(s_0)\cdots)$ satisfies g.

Domain Transformation We introduce the central principle behind our representation with an example. Consider an action like grasping in a pick and place domain. Its preconditions include real-valued vectors and preconditions require spatial reasoning. For clarity in this description, we consider a situation where objects are not stacked and assume that the target location where an object has to be placed is clear. The high-level grasp action can be specified as follows:

grasp(c, obj_1)
precon *graspable*$(c, obj_1) \wedge$ *robotAt*(c)
$\wedge \forall obj_2 \neg$ *obstructs*(c, obj_2, obj_1)
effect *in-gripper*(obj_1)

In this specification, the variable c represents robot configurations. In order to motivate our approach, consider the effect of this action in a framework like situation calculus (Levesque, Pirri, and Reiter 1998) but using a timestep rather than a situation to represent the fluents:
$\forall t, obj_1 \forall c(graspable(c, obj_1, t) \wedge robotAt(c, t) \wedge \forall obj_2 \neg obstructs(c, obj_2, obj_1, t) \rightarrow inGripper(obj_1, t+1))$

Although this is not the complete *successor state axiom* for *in-gripper* it is sufficient to illustrate the principle we use. We first use the clearer, logically equivalent form:
$\forall t, obj_1(\exists c(graspable(c, obj_1, t) \wedge robotAt(c, t) \wedge \forall obj_2 \neg obstructs(c, obj_2, obj_1, t)) \rightarrow inGripper(obj_1, t)$

which asserts more clearly that *any* value of c that satisfies the preconditions allows us to achieve the postcondition. We can now use the standard technique of Skolemization to replace occurrences of c with a function of obj_1, t:
$\forall t, obj_1((graspable(gp(obj_1, t), obj_1, t)$
$\wedge\, robotAt(gp(obj_1), t)$
$\wedge \forall obj_2 \neg obstructs(gp(obj_1), obj_2, obj_1, t))$
$\rightarrow inGripper(obj_1, t+1)$

where the Skolem function $gp(x, t)$ is just a symbol of type *location* to the discrete planner. Intuitively, it represents a robot configuration corresponding to the grasping-pose of x. A potential problem however, is that the Skolem functions will depend on t, or the current step in the plan. This can be ignored as long as it is possible to recompute (or reinterpret) f when an action's precondition is violated by its existing interpretation during the execution (as is the case in our implementation). Therefore, to represent the grasp operator for a discrete planning problem, rather than using a discretized space of configurations, we only need to add symbols of the form $f(\bar{o})$ in the planning problem specification, for each object argument tuple $\bar{o}$ consisting of the original objects. Since many classical planners don't support functions, they are reified as objects of the form $f\_\bar{o}$ with an associated set of always true relations, e.g. $is\_f(f\_o_i, o_i)$. The discrete description of grasp becomes:

grasp(ℓ, obj_1)*:*
precon *is_gp*$(\ell, obj_1) \wedge$ *graspable*(ℓ, obj_1)
$\wedge$ *robotAt*(ℓ)
$\wedge \forall obj_2 \neg$ *obstructs*(ℓ, obj_2, obj_1)
effect *in-gripper*(obj_1)
$\wedge \forall \ell_2, obj_3 \neg$ *obstructs*(ℓ_2, obj_1, obj_3)

ℓ ranges over the finite set of constant symbols of the form gp_obj_i where obj_i are the original constant symbols in the problem. Such transformations are applied whenever the continuous variable occurs only in preconditions. The putDown action has a similar specification, with the Skolem function $pdp(target_{\text{loc}})$ denoting the pose required for putting down an object o at location ℓ_1 when the robot is in configuration ℓ_2.

putDown(o, ℓ_l, ℓ_2)*:*

precon *is_pdp*$(\ell_2, \ell_1) \wedge$ *poseForPlacing*(ℓ_2, o, ℓ_1)
InGripper$(o) \wedge$ *RobotAt*(ℓ_2)
effect $\neg$ *InGripper*$(o) \wedge$ *At*(o, ℓ_1)

The *moveto*(c_1, c_2) action changes the robot's configuration with the only precondition that it is at c_1. We assume that the environment is not partitioned and the robot can move between any two collision-free areas. These three actions constitute a complete description of the high-level pick and place domain.

It is possible that a continuous variable occurs freely in preconditions and effects of an action, e.g. $\forall x, c(\varphi_{precon}(x, c) \rightarrow \varphi_{effect}(x, c))$. In this case we don't perform Skolemization. The symbol used for c in this case will be an action argument, and must either be one of the original symbols in the domain or one already introduced via Skolemization. Although this exhausts the set of actions commonly used in PDDL benchmark problems, the accurate description of an action may involve side-effects on symbols not used as its arguments. E.g., the putDown(obj_1, loc_1) action may deterministically introduce obstructions between a number of robot configurations and other objects. We don't encode such side effects in the high-level planning problem specification; these facts are discovered and reported by the lower level when they become relevant.

To summarize, we transform the given planning domain with actions using continuous arguments by replacing occurrences of continuous variables with symbols representing Skolem function application terms whenever the continuous variable occurs in the precondition but not in the effects. Every continuous variable or the symbol replacing one, of type τ gets the new type τ_{sym}. Planning problems over the modified domains are defined by adding finite set of constants for each such τ_{sym} in addition to the constants representing problems in the original domain. The added constants denote function application terms, e.g. *gp_obj17*, for each physical object and Skolem function application. This increases the size of the input only linearly in the number of original objects if the Skolem functions are unary. Note that the set of Skolem functions itself is fixed by the domain and does not vary across problem instances. The initial state of the problem is described using facts over the set of declared objects, e.g. "*is_gp(gp_obj17, obj17)*" denoting that the location name *gp_obj17* is a grasping pose for *obj17* and "*obstructs(gp_obj17, obj10, obj17)*", denoting that *obj10* obstructs *obj17* when the robot is at *gp_obj17*. In this formulation, the size of the input problem specification is independent of the sampling-based discretizations.

Discrete and Continuous States A low-level state extends a discrete state by associating actual values (interpretations) with each Skolem symbol. Thus, every low-level state corresponds to a unique discrete state representation. Conversely, each discrete state represents a set of possible low-level states. Since the low-level has complete information about states, it can compute (though not efficiently) the truth value of any atom in the vocabulary at any step, given an interpretation of every Skolem symbol used in the atom. E.g., the low-level can compute the truth value of *obstructs(***`cval, obj10, obj17`***)* where **cval** is a numeric vector representing a robot pose. However, we wish to avoid the determining the value of such predicates due to the associated computational cost and do so only when an action is not executable at a certain step in the low-level. In this case the low-level has to report a set of truth valuations of atoms showing that the action's preconditions do not hold.

3 Discrete and Continuous Planning

In this section we present the overall solution approach and its completeness properties followed by the details of the low-level interpretation and execution subroutines.

3.1 Discrete Planning and the Interface

As noted above, the initial state for a planning problem is defined using the ground atoms which are true in the initial state. However, the truth values of ground atoms over Skolem symbols like *obstructs(gp_obj17, obj10, obj17)* are not known initially. Such atoms are initialized with domain specific defaults. The relationship between the choice of defaults and completeness of the approach is discussed below.

Our overall algorithm is shown in Alg. 1. In line 2, a classical planner is called with state s, which is initialized with the input problem state. If no solution is found, the cached values of Skolem symbols in the lower layer are cleared and all facts using Skolem symbols are reset to their default values using *clear_cache()*. The test in lines 3 and 5 ensure that this is done only once per iteration. If the problem is unsolvable even after clearing the cache, line 6 terminates the algorithm. In line 7, the computed plan π is passed to the low-level, which uses sampling to estimate the values of Skolem symbols and RRT-based techniques to implement movements representing each action in the plan. Because the high initial state in line 2 used default values for facts over Skolem symbols, π may not be fully executable in the low level. If this is the case, the low-level reports the step/action of the plan that failed and the reason for that failure, represented by an assignment of atoms that violate the action precondition (a minimal violating assignment is sufficient). These components are described in Sec. 3.2.

If the operators that invalidate a symbol's cached value are known, they are provided to execute() as C. The low-level then reinterprets all affected Skolem symbols after executing an action. In line 9, *UpdateState()* uses the PDDL descriptions of actions to propagate s forward using π until *errStep* and adds *violatedPrecons* to the resulting state. The entire iteration then repeats with the resulting state unless a preset resource limit is reached.

Completeness We use the notion *probabilistically-complete* (LaValle 2006) to categorize algorithms that are guaranteed to find a solution with sufficiently many samples. We use the concept of positive and negative occurrences of atoms in formulas. Intuitively, an occurrence of an atom in a formula is negative (positive) if it is negated (non-negated) in the negated-normal-form of the formula.

Definition 1. *A planning domain is* uniform *wrt a goal g and a set of predicates R if for every $r \in R$, we have (1) occurrences of r in action preconditions and goal are either always positive, or always negative, and (2) actions can only add atoms using r in the form (positive or negative) used in preconditions and the goal g.*

Algorithm 1: Discrete-Continuous Interface

Input: PDDL state s & domain D; cache-voiding ops C

1 **repeat**
2 $\pi \leftarrow$ classicalPlanner(s, D)
3 **if** *plan not found & not cleared_cache* **then**
4 clear_cache(); continue
5 **else if** *plan not found* **then**
6 return "unsolvable"
end
7 $\langle$*errStep, violatedPrecons*$\rangle \leftarrow$ execute(π, s, C)
if errStep *is null* **then**
8 return "success"
else
9 s = UpdateState(s, π, *errStep, violatedPrecons*)
end
until *resource limit reached*

Algorithm 2: Find a Grasping Pose

Input: Object o

1 **repeat**
2 $p \leftarrow$ sampleAround(o)
3 $t \leftarrow$ sampleTorsoHeight()
4 $gps \leftarrow$ generateGrasps(p, t, o)
5 **for** $gp_i \in gps$ **do**
6 **if** *IKsolution*(p, t, gp_i) **then**
7 **if** *obstaclesFreeArmPath*(p, t, gp_i) **then**
8 return (p, t, gp_i)
end
end
end
until *max number of attempts reached*

Let P_S be the set of predicates whose atoms may use Skolem symbols as one of their arguments, and let $D = \langle \mathcal{R}, \mathcal{A} \rangle$ be a planning domain that is uniform wrt a goal g and P_S. Let the *default* assignment of truth values to atoms over P_S, be one that assigns each positively (negatively) occurring predicate the truth value true (false). Let D be a planning domain and U a set of typed constants. A dead-end for $\langle D, U \rangle$ wrt g is a state over $atoms(\mathcal{R}, U)$ which has no path to g. We say that two low-level states are physically similar if they differ only on the interpretation of Skolem symbols and atoms using Skolem symbols. We use the symbol $[s]$ to denote the discrete representation of a low-level state s, and $[s]_{default}$ to represent the version of $[s]$ obtained by using the default assignment for P_S and the assignments in $[s]$ for all other atoms.

The following result presents sufficient conditions under which our approach solves any problem which is solvable under some evaluation of Skolem symbols.

Theorem 1. *Let D be a planning domain, U a set of typed constants, g a goal, and P_S the set of predicates in D that use Skolem symbols, such that $\langle D, U \rangle$ does not have dead-ends wrt g and D is uniform wrt g and P_S.*

If the low-level sampling routines for Skolem symbols are probabilistically complete, then for any low-level state s, if there exists a physically similar state s' such that $[s']$ is solvable by the high-level planner then the execution of Alg. 1 with inputs $[s]_{\text{default}}$ and D reaches the goal.

3.2 Low-Level Interpretation and Execution

Arbitrary sampling techniques could be used for interpreting Skolem functions. For efficiency, we used methods that searched for interpretations satisfying required conditions which cannot be achieved by high-level planning alone.

We drew upon the wealth of approaches developed in the robotics literature for solving navigation, motion planning and obstacle avoidance problems. In this work we assumed that the continuous planner has complete knowledge of the state of the world. This can obtained using, for example, 3D sensors (Hsiao et al. 2010) and map-making techniques (Marder-Eppstein et al. 2010). Both arm motion planning and base movements between poses (i.e. the *moveto* action) are performed using Rapidly Exploring Random Trees (RRTs) (LaValle and Kuffner Jr 2000).

Interpreting Grasping Pose Symbols A robot pose from which an object can be grasped (i.e, a grasping pose or one that satisfies *graspable*) must satisfy two conditions: the robot base should be at a collision free configuration from where the object is reachable, and there must exist a collision-free trajectory from this configuration to one that brings the manipulator into the grasping configuration. Grasping an object requires finding a manipulator configuration that allows the robot to robustly hold it. We precalculated and stored the grasping configurations for each object in OpenRave (these configurations can be computed on-the-fly if necessary).

Our approach for finding a grasping pose is inspired by Probabilistic Roadmaps (Kavraki et al. 1996), and summarized in Alg. 2. The first step in lines 2-3 is to generate a robot's base configuration that is within a reachability area around the object. The presence of obstacles, the non-linearity of the robot actuators and the high-dimensionality of the search space render this problem very challenging (Stulp, Fedrizzi, and Beetz 2009; Berenson, Kuffner, and Choset 2008). While pre-calculating poses using inverse kinematic (IK) caching could have been used, we found that sampling from a sphere centered around the object yields the same qualitative results. During sampling, poses outside the environment's boundaries or in collision with the environment are discarded. Collisions can be efficiently calculated by approximating the robot's shape to be either a cube or a cylinder. At line 3 we generate candidate values for the PR2's torso height and retrieve the manipulator grasping configurations at line 4.

The conditions for satisfying a grasping pose are checked in lines 6 & 7. Previously found grasping solutions are cached and retained until an object changes location or when the discrete planner requires so. If no valid solution is found within a maximum number of iterations then a failure is triggered and the violated preconditions listing all the removable obstructions is reported to the discrete planner.

Interpreting Symbols for *putDown* By substituting the grasping manipulator poses at line 5 with a set of manipulator poses that lay above a tabletop, Alg. 2 can be generalized to find base poses to put down objects on free areas of a sur-

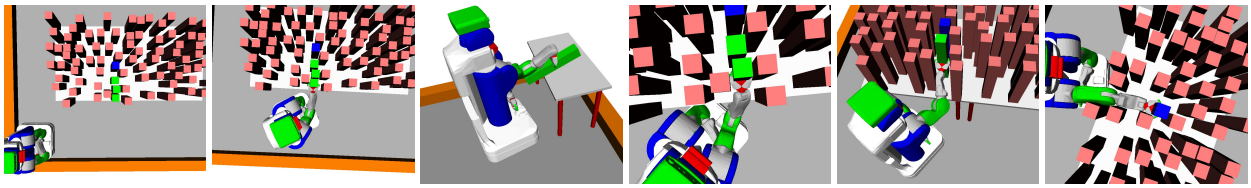

Figure 2: An example execution of a pick and place experiment. From left to right: 1) The initial disposition of the objects on the table. The blue object is the target, while the green objects are the ones identified as obstructing the target. 2) The robot selects the first object to remove. 3) The robot puts the object on a free area of an additional table. 4) The robot removes the second obstruction. 5) The robot removes the third obstruction. 6) The target object is finally reachable.

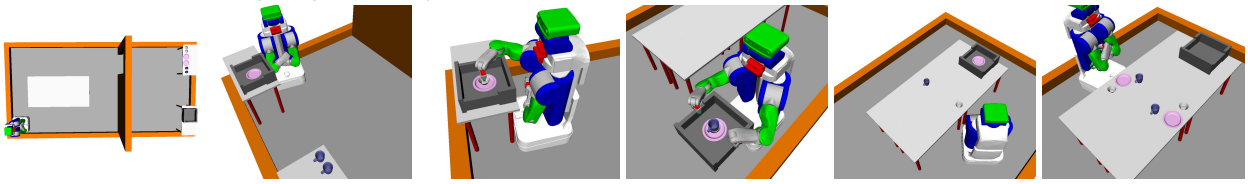

Figure 3: An example scenario of a dining set-up experiment. From left to right: 1) The initial configuration of the environment. 2) The robot places a plate on the tray. 3) Another plate and a bowl are placed on the tray. 4) The robot carries the tray with four objects to the dining table. 5) The first objects are removed from the top of the stack on the tray and placed on the appropriate locations on the table. 6) The final goal configuration is achieved.

face. Our current implementation does not take into account collisions between objects held by the robot and the environment. Therefore only obstacle free manipulator configurations are considered when looking for an empty area above a tabletop. Although our simulator provides a realistic implementation of robots' kinematics and collision detection, to speed up the computations we did not simulate physics. Therefore once the robot opens the gripper holding an object we forced the object to fall down and rest on the target surface in a horizontal position.

4 Empirical Evaluation

We tested our proposed approach in two different scenarios, simulated using OpenRave.[1] The first scenario involves grasping objects from a cluttered tabletop (Fig. 2). This problem requires the robot to identify objects that prevent a collision free trajectory to the target object, and moving them to a separate side table. The second scenario sees the robot setting up a dining table with three pairs of kitchen objects (Fig. 3). The robot can make use of a tray to help carrying more than one object, but there are constraints on how the objects can be stacked on the tray.

Our approach can be used with any classical planner. For the first experiment we used a well-established satisficing planner, FastForward (FF) (Hoffmann and Nebel 2001). The second experiment required a cost-sensitive planner. We used FastDownward (FD) (Helmert 2006) and always chose the second plan that it produced in its anytime mode if it was produced within 350s, and the first plan otherwise. When reporting the execution time we did not calculate the time required to actually perform an action, e.g. move the robot's base or the arms. All experiments were on an AMD Opteron dual-core 2GHz machine with 4GB RAM.

Pick and Place Scenario An instance of the pick-and-place problem is illustrated in Figure 2. To guarantee that objects in the middle of the table are not easily reachable we approximated their shape with over-sized bounding boxes. We modeled this problem as a realistic simulation, where the robot can make free use of the available empty space between objects to reach its goal. Thus a highly cluttered table with 80 objects may have obstructions from nearly every possible grasping pose, but the number of obstructions from each may be small. If grasping fails, the continuous planner returns a list of obstructions.

We performed 30 tests over 3 randomly generated environments. Each environment has the same configuration of static objects (tables and walls) but different configuration of movable objects. Each random environment has between 50 and 80 objects randomly placed on the tabletop. In these experiments we used 200 samples of base configurations generated while searching for grasping poses (outer loop of Alg. 2). Table 1 summarizes the results with averages over 10 runs. For each run we randomly selected a target object with at least one obstruction to be the grasping goal. As a baseline for comparison we used the time a classical planner would take to solve a version of the same problem with all obstructions precomputed and a discretized set of sampled configurations. For each object in the environment, we sampled 200 poses and kept only those from where objects are graspable. This resulted in 80 to 130 configurations for our problems. The last column of Table 1 shows the time required to pre-calculate all these obstructions, which itself is significantly higher than the time required for the computation and execution of the whole plan in our approach.

Dining Table Set-Up Scenario The second experiment illustrates the general applicability of our approach to heterogeneous problems. The robot's task is to move six objects from an initial location to a dining table (Fig. 3). The robot can only move objects between the "kitchen" and "dining" areas by placing them on a tray, moving the tray to the target table and then placing the objects from the tray to their desired configuration. Multiple round-trips are allowed. The robot had to transport two sets of plates, bowls and mugs.

Placing objects on the tray is an action that can fail, thus

[1] Source code is available at https://github.com/lorenzoriano/OpenRaving

#Objs	#Obstrns	Time(s)	#Replan	Precomp(s)
80	2.6	602	2.3	4245
65	2.0	228	2.6	3667
50	1.8	139	2.1	1777

Table 1: Results for the cluttered table scenario. The central 3 columns are averages of 10 independent runs, showing the number of obstructions, total time for planning and execution in our approach and number of calls made to the classical planner. For comparison, times for precomputing all obstruction facts are shown in the last column.

generating violated preconditions. This simulates dynamical instability between pairs of objects which is difficult to pre-compute without execution. The lower level enforced this by not allowing a plate to be stacked over other objects, and not allowing anything to be stacked over a mug. These conditions are initially unknown to the high-level planner. We also made the cost of moving across rooms 100 times the cost of other actions. As picking up the tray requires two-arm manipulation we pre-calculated both manipulators poses for grasping it. Grasping objects, placing them on the tray or on the table uses the primitives described in Sec. 3.2.

Solving the dining table scenario required 230s, including execution. Given the violations induced by stacking objects on the tray, the classical planner had been called 3 times. The solution plans typically attempt to stack a few objects on the tray and transport them in batches.

5 Related Work

This work is related to, and builds upon a vast history of research in planning and robotics. An alternate representation of our high-level problem would be a partially observable problem. However, this is a much harder problem in general (Rintanen 2004) and an offline contingent planning process would be unfeasible in our setting.

Various researchers have investigated the problem of combining discrete and continuous planning. Alami et al. (1998) describe a system architecture that uses a translation module for converting high-level actions into continuous trajectories. Volpe et al. (2001) also use a similar module, referred to as "time-line" interaction. However, these approaches do not discuss specific methods for compiling away continuous parameters of operators in order to facilitate discrete planning. Plaku et al. (2010) propose a sampling based approach for guiding the low-level search process with information from a high-level planner. They also allow non-deterministic actions. However, communication is only one-way: only the first action in the high-level plan is used, and only to bias low-level search process. In contrast, the low-level search process in our approach provides information back to the high-level planner which uses it to generate more relevant plans. Hauser (2010) observes almost the same problems in tasks like pick-and-place. His approach uses a STRIPS like language for specifying actions, but does not use a classical planner for the high-level search. Another approach that combines discrete and continuous planning is described by Choi et al. (2009). These authors propose including kinematics constraints among the information provided to the discrete planner. Their approach automatically generates logical actions from the output of a geometric motion planning algorithm. However this approach also has only one-way communication between the discrete and continuous planner. Moreover their approach is difficult to adapt to non-motion planning problems. Wolfe et al. (2010) use angelic hierarchical planning to define a hierarchy of high-level actions over primitive actions. However the high-level actions in their approach have continuous action arguments and need to be sampled. They also do not model problems with preconditions like obstructions which require geometric reasoning. Our approach also includes an angelic interpretation: the skolemized functions in our discrete, high-level operators are assumed to have *a* value that satisfies the preconditions, and is selected by the agent. Kaelbling et al. (2011) combine discrete and continuous planning by relying upon a very specific regression-based planner with task-specific components. Grasping objects in a cluttered environment is still an open problem in robotics. Dogar et al. (2011) propose an approach that replaces pick-and-place with pushing objects away from the desired trajectory. We will explore this solution to the reachability problem in future applications.

6 Conclusions

In this paper we presented an approach for planning in robotics that relies upon the synergy between a classical and a continuous planner. Our solution also addresses a broad problem in robotics: that the truth value of many predicates can only be found when the robot tries to perform an action. Both of our test scenarios featured this problem. It also enables a classical planner to efficiently represent these facts without dealing with representations that grow with the number of samples used in the lower layer. As our experiments show, this is significantly more efficient than precomputing discretized problems. While we conducted our main experiments in a pick-and-place scenario, the second experiment shows that our proposed approach is not limited to this particular application. Introducing costs, handling constraints and plans with heterogeneous actions are supported by our algorithm through its modularity. Although we simulated dynamical instability using fixed rules, a different implementation could utilize a physics simulator and prune away additional infeasible actions.

Future directions include a more general analysis of completeness and optimality, as well as applications to situations where action costs, in addition to facts, are made available during execution. While our simulation is extremely faithful in terms of the capabilities and limitations of the robot and in terms of the problem scenarios, the natural next step in this work will be to test the plans with a real robot.

Acknowledgments

We thank Malte Helmert and Joerg Hoffmann for providing versions of their planners with verbose outputs that were useful in the implementation. This work was supported by the NSF under Grant IIS-0904672.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal Of Robotics Research* 17:315–337.

Berenson, D.; Kuffner, J.; and Choset, H. 2008. An optimization approach to planning for mobile manipulation. *2008 IEEE International Conference on Robotics and Automation* 1187–1192.

Choi, J., and Amir, E. 2009. Combining planning and motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 238–244. IEEE.

Diankov, R. 2010. *Automated Construction of Robotic Manipulation Programs*. Ph.D. Dissertation, Carnegie Mellon University, Robotics Institute.

Dogar, M., and Srinivasa, S. 2011. A framework for push-grasping in clutter. *Robotics: Science and Systems VII*.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. Technical report, AI Center, SRI International. SRI Project 8259.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to pddl for expressing temporal planning domains. *J. Artif. Int. Res.* 20(1):61–124.

Hauser, K. 2010. Task planning with continuous actions and nondeterministic motion planning queries. In *Proc. of AAAI Workshop on Bridging the Gap between Task and Motion Planning*.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. of the 17th International Conference on Automated Planning and Scheduling*, 176–183.

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

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

Hsiao, K.; Chitta, S.; Ciocarlie, M.; and Jones, E. 2010. Contact-reactive grasping of objects with partial shape information. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 1228–1235. IEEE.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Proc. IEEE International Conference on Robotics and Automation*, 1470–1477.

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

LaValle, S., and Kuffner Jr, J. 2000. Rapidly-exploring random trees: Progress and prospects.

LaValle, S. M. 2006. *Planning algorithms*. Cambridge University Press.

Levesque, H. J.; Pirri, F.; and Reiter, R. 1998. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence* 2:159–178.

Marder-Eppstein, E.; Berger, E.; Foote, T.; Gerkey, B. P.; and Konolige, K. 2010. The Office Marathon: Robust Navigation in an Indoor Office Environment. In *International Conference on Robotics and Automation*.

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

Rintanen, J. 2004. Complexity of planning with partial observability. In *Proc. of the 14th International Conference on Automated Planning and Scheduling*, 345–354.

Stulp, F.; Fedrizzi, A.; and Beetz, M. 2009. Action-related place-based mobile manipulation. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* 3115–3120.

Talamadupula, K.; Benton, J.; Schermerhorn, P. W.; Kambhampati, S.; and Scheutz, M. 2010. Integrating a closed world planner with an open world robot: A case study. In *Proc. of AAAI*.

Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; and Das, H. 2001. The CLARAty architecture for robotic autonomy. In *Proc. of IEEE Aerospace Conference*, 121–132.

Wolfe, J.; Marthi, B.; and Russell, S. J. 2010. Combined task and motion planning for mobile manipulation. In *Proc. of the International Conference on Automated Planning and Scheduling*, 254–258.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *Proc. of the International Conference on Automated Planning and Scheduling*, 352–.