2011 IEEE International Conference on Robotics and Automation
Shanghai International Conference Center
May 9-13, 2011, Shanghai, China

Hierarchical Task and Motion Planning in the Now

Leslie Pack Kaelbling and Tomás Lozano-Pérez

Abstract— **In this paper we outline an approach to the integration of task planning and motion planning that has the following key properties: It is aggressively hierarchical; it makes choices and commits to them in a top-down fashion in an attempt to limit the length of plans that need to be constructed, and thereby exponentially decrease the amount of search required. It operates on detailed, continuous geometric representations and does not require a-priori discretization of the state or action spaces.**

I. Introduction

As robots become more physically robust and capable of sophisticated sensing, navigation, and manipulation, we want them to carry out increasingly complex tasks. A robot that helps in a household must plan over the scale of hours or days, considering abstract features such as the desires of the occupants of the house, as well as detailed models that support locating and getting out ingredients and cooking tools for preparing a meal. The complexity of such tasks derives from very long time horizons and large numbers of objects to be considered and manipulated. However, there are offsetting properties of such domains that can make planning in them tractable: there are few catastrophic or entirely irreversible outcomes, the geometry is not 'tight' in the sense that there is plenty of room to move objects out of the way, and strict optimality is not crucial.

Current symbolic task planners and geometric motion planners have complementary strengths. Task planners can reason over very large sets of states by manipulating partial descriptions, while geometric planners operate on completely detailed specifications of world states: a task planner could decide that the living room needs to be traversed, regardless of the detailed arrangement of its furniture. Motion planners deal beautifully with geometry, but not with non-physical aspects of the domain; they can plan how to get to the phone but not decide that a phone call needs to be made.

In this paper we outline an approach to integrating task planning and motion planning, with two key properties:

- It is aggressively hierarchical. It makes choices and commits to them, limiting the length of plans and exponentially decreasing the amount of search required.
- It operates in the domain of continuous geometry, and does not require any *a priori* discretization of the state or action spaces.

Hierarchy: Most work in hierarchical planning uses a hierarchical structure as a way to speed the construction of a complete low-level plan with guaranteed soundness or optimality conditions. Our goal is to design a system that can work effectively with non-determinism in the environment or in the low-level controllers. In such cases, planning in detail far into the future will typically be wasted, due to the inability to predict exactly what will happen. For this reason, we plan 'in the now': we construct a plan at an abstract level, commit to it, and then recursively plan and execute actions to achieve the first step in the abstract plan without constructing the rest of the plan in detail.

The risk associated with this approach is that the abstract plan might not be executable: the particular way that the first step is carried out could make it impossible to carry out subsequent steps, at least without undoing the results of earlier steps. We attempt to avoid such failures by constraining the abstract plan steps so that they are *serializable* [1]; that is, so that for any realization of the first plan step, there exist realizations for the subsequent ones. So, we simply execute the first abstract step, observe the resulting world state, and then plan in detail for the next one. This approach results in dramatic speed-ups from the hierarchical problem decomposition when serializability holds. If, for some reason, serializability fails, then we formulate an interleaved plan for achieving the effects of both steps; as long as actions in the environment are ultimately reversible, then any goal can be achieved, at the expense of sub-optimality in the behavior.

Continuous geometry: In complex, high-dimensional geometric spaces, it is crucial to avoid indiscriminate discretization. Purely geometric planners selectively construct discrete states from the problem description (e.g., vertices of configuration-space obstacles) or through a problem-driven sampling process. Task planners, on the other hand, operate over a set of instantiations of operator schemas, for every combination of possible values of the operator arguments. We handle the integration of continuous geometric planning with task planning by using geometric 'suggesters', which are fast, approximate geometric computations that construct appropriate choices for the parameters of an operator. For example, when determining the preconditions for moving an object from one room to another, the suggester can plan a path for a conservatively grown object in the 3D workspace and then establish a precondition that the swept volume of that path be free of obstacles.

II. Related Work

There is a great deal of work related to ours; we attempt to illustrate the main points of contact here.

Manipulation planning The problem of manipulation planning is to take a goal configuration of several objects, and

This work was supported in part by the NSF under Grant No. 0712012, in part by ONR MURI grant N00014-09-1-1051 and in part by AFOSR grant AOARD-104135. Thanks to the reviewers for insightful comments.

CSAIL, MIT, Cambridge, MA 02139 {lpk, tlp}@csail.mit.edu

generate a plan consisting of robot trajectories and grasping operations that will result in the desired configuration [2], [3]. Planning in hybrid spaces, combining discrete mode switching with continuous geometry, can be used to sequence robot motions involving different contact states or dynamics. Hauser and Latombe [4] have taken this approach to construct climbing robots.

Planning among movable obstacles generalizes manipulation planning to situations in which additional obstacles must be moved out of the way in order for a manipulation or motion goal to be achieved. In this area, the work of Stilman et al. [5], [6] takes an approach similar to ours, in that it plans backwards from the final goal and uses swept volumes to determine, recursively, which additional objects must be moved. Our solution to the problem of movable obstacles arises from a general regression-based symbolic planner, but, in the current implementation, is not guaranteed to find a solution whenever one exists.

Integrating symbolic and motion planning The need for integrating geometric and task planning has long been recognized [7]. In the work of Cambon et al. [8], a symbolic domain acts as a constraint and provides a heuristic function for a complete geometric planner. Plaku and Heger [9] extend this approach to handle robots with differential constraints and provide a utility-driven search strategy.

Hierarchical planning Hierarchical approaches to planning have been proposed since the earliest work of Sacerdoti [10], whose ABSTRIPS method generated a hierarchy by leaving off preconditions. Marthi et al. [11] give hierarchical domain descriptions semantics based on angelic non-determinism, and can dramatically speed up the search for optimal plans based on upper and lower bounds on the value of refinements of abstract operators. Goldman [12] gives an alternative semantics that incorporates angelic non-determinism during planning and adversarial non-determinism during execution. Nourbakhsh [13] suggests a hierarchical approach to interleaving planning and execution that is similar to ours, but does not integrate geometric reasoning. The work of Wolfe et al. [14] provides a hierarchical combined task and motion planner based on hierarchical transition networks (HTNs) and applies it to a manipulation-planning problem.

III. Example

Consider the domain shown in figure 1.1. The goal is for the object labeled A to be clean and put away in the storage room. The robot must take A, put it into the washer, wash it, and then move it to the storage room. Accomplishing this requires moving other objects. In this section, we describe informally how this problem is solved by our system.

The initial state is given as a three-dimensional geometric model (the figures here are shown looking down from above.) The goal is specified as a conjunction: $In(a, storage) \wedge Clean(a)$. A recursive process of planning and execution takes place, as shown in figure 2.

1. Blue nodes in the tree, labeled with numbers, denote planning problems. The first planning problem is the top-level goal, which is first addressed with abstract versions of the operators. Operators are abstracted by postponing preconditions; we make them progressively more concrete by requiring more preconditions to hold. For this goal, a two-step plan is made; it is shown as two descendant purple nodes, each of which represents an operation (the notation **Ai** means that the operator is at abstraction level **i**.) The plan is to run the washer with a in it, and then to place a into the $storage$ region. That plan is recursively executed, by planning for and executing each of its operations in turn. If an operation is a primitive action, then it is executed directly; otherwise, a subgoal is constructed, consisting of the conditions necessary to guarantee that the rest of the high-level plan will succeed, and a plan is made for that subgoal. Here, the abstract $Wash(a)$ operation is refined into the subgoal $Clean(a)$.

2. We now plan for the goal $Clean(a)$, generating a plan with two operations. The $Wash$ operator is considered more concretely, so we plan to satisfy its precondition, that a be in the washer, by an abstract $place$ operation that puts a into the washer.

3. We expand the first operation, planning to pick a up from its starting location and place it into the washer.

4. Now, we plan to satisfy the goal of holding object a. The resulting plan has two steps. The first requires that a swept volume of the robot moving to object a and picking it up be free. The swept volume is shown in figure 3.1 as a complex brown polygon; it was computed using a fast planner that considered only translations of the object, with a gripper attached to it, and of the robot base. It returns the union of the swept volumes of the base, the gripper, and the object. The second operation is a concrete $pick$ of object a.

5. The next subgoal now includes all of the preconditions for the $pick$ operation to succeed: the robot needs to be holding nothing, the swept volume for a needs to be clear, and object a should be in its starting place (that is the place for which the swept volume was computed; if a is moved, then clearing the swept volume we just computed will not necessarily suffice.) The resulting plan is comprised of abstract operations to remove both b and c from the swept volume. Because our cost model is still somewhat weak, it chooses to remove b first.

6. To remove b from the swept volume, a parking place, shown as PB in figure 3.1, is suggested. The suggestion is guaranteed not to conflict with picking a. The planner now determines that c is in the swept volume of b, and finds a parking place PC for it, as shown in figure 3.2. The plan is to $pick$ and $place$ c and then to $pick$ and $place$ b. The operation to pick c is refined into a primitive operation. At this point, a grasp location is selected and a robot motion planner (in this case, a simple RRT implementation) is called to plan the $pick$ operation. The primitive operation is **executed in the world**, which results in the robot grasping c. Then it plans to place c in the parking place and executes that plan, with the result shown in figure 1.2. Similarly, a detailed motion for moving b is planned and **executed in the world**, resulting in figure 1.3. We continue with the recursive execution of the

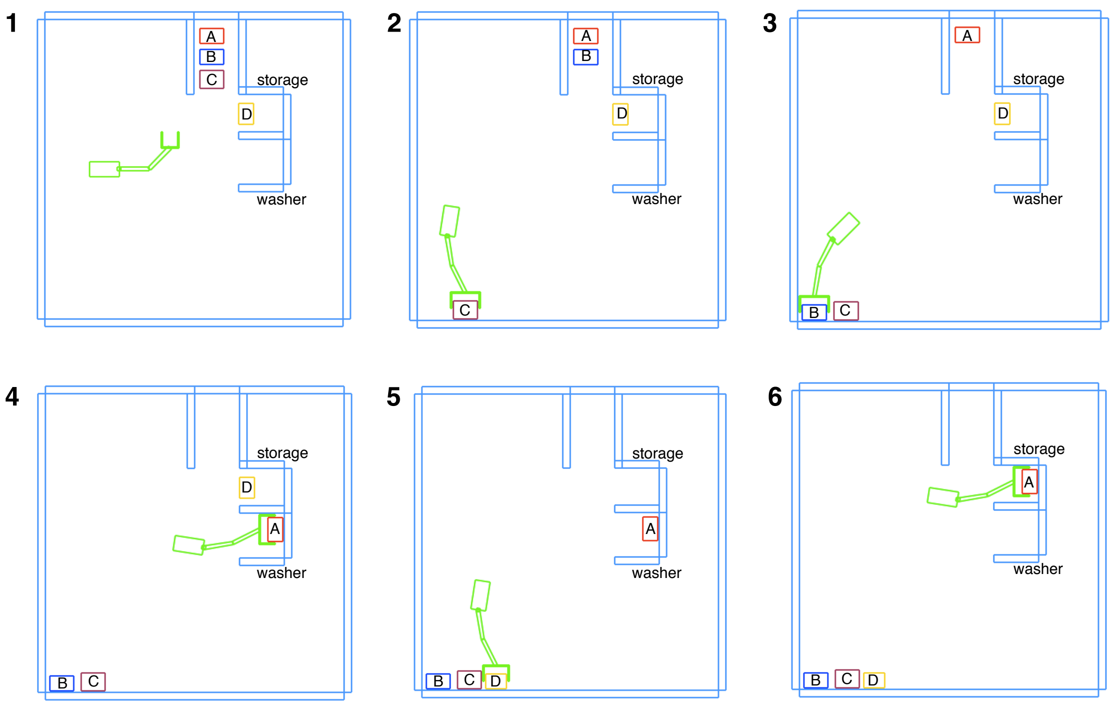

Fig. 1. Washing domain, in which the robot must move object A to the washing area, wash it, and put it in the storage area.

tree we have constructed. Note that executing the operator for removing c from the swept volume of a requires no further planning or execution, as the condition it was intended to establish has already been achieved as part of removing b.

7. Now, we plan and **execute** motions to $pick$ a and $place$ it in a location inside the washer, resulting in figure 1.4. The symbolic primitive $Wash$ is now **executed**, and the object a is clean.

8. We have come back to the root of the tree and now have the job of planning to put a in storage; notice that it is required that we maintain $Clean(a)$, which was established by the previous operation. This planning task is illustrative of the idea of *planning in the now*: The object a was placed in some particular pose inside the washer by the low-level geometric planning and execution system. We never had to simulate exactly where it would end up. Instead, we have actually executed it, and the planning problem in this step is solved with respect to a new starting state, corresponding to figure 1.4. We plan to $pick$ a from its new location in the washer (it is denoted aX in the figure, but internally to the planner it is the exact new geometric location of a) and then place it in storage.

9. Because there is nothing occluding the path from home to a's location in the washer, we need only **execute** the $pick$ primitive, resulting in the robot grasping a.

10. Now, we plan to place a in storage, and discover that there is an object in the swept path, so the plan consists of clearing the swept path and then placing a.

11. The only step required is to move d out of the new swept volume for a.

12. A parking place is suggested for d, shown as PD in figure 3.3, and we plan to move d there. The resulting plan first ungrasps a, then $pick$s and $place$s d, and finally regrasps a. Regrasping a is crucial in order to maintain the correctness of the high-level plan 10. We then plan and **execute** primitive motions to move d, resulting in figure 1.5. Finally, we plan and **execute** primitive motions to move a into storage, resulting in figure 1.6.

IV. Representation

Fluents The logical aspects of a domain are characterized using *fluents*. A fluent is a symbolic predicate applied to a list of arguments, which may be variables or constants. Variables begin with a capital letter. Constants can be names of objects or geometric specifications of regions of space. A fluent whose arguments are all constants is called a *ground fluent*. A state of the world determines the *value* of every ground fluent. Values are often Boolean, but need not be.

The fluents used to characterize the washing example are:

- $In(O, R)$: has value $True$ if object O is entirely contained in region R, otherwise $False$;
- $Overlaps(O, R)$: has value $True$ if object O overlaps region R, otherwise $False$;
- $ClearX(R, Os)$: has value $True$ if region R is is not overlapped by any object except those in the list Os, otherwise $False$;
- $Holding()$: has value $None$ if the robot is not grasping an object; otherwise the object being grasped; and
- $Clean(O)$: has value $True$ if object O is clean and otherwise $False$.

World States A *world state* is a completely detailed description of both the geometric and non-geometric aspects of a situation. World states can be represented in any way that is convenient. We never attempt to represent a complete

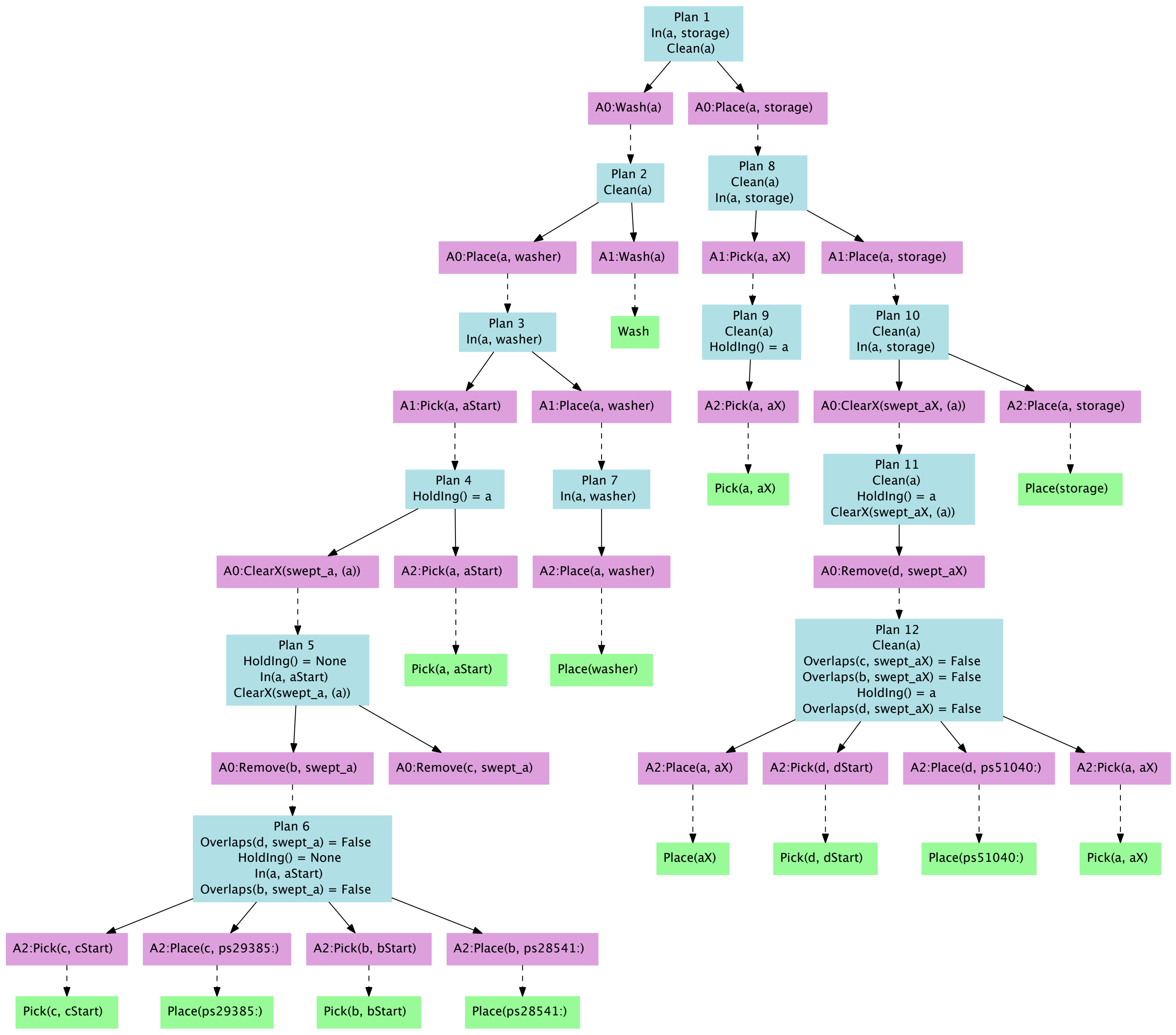

Fig. 2. Planning and execution tree for washing and putting away an object.

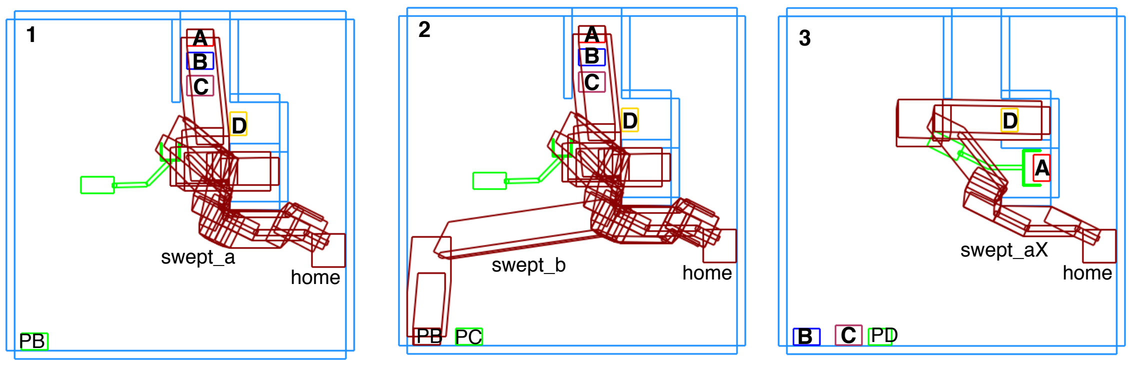

Fig. 3. Suggestions for swept paths and parking locations. The region $swept_b$ is clear in the starting state, so it never appears in the planning subgoals.

characterization of the world state in terms of fluents; this is important because the set of possible geometric regions that could serve as arguments to a fluent is infinite. The only requirement is that, for each fluent type, there is a procedure that will take a list of ground arguments and return the value of that fluent in the world state.

In our implementation, the world state is represented by a configuration of the robot and a set of objects, each of which has attributes including pose, shape (union of convex polyhedra that are extrusions in z), whether it is grasped by the robot, and whether it is clean.

Goals A goal for our planning and execution system is a set of world states, described using a conjunction of ground fluents. The goal of having object a to be clean and in the washer can be articulated as:

$$\mathit{In}(a, \mathit{washer}) = \mathit{True} \wedge \mathit{Clean}(a) = \mathit{True} \ \ .$$

During the course of regression-based planning intermediate goals are represented as conjunctions of fluents as well. We will write $s \in G$ to mean that world state s is contained in the set of goal states G.

Operators A planning domain is characterized by a set of primitive actions. In our formulation, we handle the lowest level of planning with a special-purpose geometric grasp and path planner for the robot. Thus, for the purposes of the rest of the planning, the primitives are actions that encapsulate the planning and execution of two primary operations: $\mathit{Pick}(O)$, which causes the robot to move to object O and pick it up; and $\mathit{Place}(R)$, which causes the robot to take the currently held object, possibly move to an open part of the space and regrasp it, and then move to an appropriate pose and ungrasp the object, guaranteeing that the object is entirely contained in region R. There is one additional primitive that has no geometric component: $\mathit{Wash}()$ simply causes the washing machine to be run, and any objects that are in the washer area will become clean.

Each of these operations is characterized by one or more operator descriptions. Each operator description can be used at multiple levels of abstraction: we begin by describing them in their most concrete form. In a discrete domain, we can define the operators in a STRIPS-style form:

$F(A_1, \ldots, A_n) = V$:
  exists: $B_1, \ldots, B_k$
  pre: $\phi_1, \ldots, \phi_m$
  sideEffects: $\psi_1, \ldots, \psi_l$
  prim: π
  cost: c

where $F(A_1, \ldots, A_n) = V$ is the *target* fluent, the A_i and V are variables or constants, the B_i are variables, the ϕ_i and ψ_i are fluents whose arguments are constants or variables that occur as As or Bs, π is a primitive action, and c is a positive real cost. This is an operator *schema* that stands for a whole family of ground operators, for all possible assignments of constant values in the domain to variables in the target fluent or in the **exists** list. The semantics of the operator description is that, if the primitive action π is executed in any world state s in which all of the ϕ fluents hold, then the resulting world state will be the same as s, except that any fluent mentioned as the target fluent or a side effect will have the value specified by those fluents.

To operate in infinite domains, we augment the standard operator descriptions with the following features:

Suggesters, which are procedures that map current start and goal states, and bindings of other variables, to restricted domains for existential variables. This significantly decreases the branching factor and increases the likelihood that serialization will succeed, by making intelligent choices of bindings for free variables. A given set of suggesters is applicable to any domain described by the same set of geometric operators and general robot type, e.g. an arm on a mobile base.

Procedural operator definitions, which map variable bindings into lists of preconditions, side effects, new bindings, and costs. These allow us to call the suggesters flexibly, depending on which variables are bound, and to generate lists of preconditions and side-effects whose length varies depending on the situation.

Inferential attachments, which are two types of procedures attached to fluents. The *entails* attachment of fluent ϕ computes whether ϕ logically entails another fluent ϕ'. In goal regression, when applying an operation to a goal g, the goal fluent and any side effect fluents are always removed from g; in addition, we remove any fluents in g that are entailed by the goal fluent or one of the side effects. The *contradicts* attachment of fluent ϕ computes whether ϕ logically contradicts another fluent ϕ'. In goal regression, if any of the condition or side-effect fluents contract a fluent in the goal, then the operation is not considered. *Non-deterministic side effects*, which model abstract actions with non-deterministic effects by setting the value of side-effect fluents to be *None*, indicating that the resulting value is unknown.

Following are operator descriptions used in our example domain. The numbers preceding the preconditions refer to the abstraction level; some irrelevant preconditions have been omitted for clarity. The descriptions refer to **goal**, which is the current regression subgoal in the planning process to which this operator is being applied and to $\mathbf{start}$ which is the current world state at the time this particular planning problem is being solved, which is not necessarily the initial state for the entire planning and execution problem.

The *pick* operation results in the robot holding object O:

$\mathit{Holding}() = O$:
  define: $\mathit{Ts} = \{T : \mathit{ClearX}(T, X) \in \textbf{goal} \wedge O \notin X\}$
  exists: $L \in \{\mathit{Location}(O, \mathbf{start}),$
      $\mathit{SuggestParking}(O, \mathit{Ts}, \mathbf{start})\}$
    $P \in \mathit{SuggestPaths}(O, L, \mathit{home}, \mathbf{start})$
  pre: 0. $\mathit{Holding}() = \mathit{nothing}$
    0. $\mathit{In}(O, L) = \mathit{True}$
    2. $\mathit{ClearX}(\mathit{sweptVol}(P), [O]) = \mathit{True}$
  sideEffects: $\forall L'.\mathit{In}(O, L') = \mathit{False}$
  prim: $\mathit{Pick}(O)$

Other variables that determine the behavior of this operation are L, which is the location of O when it is to be picked up, and P, which is a path that the robot can traverse from a pose in which it is grasping the object, back to its home location. The preconditions to this operation are that the robot not be holding anything, that O be in location L, and that the swept volume of path P be clear of all objects except O. Once these preconditions are satisfied, then the primitive pick operation can be planned in detail and executed by the fully geometric part of the planner.

The main subtlety here is in picking values of L and P. A standard symbolic planner enumerates all possible values of free variables, and then rules out instantiations later if they conflict with other aspects of the current planning goal. Because these variables both have infinite domains in our setting, we cannot enumerate them. Instead, we use procedures to suggest values that are likely to (a) not conflict with the current planning goal and (b) ensure that the preconditions of the operation can be effectively serialized and that this operation serializes well with other operations.

To avoid immediate conflicts, we look in the current goal, to find "taboo" regions Ts that must be kept clear, and then consider two different bindings of variable L: either it is the object's location in the current true world state, or it is a "parking" place, suggested to be not overlapping with the taboo regions, nor with other objects in the starting state. To guarantee that the robot can move to the object at that location and pick it up, we find a path from the robot's home location to L. Each of our operations guarantees that there is a free path for the robot to move to and from its home location: this condition guarantees that the robot will not destroy serializability by blocking itself in, but it does not constrain the geometric planner, when it is called in the now to generate the robot's actual path, to go through the home location. Note that the taboo regions are only taboos for placing objects: they must be kept free, but the robot may take paths that move through them. The place operation is very similar:

$\mathit{In}(O, R) = \mathit{True}$:
- **define:** $Ts = \{T : \mathit{ClearX}(T, X) \in \mathbf{goal} \wedge O \notin X\}$
- **exists:** $P \in \mathit{SuggestPaths}(O, R, \mathit{home}, \mathbf{start})$
- **pre:** 1. $\mathit{Holding}() = O$
  2. $\mathit{ClearX}(\mathit{sweptVol}(P), [O]) = \mathit{True}$
- **sideEffects:** $\mathit{Holding}() = \mathit{nothing}$
- **prim:** $\mathit{Place}(R)$

The operator for clearing a region has no primitive action. It is a definition of what it means for a region to be clear, articulated in the preconditions. The preconditions require that each object X that is in the domain, but not the list of exceptions, not overlap with R.

$\mathit{ClearX}(R, \mathit{Os}) = \mathit{True}$:
- **pre:** 1. $\forall X \in \mathit{Objects} - \mathit{Os} : \ \mathit{Overlaps}(X, R) = \mathit{False}$
- **prim:** none

The operator that causes an object O not to overlap a region R is also definitional, with no primitive action. It requires O to be in a location L that does not overlap the region R and that L be kept clear of other objects; it suggests values of location L that do not conflict with regions that are required to be clear, as well as with locations of objects in the starting configuration.

$\mathit{Overlaps}(O, R) = \mathit{False}$:
- **define:** $Ts = \{T : \mathit{ClearX}(T, X) \in \mathbf{goal} \wedge O \notin X\} \cup \{R\}$
- **exists:** $L = \mathit{SuggestParking}(O, Ts, \mathbf{start})$
- **pre:** 1. $\mathit{In}(O, L) = \mathit{True}$, $\mathit{ClearX}(L, [O]) = \mathit{True}$
- **prim:** none

Finally, we have a simple operator to make an object clean, which requires that the object be located in the washer in order for running the washer to make it clean.

$\mathit{Clean}(O) = \mathit{True}$:
- **pre:** 1. $\mathit{In}(O, \text{WASHER})$
- **prim**: $\mathit{Wash}()$

Hierarchy Inspired by Sacerdoti's [10] approach to constructing a planning hierarchy, we specify a hierarchy by postponing consideration of some or all preconditions of an operator. Each precondition in an operator definition is labeled with a level of abstraction. Increasingly detailed levels of abstraction are determined by requiring increasingly higher-numbered conditions.

Consider an operator description with target fluent r:
- **pre:** $p_1, \ldots, p_n$
- **prim:** o

By postponing precondition p_n, we effectively create a new operator description:
- **pre:** $p_1, \ldots, p_{n-1}$
- **prim:** achieve p_n maintaining $p_1, \ldots, p_{n-1}$; o

It is not generally true that this new operator description will hold in the original domain. There are two potential problems. First, it may not be possible to make p_n true without undoing $p_1, \ldots, p_{n-1}$. In this case, the planner will achieve $p_1, ..., p_n$ in whatever way it can, and then execute o and r will be achieved; the potential problem is suboptimality, in that wasted work will have been done to achieve conditions that will have to be undone.

Second, the side-effects of the abstracted operator will generally be different from the side-effects of the single primitive, because the process of achieving p_n may have additional effects. Our formalism for describing abstracted versions of the operators allows different side effects to be specified at different levels of abstraction. To model the inability of the plan at the abstract level to determine which particular realization of an abstract plan step will take place, we allow side effects to be non-deterministic and extend our planning algorithm to compute pre-images appropriately.

The degree to which abstract operators can be serialized depends on how the domain is formalized. For example, if it is important that the object not be regrasped as part of the *Place* operation, it is possible to 'expose' the choice of grasp at a higher level of abstraction in the formalization, and pass it down to both the *Pick* and *Place* operations to achieve the desired coordination. If at attempt at serializing operations at an abstract level fails, then the planning problem is addressed again with no abstraction, and solved with no further attempts at serialization.

V. Algorithms

Interleaved planning and execution A relatively standard regression-planning algorithm, based on an A* search that works backward from the goal, generating sub-goals that are the weakest precondition of the goal under each applicable action, is used to solve single planning sub-problems. The architecture can be thought of as doing a depth-first traversal of a planning tree, and is implemented as a recursive algorithm, as shown below. The planning and execution system is invoked by calling HPN$(currentState, goal, operators, absLevel, world)$, where $currentState$ is a description of the current state of world; $goal$ is a conjunction of fluents describing a set of goal states; $operators$ is a set of hierarchical operator descriptions; $absLevel$ specifies, for any ground fluent, the number of times it has served as a plan step in the HPN call stack above it; and $world$ is an actual robot or a simulator in which primitive actions can be executed. In this paper $world$ is a geometric motion planner coupled with an execution capability.

```
HPN($currentState, goal, operators, absLevel, world$):
    if holds($goal$, $currentState$):
        return TRUE
    else p = PLAN($currentState$, $goal$, $operators$, $absLevel$)
        for $(o_i, g_i)$ in p
            if prim($o_i$):
                $currentState$ = $world.execute(o_i)$
            else HPN($currentState$, $g_i$, $operators$,
                    NEXTLEVEL($absLevel$, $o_i$), world)
```

The PLAN procedure is called with the current state, the goal, a set of operators, and an abstraction level. It returns a list $((o_1, g_1), ..., (o_n, g_n))$ where the o_i are operator instances, $g_n = goal$, g_i is the weakest precondition of g_{i+1} under o_{i+1}, and $\mathbf{start} \in g_0$. We use goal regression because it computes, for each plan step, the weakest conjunctive subgoal that can serve as the target for the planning problems at the next level down. In our implementation, $absLevel$ is a dictionary, mapping ground fluents to numeric abstraction levels. Whenever we descend, recursively, to address a new planning problem, the NEXTLEVEL procedure increments the abstraction level associated with that fluent.

Suggesters The operator definitions in our domain use two suggesters: *SuggestPaths* and *SuggestParking*. These suggesters are constructed using some additional suggesters: *SuggestGrasps*, *SuggestPoses* and *SuggestPathsTo*.

$SuggestGrasps(O)$: finds grasps for O (gripper poses relative to O) with sufficient overlap of the fingers and an available approach configuration of the robot.

$SuggestPoses(O, R, Taboos)$: finds a set of poses for O where it is completely inside region R, there is no collision with taboo regions, and there is some valid grasp (as per $SuggestGrasps$) for the object in that pose. The implementation generates poses in the region and discard those that fail that grasping accessibility tests.

$SuggestPathsTo(O, R)$: finds paths for O from the robot's home pose to some pose within region R (as per $suggestPoses$). A motion planner lazily builds a 4-dof visibility-graph; x, y translation constraints are represented as C-space polygons for discrete ranges of z and θ. Links in the visibility graph represent either pure x, y translation, z offset or θ offset. In the examples in this paper, it was constrained to do translation only and to return a single path.

$SuggestParking(O, Taboos, \mathbf{start})$: find an "out of the way" location for O that does not overlap any of the regions in $Taboos$. The implementation currently is simply $SuggestPoses$ in some designated parking regions; the parking places are not pre-specified; they are suggested using some criteria that tends to pack them near the edges of the specified region.

The actual motion planning in the base domain can be done by any motion planner that can plan paths between specified robot configurations. Our implementation uses an RRT-based planner.

VI. Correctness

Because HPN is a combined planning and execution system, the standard notions of correctness and completeness from planning do not apply directly. Our correctness criterion is that, if a goal state was reachable from the starting state under some sequence of operations, that HPN will eventually cause the system to reach a goal state.

If we were in a discrete domain, using every instantiation of the operator schemas and did not introduce any hierarchical planning levels, then it is clear that the regression planner would produce a correct plan if one exists and it would be executed step by step. So, we need to examine the effects of hierarchy and of operating in infinite domains on the ability of HPN to achieve feasible goals.

Hierarchy Given a hierarchy of operator descriptions for each target fluent, in a finite domain, we can construct an overall hierarchy of planning domain descriptions (PDDs), as follows. Let $\mathcal{F}$ be the set of all ground instances of all fluents that can be the target of an operator in any of the PDDs, and let $\mathcal{H}$ be a mapping from $\mathcal{F}$ into integers selecting the level of abstraction of that fluent's operator. We specify a partial ordering on abstraction levels as follows: $\mathcal{H}_1 > \mathcal{H}_2$, that is, $\mathcal{H}_1$ is more concrete than $\mathcal{H}_2$, iff (1) for all fluents $f \in \mathcal{F}$, $\mathcal{H}_1(f) \geq \mathcal{H}_2(f)$ and (2) there exists a fluent $f \in \mathcal{F}$ for which $\mathcal{H}_1(f) > \mathcal{H}_2(f)$. We define $\mathcal{H}^*$ to be the abstraction level that maps each fluent to its completely concrete operator and $\mathcal{H}_0$ to be the abstraction level that maps each fluent to its most abstract operator.

A state s has *static connectivity* in a domain, if all states s_1 that are reachable from s are also reachable from any s_2 that is reachable from s. That is, any choice of action is ultimately 'reversible' in the sense that it can be undone through a sequence of actions. A domain has static connectivity if all of its states do.

Theorem: If (1) the PDD specified by operators ops at abstraction level $\mathcal{H}^*$ is a complete and correct formalization of the primitive actions of domain w, (2) $\mathtt{start}$ has static connectivity in that domain, and (3) G is reachable from $\mathtt{start}$, then executing HPN$(start, G, ops, \mathcal{H}\prime, w)$ will cause world w to be in a state $s \in G$.

Proof: On each recursive call to HPN, the *absLevel* argument is increased for one fluent; it will eventually be completely concrete for all fluents, causing a correct plan made of primitive actions to be constructed and executed.

The very last invocation of PLAN is on the goal G at abstraction level $\mathcal{H}^*$ in some state s that was reached after taking some sequence of actions previously selected by the HPN procedure. Under the static connectivity assumption, $s$tart is reachable from s, and so, there is a plan from s to G if there was one from $start$ to G. This plan will be constructed and then executed by HPN, driving w into a state in G.

Infinite domain: Because our domains are infinite, we cannot consider all instantiations of the operations. Our current implementation of suggesters only considers a small number of possible instantiations of the operations. We could recover the relatively weak properties of probabilistic completeness by having the suggesters be generators of an infinite stream of samples, and managing the search as a non-deterministic program over those streams.

VII. Empirical Results

We have applied the HPN method to several different configurations of objects in the simple domain of figure 1. In addition, we constructed a larger 'household' domain with 6 connected rooms, a 'vacuum', a 'mop', and several randomly placed 'junk' objects. In the household domain, we wish to make the floors of all the rooms clean, which requires vacuuming and mopping them (by putting the mop or vacuum in the room and executing a non-geometric operation) and also moving any junk that might be in the room into the closet. This larger domain requires addition of operations for mopping and vacuuming as well as for moving between rooms of the house.

In each of these domains, we ran the HPN algorithm; we report the number of plans that were made (Num), the length of the longest of these plans (Longest), the total number of steps executed (Steps). In these particular problems, the plans produced have no or few redundant steps. The planner uses a non-deterministic implementation of priority queue, so different executions of the algorithm make different serialization decisions, generating slight variations in the number and length of subplans; the numbers shown here are a representative example.

Domain	Num	Longest	Steps
swap	22	4	8
wash	14	4	13
wash all	26	6	22
clean house	89	4	36
clean and tidy	169	7	65

The swap problem involves interchanging the locations of two blocks. It is the classic example of non-serializable goals; it also requires careful geometric planning. The hierarchy does not really help, but the planner still finds an answer. The rest of the domains require a mix of geometric and symbolic reasoning, including moving objects to locations that aren't specified in the goal (temporarily out of the way, or into a closet). To the best of our understanding, none of the existing mixed task and motion planners could solve these problems.

Generally speaking, the search process is exponential in the length of the plan. What we can see is that the hierarchy is having a huge impact, especially in the larger 'household' domains, allowing us to solve many small problems instead of a single large one, which would be completely impractical for most of these problems.

VIII. Conclusions and Future Work

This paper has outlined a general strategy for hierarchical planning and execution for task and motion planning. It has the potential for dramatic speedups, but relies on good domain-dependent choices in selecting a hierarchical formalization and in suggesting a small set of plausible values from an infinite set of operator bindings. We hope to be able to apply learning algorithms to improve these choices over time. The HPN architecture is very well suited to re-planning approaches to stochastic domains; based on the outcome of executing a primitive, a decision could be made to re-plan at any level above that step. We will apply the replanning approach to uncertainty in outcomes, and extend the approach to apply to belief spaces in partially observable domains.

References

[1] R. E. Korf, "Real-time heuristic search: First results," in *AAAI*, 1987.

[2] T. Lozano-Perez, J. Jones, and E. Mazer, "Handey: a robot system that recognizes, plans and manipulates," in *ICRA*, 1987.

[3] R. Alami, J.-P. Laumond, and T.Siméon, "Two manipulation planning algorithms," in *WAFR*, 1994.

[4] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *IJRR*, vol. 29, pp. 897–915, 2010.

[5] M. Stilman and J. J. Kuffner, "Planning among movable obstacles with artificial constraints," in *WAFR*, 2006.

[6] M. Stilman, J.-U. Schamburek, J. J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *ICRA*, 2007.

[7] S. Kambhampati, M. R. Cutkosky, M. Tenenbaum, and S. H. Lee, "Combining specialized reasoners and general purpose planners: A case study," in *AAAI*, 1991, pp. 199–205.

[8] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *IJRR*, vol. 28, 2009.

[9] E. Plaku and G. Hager, "Sampling-based motion planning with symbolic, geometric, and differential constraints," in *ICRA*, 2010.

[10] E. Sacerdoti, "Planning in a hierarchy of abstraction spaces," *Artificial Intelligence*, 1974.

[11] B. Marthi, S. Russell, and J. Wolfe, "Angelic semantics for high-level actions," in *ICAPS*, 2007.

[12] R. P. Goldman, "A semantics for HTN methods," in *ICAPS*, 2009.

[13] I. Nourbakhsh, "Using abstraction to interleave planning and execution," in *Third Biannual World Automation Congress*, 1998.

[14] B. Marthi, S. Russell, and J. Wolfe, "Combined task and motion planning for mobile manipulation." in *ICAPS*, 2010.