Hierarchical Task and Motion Planning in the Now *

Leslie Pack Kaelbling and **Tomás Lozano-Pérez**
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
32 Vassar St., Cambridge, MA, USA

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. Importantly, our approach also limits the need to project the effect of actions into the far future. It operates on detailed, continuous geometric representations and partial symbolic descriptions. It does not require a complete symbolic representation of the input geometry or of the geometric effect of the task-level operations.

Introduction

As robots become more robust and capable of sophisticated sensing, navigation, and manipulation, we will want them to carry out increasingly complex tasks over long time horizons. 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, or what time the UPS delivery is likely to arrive, down to detailed geometric reasoning in support of putting objects away in cupboards or washing dishes. Such long-term planning requires integration of task and motion planning.

The strength of symbolic task planners is their ability to reason over very large sets of states by manipulating partial descriptions, for example, all the possible states in which "the robot is in the kitchen and the blue chair is in the kitchen". However, these task planners work by enumeration: all possible operations are considered in a state, in order to expand it in the search. In geometric domains, enumeration of possible operations and complete symbolic description of states is difficult or impossible, depending on selected vocabulary and desired resolution.

Motion planners, on the other hand, deal beautifully with geometry, but not with abstract features of the domain; they can plan how to get to the phone but not decide that a phone call needs to be made. Motion planners also have limited ability to deal with partially specified states. They do not compute paths for the robot from the kitchen to the living room without having to know where all the furniture is.

*This work was supported in part by the National Science Foundation under Grant No. 0712012.

In this paper we outline an approach to 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. Importantly, our approach also limits the need to project the effect of actions into the far future.
- It uses goal regression, constructing partial symbolic descriptions of desired subgoals and making queries in a continuous geometric representation of the initial state. It does not require a complete symbolic representation of the input geometry or of the geometric effect of the task-level operations.

Hierarchy Most work in hierarchical planning uses the hierarchy as a kind of search heuristic: it can speed the construction of a complete low-level plan with guaranteed soundness or even optimality conditions. Our proposal is more aggressive: we will sacrifice optimality in exchange for a method that solves a small planning problem at a high level of abstraction, commits to the plan, solves a problem of achieving the first subtask in that plan, commits to that plan, and so on, until primitive actions are reached and executed.

This aggressive approach to hierarchical planning has several consequences:

- All planning problems are short-horizon and therefore efficient. Planning cost is the minimum of an exponential in the horizon and a polynomial in the size of the state space; we expect to be working in domains in which the state spaces are very large or infinite, meaning that decreasing the horizon is crucial.
- Subtasks can be serialized by propagating constraints across a planning level and down to lower planning levels.
- Detailed forward progression of the effects of actions is not necessary. When, for example, it is time to make the detailed plan for the second subtask of a more abstract plan, we will already have executed the first subtask, and can plan "in the now," conditioned on the actual state of the world. This property is useful even in deterministic domains, but becomes crucial when world dynamics are uncertain.

- The robot will sometimes embark on a plan, and begin executing it, only to discover that some or all of the early steps were mistaken. Our assumption is that most actions are reversible without huge cost. In the case of subtasks that involve irreversible or highly expensive actions, we would invoke a more complete planning algorithm before beginning execution.

Symbolic and geometric planning We use a goal-regression planner, which starts with a symbolic representation of the goal, and works backwards, constructing symbolic descriptions of subgoals, until all of the conditions in a subgoal hold in the initial state. The initial state is represented geometrically and can support any query about the truth of a condition. This planning structure allows 'new' geometric entities, such as regions of interest in task space, to be constructed during the planning process, and does not require a complete *a priori* enumeration of salient locations or objects in advance. Because the domain of objects is not specified in advance, we cannot use standard STRIPS add and delete lists to characterize the effects of actions; we augment the planning approach with additional geometric reasoning capabilities.

We handle the integration of continuous geometric planning with enumerative task planning by using geometric 'suggesters', which are fast, approximate geometric computations that help the high-level processes make appropriate choices. For example, it is possible to determine which objects need to be moved out of the way by planning a path for a conservatively grown object in the 3D workspace rather than in the high-dimensional configuration space of the robot.

This paper outlines a basic framework for hierarchical planning in the now, and provides very preliminary demonstrations in mobile manipulation-planning tasks of moderate complexity. It is currently only applied in deterministic domains, but we have designed the framework to be extended to domains with uncertainty in both state and dynamics.

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 generate a plan consisting of robot trajectories and grasping operations that will result in the desired configuration (Lozano-Perez, Jones, and Mazer 1987; Alami, Laumond, and T.Simon 1994; Koga and Latombe 1994). 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 (Hauser and Latombe 2009) have taken this approach to construct climbing robots.

Planning among movable obstacles generalizes manipulation planning to situations in which additional obstacles may need to 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. (Stilman and Kuffner 2006; Stilman et al. 2007) 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 framework treats the problem of movable obstacles in the context of a general regression-based symbolic planner. In the current implementation, it does not consider sufficiently many object placements to be complete.

Integrating symbolic and geometric planning In the work of Cambon et al. (Cambon, Alami, and Gravot 2009), a symbolic domain acts as a constraint and provides a heuristic function for a complete geometric planner. Plaku and Heger (Plaku and Hager 2010) extend this approach to handle robots with differential constraints and provide a utility-driven search strategy. Choi and Amir (Choi and Amir 2009) solve the problem of hand-constructing symbolic representations of geometric states and actions of interest by constructing a roadmap of the geometric space and extracting salient features to construct a symbolic domain description.

Hierarchical planning Hierarchical approaches to planning have been proposed since the earliest work of Sacerdoti (Sacerdoti 1974), whose ABSTRIPS method generated a hierarchy by leaving off preconditions, in a way similar in spirit to our method. Marthi et al. (Marthi, Russell, and Wolfe 2007) have developed a framework that gives hierarchical domain descriptions real semantics, and can dramatically speed up the search for optimal plans based on upper and lower bounds on achievability or value that are specified for abstract operators. Our work is similar in spirit, but sacrifices optimality quite aggressively for efficiency.

Nourbakhsh (Nourbakhsh 1998) suggests a hierarchical approach to interleaving planning and execution that is similar to ours and runs experiments on a real mobile robot, but with no detailed geometric reasoning. The work of Wolfe et al. (Marthi, Russell, and Wolfe 2010) provides a hierarchical combined task and motion planner based on hierarchical transition networks (HTNs) (Nau et al. 2003) and applies it to a manipulation-planning problem.

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. To do this, the robot must take A, put it into the washing room, wash it, and then move it to the storage room. Accomplishing this requires moving other objects. In the following, we describe informally how this problem is solved by our system.

- The domain is formalized using *fluents*, which are like logical predicates, to describe its symbolic aspects. The fluents are *In*(*object*, *region*), *Overlaps*(*object*,*region*), *Clean*(*object*), and *ClearX*(*region*,*objects*). The last means that the region is clear *except* for certain objects.
- Possible operations are described using a generalization of STRIPS planning rules. The operations are *PickPlace*(*obj*, *startRegion*, *targetRegion*), *ClearX*(*region*, *objects*), *RunWasher*(*object*), and *Remove*(*object*, *region*).

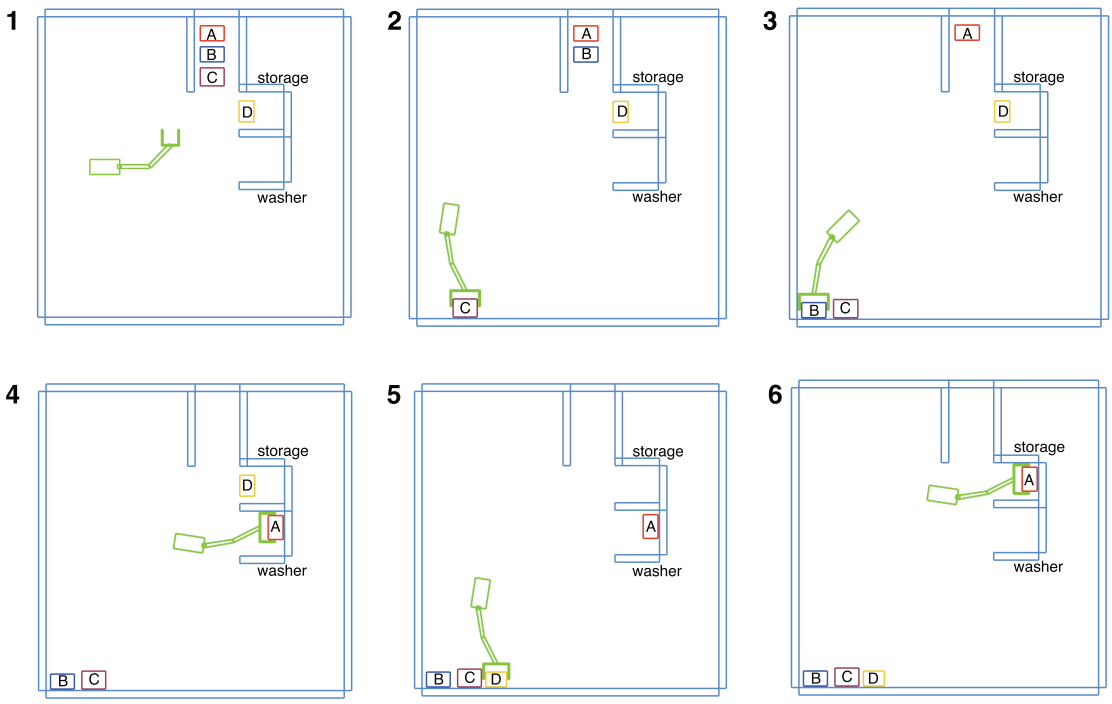

Figure 1: Washing domain, in which the robot must move object A to the washing area, wash it, and put it in the storage area.

- A starting state is determined, as a geometric model, shown in 1.1. This domain is three-dimensional, and the figures here are shown looking down from above.
- The goal is specified as a conjunction of fluents: $In(a, Storage) \wedge Clean(a)$.
- A recursive process of planning and execution is begun; the entire process is shown as a tree in figure 2.
- **1.** Blue nodes in the tree, labeled with numbers, denote planning problems. The first planning problem is the top-level goal. It is first addressed with abstract versions of the subtasks, for which it is not required to make the preconditions true. In this case, a two-step plan is made; it is shown as two descendant purple nodes, each of which represents a subtask. The plan is to run the washer with a in it, and then to place a into the *Storage* region.
- Now, that plan is recursively executed, by executing each of its subtasks in turn. If a subtask is a primitive domain action, then it is executed directly; otherwise, it is *refined*. A subtask's refinement is typically another goal to be planned for and achieved. Here, the abstract $runWasher(a)$ subtask is refined into the goal $Clean(a)$.
- **2.** We now plan for the goal $Clean(a)$, and generate a plan with two subtasks. Because it has the abstract *RunWasher* as an ancestor, this time, the *RunWasher* subtask is considered concretely, and its precondition, that a be in the washer, is satisfied by a preceding pick-and-place abstract subtask that puts a into the washer.
- **3.** We execute the first subtask, which causes us to plan to put a into the washer. The resulting plan has two subtasks. The first abstractly requires that a swept volume that results from moving the object a, as well as the robot, from its current location to a location in the washer, via a home location, is 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. The second subtask is a concrete pick and place operation on a, which can take place once that swept volume is cleared.
- **4.** Now, we plan concretely for clearing the swept volume, while maintaining that a is in its starting location (if it gets moved elsewhere in the process, then the swept volume may no longer be adequate); this condition will be present in all goals of this subtree. The resulting plan is comprised of abstract operations to remove both b and c from the swept volume. Because we do not yet have good cost estimates for abstract actions, the planner decides to remove b first.
- **5.** To remove b from the swept volume, a parking place, shown as *PB* in figure 3.1, is suggested. The suggestion is made to guarantee that it will not conflict with moving 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 move c and then to move b.
- The subtask to move c is refined into a primitive subtask. At this point, a grasp location is selected and a robot motion planner (in this case, a simple RRT implementation)

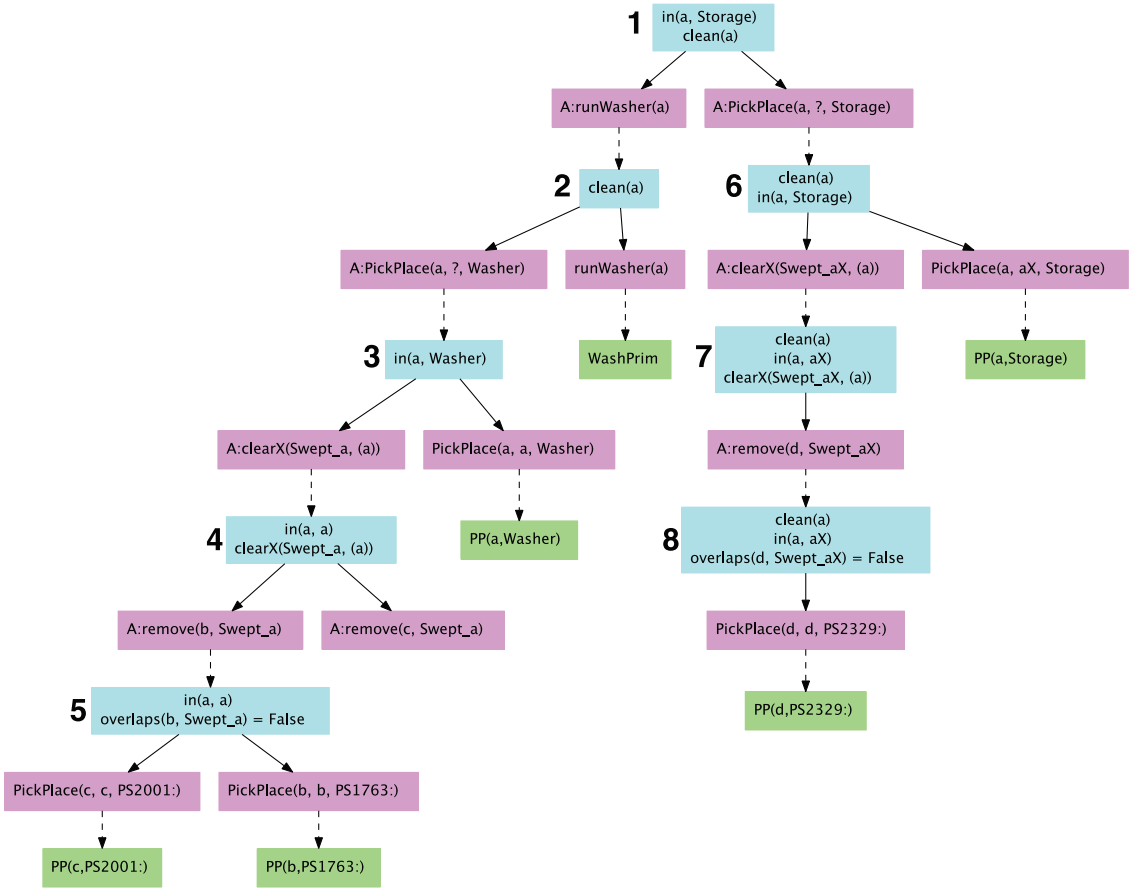

Figure 2: Planning and execution tree for washing and putting away an object. Dashed arrows are subtask refinements.

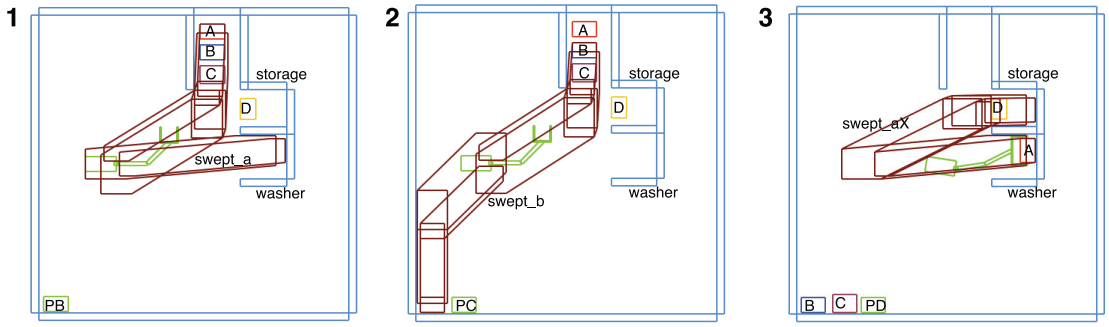

Figure 3: Suggestions for swept paths and parking locations.

is called to plan both phases of the pick and place operation. **The primitive operation is executed in the world**, 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 tree we have constructed. Executing the subtask for removing c from the swept volume of a requires no further work, as the condition it was intended to establish has already been done as part of removing b.
- Now, we plan and **execute in the world** motions to move a to a location inside the washer, resulting in figure 1.4.
- The symbolic primitive *runWasher* is now **executed in the world**, and the object a is clean.
- **6.** 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 subtask. 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 pick-and-place planner. 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 executed with respect to a new starting state, corresponding to figure 1.4.

  So, with aX being the pose of a inside the washer, we find a path and corresponding swept volume that would move it to the storage region, and plan to clear that new swept volume, then move a to storage.
- **7.** The only step required is to move d out of the new swept volume for a.
- **8.** A parking place is suggested for d, shown as DP in figure 3.3, and we plan to move d there.
- We 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.

Hierarchical Planning

Now, we describe the system somewhat more formally.

Representation

Any planner that has to reason about geometric and non-geometric properties of the world requires a hybrid representation. It is tempting to try to make a complete symbolic summary of the geometric state of the world. That is ultimately quite difficult when, for example, the planner needs to reason about different regions of space that are specified dynamically or constructed during the process of planning.

For this reason, we only represent the world state that is current at the time of planning in complete geometric detail. During planning, the goal "state" (which is actually a symbolic specification of a set of satisfactory states in terms of both geometric and non-geometric properties) and intermediate states are specified symbolically, but include geometric predicates that constrain the underlying physical states of the world.

The goal, as well as intermediate planning states, is represented as a conjunction of *fluents*. A fluent is a symbolic predicate, such as *In(obj, region) = True*, whose arguments may be variables or constants. Constants can be names of objects or geometric specifications of regions of space. Fluents have several procedural attachments, to facilitate both geometric and symbolic reasoning:

- *test*: a procedure that, given ground values of the arguments, can be applied to the current geometric world state, and returns *True* or *False*.
- *contradicts*: a procedure that takes another ground fluent and returns *True* if they contradict one another and *False* otherwise.
- *entails*: a procedure that takes another ground fluent and returns *True* if this fluent logically entails the other one, and *False* otherwise.

We encode the knowledge of the preconditions and effects of operations in a set of *subtask* descriptions. Subtasks are not organized into a rigid hierarchy, but may refer to one another as needed. A subtask is specified by the following components:

- *target fluent*: A single fluent which is the 'desired effect' of the subtask. The arguments of the fluent are variables.
- *variables*: The subtask variables include the variables in the target fluent as well as existential variables whose value is chosen during the planning process.
- *preconditions*: A procedure that maps bindings of the target-fluent variables into a conjunction of fluents that describe the set of states in which executing this subtask will cause the target fluent to be true. This procedure also takes the current geometric state and the symbolic state to which the operator is being applied as input, so that it can make choices of existential variables that are unlikely to generate contradictions.
- *side-effects*: A list of fluents describing additional effects of executing this subtask; fluents may have an unspecified value, which signifies that the subtask may change the value of the fluent, in an undetermined way. It is important to be able to have an incomplete effects model at more abstract levels of the hierarchy.
- *constraints:* A list of fluents that should be true when this subtask is executed and which must be true when it terminates. These are derived during the planning process and passed backward and down the plan tree.
- *refinement*: A procedure that maps bindings of the subtask variables, the current geometric world state, and current constraints into a refinement of this subtask at the next lower hierarchical level: a refinement can be a primitive action or a conjunctive goal specification (which will require subsequent planning).

Subtask definitions for our domain are provided in section .

Constructing an operator hierarchy

Inspired by Sacerdoti's (Sacerdoti 1974) approach to constructing a planning hierarchy, we also build our hierarchy on the idea of postponing consideration of some or all preconditions of a subtask. In the current implementation, we first consider a subtask completely abstractly, ignoring its preconditions during planning, and assuming they can be made true when it is time to plan for and execute the subtask. When the abstract subtask is executed, a new plan is made, taking all of its preconditions into account. In future implementations, we would have multiple levels of abstraction and some additional reasoning about how abstractly to consider a subtask each time it is encountered.

Planning by goal regression

Our starting state is encoded in complete geometric detail, in the form of an actual world that we can measure or a highly accurate geometric model. Our goal is specified by a conjunction of symbolic fluents. There are generally three choices in designing the search process for a planner: forward search over sequences of operations from the starting state, backward search over sequences of operations from the goal, or a more general search in the space of plans. Plan-space search can be very difficult to guide heuristically, so we opt for a search over operation sequences.

It is difficult to compute a complete symbolic representation of the initial state, which would be necessary to support fully symbolic forward-search planning. So, we perform goal regression, starting from a symbolically-encoded goal (which will, in general, contain fluents with geometric content). The search works backward from the goal to other symbolic precursor conditions, until it reaches a symbolic condition that holds in the geometric representation of the start state; the *test* attribute of each fluent allows it to be evaluated in the geometric model.

Our planner is a relatively standard goal-regression planner, with one important exception. In standard goal regression, states of the search are conjunctive goal conditions. A state is expanded (backwards) by computing, for each possible operator, the weakest precondition of the state under the operator: that is, what would have to be true at some time t in order for the execution of that operator to make the state true at time $t+1$. Our difficulty is that we cannot enumerate all possible operators: the sets of poses and grasps for objects and paths between locations are infinite. During regression we apply *geometric suggesters*: procedures that take the current geometric reality and the state that we desire to achieve and suggest bindings of existential variables in the subtask descriptions, such as paths and grasps, based on a fast, approximate motion planner.

For example, consider a case where the goal condition specifies that some object a be in a particular region. A subtask of moving a into that region can make that condition true; but the preconditions of the subtask will require a decision about where that object is to be moved *from*. Traditional symbolic planners enumerate all possible such locations, but in general geometric spaces, this is impossible. It is crucial, and often sufficient alone, to consider the current geometric starting pose of the object. Another useful type of location to consider is a "parking place": a location that is relatively out of the way of the objects involved in the goal. We discuss our particular set of geometric suggesters in more detail in section .

Unless the suggesters are well informed about the context of their suggestions, there is a risk that the suggestions will generate contradictions with other aspects of the goal, and therefore be rejected. So, we adopt the strategy of moving part of the test into the suggester: the suggesters are given the context of the fluent that is to be achieved (that is, the other conjuncts in the current regression goal) and are asked to guarantee that the variable bindings they suggest will be compatible with the rest of the goal condition.

This mechanism can be used to manage resources more generally: a subtask can "reserve" a resource by adding a precondition that requires the resource to be available before the subtask is executed. Other subtasks that may be added to the plan will be able to observe the reservation and respect it while making their own choices.

We define the planning domain as follows:

- **initial state**: The goal condition, represented as a conjunction of ground fluents.
- **termination condition**: A procedure that takes a state of the search, which is a conjunction of ground fluents, and returns *True* if all of the fluents hold in the initial geometric state and *False* otherwise.
- **successor function**:
  - Given a state, find all subtasks whose target fluent can be matched with a fluent in the state.
  - For each such subtask, generate one or more ground instances by suggesting values for existential variables. If an abstract version of the subtask is *not* an ancestor of this problem in the planning/execution tree, then consider this subtask abstractly by ignoring its preconditions in the next step.
  - Compute preconditions and side-effects for each ground subtask.
  - If neither the preconditions nor the side-effects are in contradiction with any other fluents in the state, then construct a successor state from the conjunction of all fluents from the original state and all of the preconditions, with the target fluent and any other fluents that are entailed by the precondition removed.
  - Annotate any ground subtask that is used to generate a legal successor state with any fluents that occur both before and after the execution of that subtask: those fluents are maintenance constraints that are passed down to the expansion and execution of the subtask.
  - Return a set of pairs of (subtask, successorState) so constructed.

The planning procedure, then, is:

PLAN(*startState*, *goal*):

A^* search in the space defined above
heuristic(s): num fluents in s not true in *startState*

Hierarchical planning and execution

The regression-planning algorithm is used to solve single planning subproblems within the overall hierarchical planning and execution architecture. The architecture can be thought of as doing a depth-first traversal of a planning tree, and is implemented as a recursive algorithm. Because the hierarchical structure is not uniform (it may be deeper in some parts of the tree than others) the process is framed in terms of *doing* jobs, dispatching on the type of the job to be done. Jobs can be of the following types:

- *Primitive*: an action that may require further geometric-only planning, but that requires no further symbolic planning. Examples include putting an object in a location, or turning on a washing machine. This is a leaf of the hierarchical planning process, which ultimately results in a change in the real physical world and in the model that is being used by the planner.
- *State:* symbolic description of a desired world state, given as a conjunction of fluents.
- *Sequence:* ordered list of subtasks.
- *Subtask:* A step in a plan. Can be refined in the current world into a primitive job, a sequence of jobs, or a state to plan for.

The algorithm is as shown below. The planning and execution system is invoked by calling do($job, world$), where *job* is the highest level goal for the system and *world* is a queryable representation of the world (either the world itself or a high-fidelity model). The last case is entered and the regression planner called to make a plan p. That plan is then executed, by recursively executing each of its subtasks in sequence. If a subtask's fluent has been serendipitously made true by a previous step, then it requires no further action.

DO($job, world$):
  if type(job) == PRIMITIVE:
    then EXECUTE(job, $world$)
  elseif TYPE(job) == SUBTASK $\wedge\neg$ $holds(job.fluent, world)$:
    then DO($job.refinement(world)$)
  elseif TYPE(job) == SEQUENCE:
    then for *task* **in** $job.tasks$:
      do DO(*task*, $world$)
  else ; TYPE(job) == STATE:
    $p \leftarrow plan(\text{STATE}([], world), job)$
    if p:
      then DO(p, $world$)
      else Raise *failed*

Domain description

In order to develop intuition for this hierarchical planning representation and algorithm, we present the formalization of a simple domain in which a robot can move objects around in the world, and wash them by putting them in a special "washer" location.

The fluents in this domain are: $In(O,R)$, meaning that object O is entirely contained in region R; $ClearX(R,Os)$, meaning that region R is clear except for the list of objects Os; $Overlaps(O,R)$, meaning that object O overlaps region R, and $Clean(R)$, meaning that object R is clean.

The primitives in this domain are $PickAndPlace(O,R)$, which causes the robot to move object O from its current starting pose into a pose such that the object is entirely contained in region R; and $RunWasher()$, which simply causes the washing machine to be run.

Here is the definition of the subtask of moving an object O into a region R. It begins by looking in the constraints that apply to it, to find "tabu" regions Ts that it must keep clear; then it considers two different bindings of variable L, which is the location *from* which the object will be moved. 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 tabus, nor with other objects in the starting state. It also suggests one or more possible paths that O might move through in order to get from L to R. The preconditions to doing the move, then, are that O be in the starting location L, and that the swept volume of path P be clear of all objects except O. Once these preconditions are satisfied, then the subtask may be refined to a pick-and-place operation to be executed by the fully geometric part of the planner.

PICKPLACE(O,R):
  pre:
    define: $Ts \leftarrow \{T : ClearX(T,X) \in constraints\}$
    exists: $L \in \{Loc(O,start), SuggestParking(O,Ts,start)\}$
      $P \in SuggestPaths(O,L,R)$
    $In(O,L), ClearX(sweptVol(P),[O])$
  ref: $PickAndPlace(O,L,R)$

The subtask for clearing a region has no refinement. It is, essentially, a definition of what it means for a region to be clear, which is articulated in the preconditions. It finds all objects $X \notin Os$ that overlap the region of interest R and creates a list of preconditions requiring that each of those objects X not overlap with O. In addition, it requires that no additional objects be put into the region.

CLEARX(R, Os):
  pre:
    define: $Xs \leftarrow \{X : Overlaps(X,R,start) \in w \wedge X \notin Os\}$
    forall: $X \in Xs : \neg Overlaps(X,R)$
    $ClearX(R, Os \cup Xs)$
  ref: none

The following operator causes an object O not to overlap a region R. Like the previous operator, it is definitional, and has no refinement. Like pick-and-place, it starts by finding a set of tabu regions that are constrained to be kept clear (excluding those that would allow O to remain in them); it then asks a geometric suggester for a region P, to 'park' this object in. It will attempt to find and return such a region that is reachable from the robot's current position and that does not overlap any tabu regions. The preconditions, then, are that that parking region remain clear except for O and that O be in P.

REMOVE(O,R):

pre:
define: $Ts \leftarrow \{T : ClearX(T,X,constraints) \in state \wedge O \notin X\}$
exists: $P \leftarrow suggestParking(O,Ts,start)$
$In(O,P), ClearX(P,[O])$
ref: none

Finally, we have a simple subtask to make an object clean, that articulates the geometric precondition that the object be located in the washer.

WASH(O):

pre: $In(O,\text{WASHER})$
ref: $RunWasher()$

Correctness

Our goal in this work is to design a planning and execution system that can solve extremely long-horizon planning problems. It is well known that solving such problems exactly is intractable, so it is not reasonable to expect that we will get something for nothing. We hope that our system will solve the vast majority of planning problems reasonably well: probably not optimally, but not ridiculously. It will depend the environment being relatively benign, and an expectation that problems posed will not be puzzles.

This approach rests two major assumptions:

- Conservative preconditions on abstract operations can be computed efficiently and correctly.
- Subgoals are often serializable.

The first assumption is embodied in the geometric suggesters: it is their job to perform a 'quick and dirty' computation to suggestion how we will want to move objects through space or to select an 'out of the way' location to put something. Our current approach to the suggesters is entirely heuristic: we have implemented them directly, based on a visibility-graph planner. However, it is our vision that the suggesters could actually learn appropriate behavior from experience: every time a plan is successfully or unsuccessfully executed, it constitutes training examples for the suggesters, which could eventually come to learn to make good suggestions in related situations.

In the current system, there is a risk that, if the suggesters are too liberal, we will plan successfully at the high level but fail at geometric planning time. Although it is not implemented, it would be straightforward to detect such failures and invoke the detailed geometric planner in place of the suggester to get a feasible suggestion.

The second assumption is embodied in the abstract handling of subtasks. When we decline to consider the detailed preconditions of a subtask, we are implicitly asserting that the subtask is independent of other subtasks: that is, it can be achieved in a way that does not depend on the method by which any of the other subtasks at that level are achieved. When this is true, it makes planning nearly trivial: a small set of subtasks must be selected and ordered at the high level, and can be planned for completely independently.

Completeness and suggestions This planner is not complete, in the sense that it considers only finitely many suggestions in an infinite domain. If the suggesters were extended to the form of 'generators', that could be called repeatedly to generate new suggestions, ultimately via statistical sampling or systematic enumeration, we conjecture that the planner would be complete.

Completeness and serializability: Our approach is aggressive about treating subtasks as being serializable: this allows very fast planning when subtasks are, in fact serializable. What happens when they are not? We make the following conjecture: When primitive actions are reversible, then this planning and execution algorithm is complete with respect to any necessary interleaving of subtasks.

Rather than providing a formal proof, we illustrate this property in an example problem containing only blocks a and b, configured as they are initially in the washing domain. The goal is to swap the locations of the two blocks. The planning and execution tree is shown in figure 4. The process operates as follows:

- At the highest level, an arbitrary serialization of the steps of putting each of the objects, a and b, in its target region is chosen. The locations from which the objects will be moved into the goal locations are, as yet, unbound and shown as '?'.
- A detailed plan is made for putting a in its destination location, consisting of clearing a path from a's starting location to some location in its destination region, and then executing a pick-and-place operation.
- Clearing the path for a requires causing object b not to overlap with the path.
- A 'parking place' for b is suggested, and a plan is made to move b from its starting location to its parking location.
- **Object b is actually moved to a parking place**.
- **Object a is then actually moved to its goal location**.
- Now, it is time to refine the step of putting b in its goal location; the planner adds the regression condition that a be in its target region.
- Because it is impossible to put b in its target location without moving a, a new plan is made, involving three pick-and-place operations, that results in a correct final configuration.

Had the subgoals been completely serializable, then the planning process would have been extremely efficient, and no extra actions would have been taken. But, because serialization fails in this case, the planner is able to fall back on doing general purpose regression to generate an interleaved plan.

Geometric level

Our implementation uses two simple motion planners. As a basis for the geometric suggesters, we use a planner that focuses on the motions of objects, and for executing primitive operations, we use a traditional planner that focuses on the motions of the robot (potentially with object in hand). Our figures are showing two projected views of what is, in fact, a three-dimensional domain.

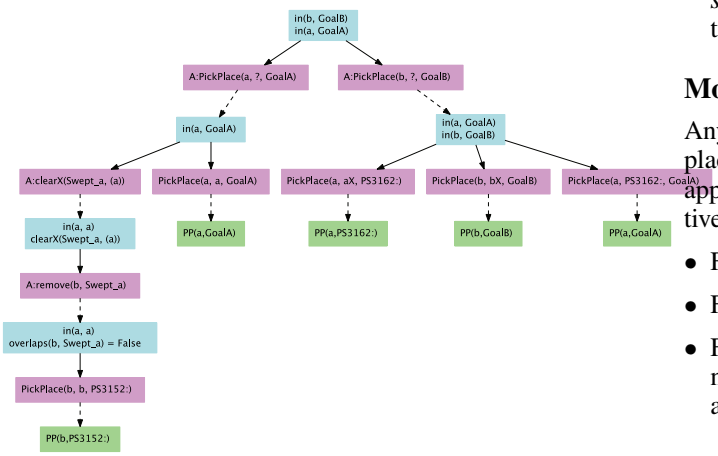

Figure 4: Planning and execution tree for swapping the position of two objects with strong ordering constraints.

Geometric suggestions

The subtask definitions in our domain use two suggesters: *suggestPaths* and *SuggestParking*. These suggesters are constructed using some additional suggesters: *suggestGrasps*, *suggestPoses* and *suggestPathTo*.

- *suggestGrasps*(O) – find grasps for O (gripper poses relative to O) with sufficient overlap of the fingers and an available approach configuration of the robot. The implementation enumerates pairs of faces to generate candidate grasps and discards those that fail the accessibility tests.
- *suggestPoses*$(O,R,Tabus)$ – find a set of poses for O where it is completely inside region R, there is no collision with tabu 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.
- *suggestPathTo*(O,R) – find a path from O's current pose to some pose within region R (as per *suggestPoses*). An enlarged volume that includes space for the gripper to approach the object is used as the moving object. The implementation uses a motion planner that lazily builds a 4-dof visibility-graph during the search for a path. 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.
- *suggestPaths*(O,R_1,R_2) – find a region such that having that region clear enables the robot to reach the object O and move it from some location in R_1 to some location in R_2. This is implemented with two calls to the *suggestPathTo* suggester, one to R_1 and the other to R_2. The result region is the swept-volume of the enlarged object along the paths.
- *suggestParking*$(O,Tabus)$ – find an "out of the way" location for O that does not overlap any of the regions in *Tabus*. The implementation currently is simply *suggestPoses* in some designated parking regions; in future, the parking regions should be chosen dynamically.

Motion planner

Any approach to finding concrete plans for single pick-and-place operations can be used as the geometric level of this approach. The motion planner for the pick-and-place primitives is asked to:

- Find poses in regions, which it does using *suggestPoses*.
- Find grasp poses, which it does using *suggestGrasps*.
- Find robot paths, which it does using an RRT-based planner in 8 dof (3 dof of base, 4 dof of hand relative to base, and grip opening).

Example results

The plan for washing block a requires 6 primitive actions. A flat symbolic planner would have required significant search to find the plan; a geometric planner in the full configuration space could never have started. Here, we solved 8 small planning problems, the biggest of which required a two-step plan, and also solved many simple motion plans for suggestions. Finally, we solved detailed robot-motion planning problems for each primitive action separately.

The web site
`http://people.csail.mit.edu/tlp/hierarchicalVideos/`
contains movies of the robot solving the swap and wash examples, as well as several more complex problems. In all of these cases, we find a considerable decrease in planning horizon, which comes with an exponential decrease in the size of the space to be searched.

Conclusions

This paper hints that a strong hierarchical planning and execution approach may be feasible for solving very large, long-horizon problems in complex (but not intricate or dangerous) domains. It will have to be augmented with reasoning about present and future uncertainty, and with the ability to refine and acquire the planning models at every level of abstraction, to be of real future use.

References

Alami, R.; Laumond, J.-P.; and T.Simon. 1994. Two manipulation planning algorithms. In *WAFR*.

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

Choi, J., and Amir, E. 2009. Combining planning and motion planning. In *ICRA*.

Hauser, K., and Latombe, J. 2009. Integrating task and prm motion planning: Dealing with many infeasible motion planning queries. In *ICAPS09 Workshop on Bridging the Gap between Task and Motion Planning*.

Koga, Y., and Latombe, J.-C. 1994. On multi-arm manipulation planning. In *ICRA*, 945–952.

Lozano-Perez, T.; Jones, J.; and Mazer, E. 1987. Handey: a robot system that recognizes, plans and manipulates. In *ICRA*.

Marthi, B.; Russell, S.; and Wolfe, J. 2007. Anglic semantics for high-level actions. In *ICAPS*.

Marthi, B.; Russell, S.; and Wolfe, J. 2010. Combined task and motion planning for mobile manipulation. In *ICAPS*.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An HTN planning system. *J. Artif. Intell. Res. (JAIR)* 20:379–404.

Nourbakhsh, I. 1998. Using abstraction to interleave planning and execution. In *Proceedings of the Third Biannual World Automation Congress*.

Plaku, E., and Hager, G. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *ICRA*.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*.

Stilman, M., and Kuffner, J. J. 2006. Planning among movable obstacles with artificial constraints. In *WAFR*.

Stilman, M.; Schamburek, J.-U.; Kuffner, J. J.; and Asfour, T. 2007. Manipulation planning among movable obstacles. In *ICRA*.