Combining Motion Planning with an Action Formalism

Jaesik Choi and **Eyal Amir**
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{jaesik,eyal}@cs.uiuc.edu

Abstract

Robotic manipulation is important for real, physical world applications. General Purpose manipulation with a robot (eg. delivering dishes, opening doors with a key, etc.) is demanding. It is hard because (1) objects are constrained in position and orientation, (2) many non-spatial constraints interact (or interfere) with each other, and (3) robots may have multi-degree of freedoms (DOF). In this paper we solve the problem of general purpose robotic manipulation using a novel combination of motion planning and an action formalism (Situation Calculus). Our approach integrates motions of a robot with other actions (non-physical or external-to-robot) to achieve a goal while manipulating objects. It differs from previous, hierarchical approaches in that (a) it considers kinematic constraints in configuration space (CSpace) together with constraints over object manipulations; (b) it automatically generates high-level (logical) actions from a CSpace based motion planning algorithm; and (c) it decomposes a planning problem into small segments, thus reducing the complexity of planning.

Introduction

Algorithms for general purpose manipulations of daily-life objects are still demanding (e.g. keys of doors, dishes in a dish washer and buttons in elevators). It was shown that planning with movable objects is P-SPACE hard (Chen and Hwang 1991; Dacre-Wright, Laumond, and Alami 1992; Stilman and Kuffner 2005). Nonetheless, previous works examined such planning in depth (Likhachev, Gordon, and Thrun 2003; Kuffner and LaValle 2000; Kavraki et al. 1996; Brock and Khatib 2000; Alami et al. 1998; Stilman and Kuffner 2005) because of the importance of manipulating objects. The theoretical analysis gave rise to some practical applications (Alami et al. 1998; Cortés 2003; Stilman and Kuffner 2005; Conner et al. 2007), but general purpose manipulation remains out of reach for real-world-scale applications.

Motion planning algorithms have difficulty to represent non-kinematic constraints despite of its strength in planning with kinematic constraints. Suppose that we want to let a robot push a button to turn a light on. CSpace[1] can represent such constraints. However, the CSpace representation could be (1) redundant and (2) computationally inefficient because CSpace is not appropriate for compact representations. It could be redundant, because it always considers the configurations of all objects beside our interests (i.e. a button and a light). Moreover, mapping such constraints into CSpace would be computationally inefficient, because mapping a constraint among n objects could take $O(2^n)$ evaluations in worst case. Thus, most of motion planning algorithms assume that such mappings in CSpace are encoded.

AI planning algorithms and description languages (e.g. PDDL (McDermott 1998)) have difficulty to execute real-world robots despite of its strength in planing with logical constraints. Suppose that we have a PDDL action for 'push the button' which makes a button pushed and a light turned on. However, the PDDL description could be (1) ambiguous and (2) incomplete (require details). Given a robot with m joints, it is ambiguous how to execute the robot to push the button, because such execution is not given in the description. Instead, it assumes that there is a predefined action which makes some conditions (e.g. a button pushed) satisfied whenever precondition is hold and the action is done.

Both methods solve this problem in different ways. Motion planning algorithms use abstractions to solve this problem. AI plannings use manual encodings. Although abstraction provides solutions in a reasonable amount of time in many applications, abstraction lose completeness. Thus, it has no computational benefit in worst cases. Although AI plannings have no need to search the huge CSpace, it requires manual encodings which are not only error-prone but also computationally inefficient.

We solve this problem with combining a motion planning and an AI planning in a model. We extend our previous framework in PDDL (Choi and Amir 2009) into Situation Calculus which provides logic formalisms. That is, Situation Calculus reflecting kinematic constraints are extracted from a graph constructed by a resolution-complete motion planning algorithm. In detail, our algorithm is composed of three subroutines: (1) extracting a graph from a motion planner, (2) building new actions from abstract actions using the built graph, (3) finding a solution in the built action theory, and (4) decoding it into CSpace.

In detail, our algorithm unifies a general purpose (logical) planner and a motion planner in one algorithm. Our algorithm is composed of three subroutines: (1) extracting logical actions from a motion planner, (2) finding an ab-

[1] CSpace is the set of all possible configurations

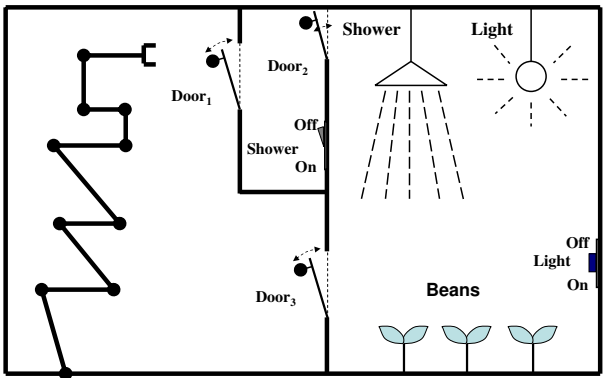

Figure 1: This figure shows an example of manipulating objects with a robotic arm. The goal is to take care of beans in a glasshouse. Beans require water and light everyday. The robot will provide water and light for beans. To accomplish this goal, the arm needs to manipulate objects such as doors and switches.

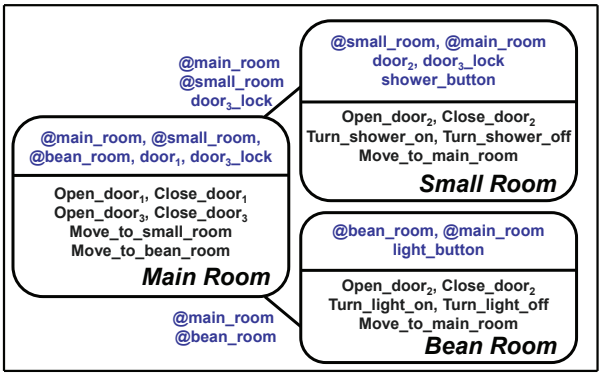

Figure 2: This is a possible tree decomposition for the toy problem of figure 1. The shared propositions appear on edges between subgroups. For example, a proposition ('$@door_3\_lock$') is shared by two subgroups ('Main Room' and 'Small Room') because the proposition is used by actions of two subgroups (respectively '$Open(Close)\_door_3$' and '$Turn\_shower\_on(off)$'). The theory is decomposed into small groups based on the geometric information (eg. the configurations of the room).

stract plan from the logical domain, and (3) decoding it into CSpace. It extracts Situation Calculus actions (McCarthy and Hayes 1987; Reiter 2001) from a tree constructed by a motion planner in CSpace. Then, it combines extracted actions with a given BAT (Basic Action Theory explained in Section) that has propositions, axioms (propositional formulae) and abstract Situation Calculus actions. To find an abstract plan efficiently, we automatically partitioned the domain by a graph decomposition algorithm before planning. In the planning step, an abstract plan is found by a factored planning algorithms (Amir and Engelhardt 2003; Brafman and Domshlak 2006) which are designed for the decomposed domain. In decoding, a motion plan is found from the abstract plan.

Section provides a motivational example. Section explains our encoding to build a theory in Situation Calculus. Sections and show our algorithm. Section presents related works. Finally, section provides experimental results followed by the conclusion in section .

A Motivating Example

Figure 1 shows a planning problem. The goal is to provide water and light to beans. The robotic arm should be able to manipulate buttons in the spatial space to provide water and light. There are also non-spatial constraints. At any time either the $shower$ is off or $door_3$ is closed or both.

The planner requires both a general purpose (logical) planner and a motion planner. It requires general purpose planner because the arm needs to revisit some points of CSpace several times in a possible solution. The way points may include '$Open\_door_1$', '$Close\_door_1$', and '$Turn\_light\_on$'. Note that the internal state (values of propositions) can be different, whenever the robot revisits the same point in the CSpace. It is certainly motion planning problem because the kinematic constraints of the arm should be considered. For example, the arm should not collide with obstacles, although the hand of the arm may contact objects.

Hierarchical planners have been classical solutions for these problems. A hierarchical planner takes in charge of high level planning. A motion planner takes in charge of low level planning. However, researchers (or engineers) need to define actions of the robot in addition to axioms among propositions for objects. Without the manual encodings, the hierarchical planner may need to play with the large number of propositions ($O(exp(DOF_{robot}))$=|*discretized CSpace*|) , when DOF_{robot} is the DOF of the robot. With such naive encoding, computational complexity of planning become ($O(exp(exp(DOFs))))$.

Moreover, naive hierarchical planners often have difficulty to find solutions for the following reason. Firstly, it requires interactions between subgoals. For example, the arm must go into the "Bean room" and turns the "light" on (subgoal) before it goes into the "small room" and turns the "shower" on (subgoal). This is essentially the *'Susman anomaly'* which means that the planner dose one thing (being in the Bean room) and then it has to retract it in order to achieve other goal (turning the shower on). Thus, it may require several backtrackings in planning. Secondly, there are two ways of (in principle) achieving "on(light)": (1) going through the small room; and (2) opening door to the Bean room from the Arm-base room. Unless manual encoding is given by an engineer, The latter way (going through the small room) is fine from the perspective of hierarchical planning. However, it will not work in practice because the arm is not long enough (kinematics). Formally, there is no *downward solution*.

Thus, this toy problem shows that (1) hierarchical planning does not work with a naive (simple) encoding, and (2) a complete encoding is too complex to encode manually. We are interested in general principles that underlie a solution to this problem.

In motion planning literature, hybrid planners are used to address these issues (Alami, Siméon, and Laumond 1989; Alami, Laumond, and Siméon 1997; Alami et al. 1998; Conner et al. 2007; Plaku, Kavraki, and Vardi 2008). How-

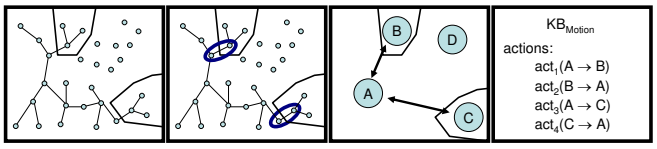

Figure 3: This figure illustrates a process to encode a motion plan into ATM (Action Theory with Motion). The process is follows: (1) a motion plan (a tree) is built by a motion planning algorithm; (2) actions which changes the states of objects are found; (3) propositions are generated (and grouped) based on the found actions; and (4) a ATM is created. Here, we assume that we have a function which provides discrete states of objects given the configuration of an object in finding actions (2). In this figure, the $door_1$ in figure 1 and 2 is closed in a set of states (A). The $door_1$ is moved little in B. However, the $door_1$ is not fully opened. Thus, configurations in the area D is not connected. The area C corresponds to the pushed light button on figure 1 and 2.

ever, these are either hard to build due to manual encodings, or infeasible to conduct complex tasks due to the curse of dimensionality of expanded CSpace. The size of CSpace of a hybrid planner exponentially increases with additional movable objects and given propositions. Thus, solving a complex problem may require extensive searches.

Here, we seamlessly combine the general purpose planning and the motion planning. Our planner finds all researchable locations and possible actions that change states of object, states of propositions, or the reachable set of objects.[2] Thus, high-level planner can start to plan based on actions extracted by a motion planner.[3]

However, the number of actions and states can be still intractable. To solve this problem, we partition the domain into the smaller groups of actions and states. For example, the domain can be partitioned as shown in figure 2. It is composed of three parts: (1) operating the shower switch; (2) operating the light switch; and (3) operating in between. The partition can be automatically done with approximate tight bound (Becker and Geiger 1996; Amir 2001).

A factored planner (Amir and Engelhardt 2003) efficiently finds a plan with the partitioned domain. The partitioned groups are connected as a tree shape. In each partitioned domain, our factored planner finds all the possible effects of the set of actions in each factored domain. Then, the planner passes the planned results into the parent of the partition in the tree. In the root node, all the valid actions and effects are gathered. The planner finds a plan for the task, if it exists.

Then, we use a local planner to find a concrete path in CSpace at the final step. However, there is no manual (explicit) encoding (eg. 'turning the *switch A*') between two layers, except logical constraints and mapping functions provided as input.

[2]Here, we assume that we know states of objects without uncertainty as in (Conner et al. 2007).

[3]Our planner may have more actions and states than the hand-encoded case.

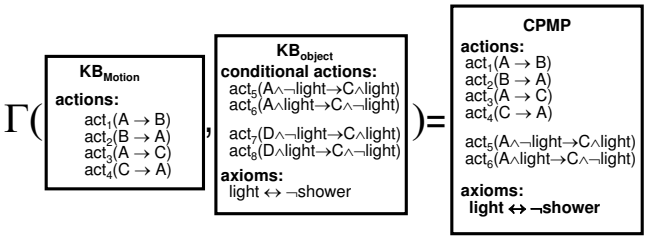

Figure 4: This shows an operation (or algorithm) to combine the extracted ATM with pre-existing BAT. BAT is independently given in a general form to a robot. Thus, ATM can be reusable for robots with different configurations space. Meanwhile, BAT is specific to a robot. Thus, some actions (e.g. act_7 and act_8) in BAT are invalidated, thus excluded in BAT.

Problem Formulation

Preliminaries

Situation Calculus The Situation Calculus is an action formalism which describes the precondition and the effect of each action with First-Order predicate logic formulae. We describe desirable constraints, if it is represented by First-Order formulae.

The Situation Calculus (McCarthy and Hayes 1987; Reiter 2001) is a sorted first order language for representing domains by means of actions, situations, and fluents. Actions and situations are first order terms, and situation-terms stand for history of actions, compound with a function symbol *do*: *do(a, s)* return the situation obtained by executing the action *a* in a situation *s*, which is a sequence of actions.

The dynamic domain is described by a Basic Action Theory BAT = (Σ, D_{S0} , D_{ss}, D_{una}, D_{ap}). Σ includes a set of foundational axioms for situations. D_{S0} is a set of first-order sentences that are valid in S_0. D_{ssa} is a set of successor state axioms for functional and relational fluents. D_{una} is the set of unique names axioms for actions. D_{ap} is a set of action precondition axioms. Please, refer the (Reiter 2001) for detail.

Configuration Space Given a robot and objects, a configuration describes the pose of the robot (r) and objects (O). Configuration space, $CSpace$ is the set of all possible configurations.

The set of configurations that has no collision with obstacles is called the free space C_{free}. The complement of C_{free} in C is called the obstacle region.

Here, we use a sampling-based motion planner (e.g. PRM (Kavraki et al. 1996) or RRT (Kuffner and LaValle 2000)) which extracts a connectivity graph among sampled configurations.

Combining Planning and Motion Planning

Inputs of problem is described as follows.

- $CSpace$: The configuration space of the robot and objects.
- BAT: The Basic Action Theory regarding to actions of the robot over objects.

- C_{init}: The initial configuration of the robot and objects.
- $Goal$: A first-order formula describing the goal condition.
- *Shared Fluents*: Predicates and functions shared by the Situation Calculus and the CSpace.

Here, if CSpace has n-dimensions, CSpace is $C_1 \times C_2 \times \ldots \times C_n$. We can represent $CSpace$ as $C_{Shared} \times C_{Motion}$. C_{Shared} is the cross product of dimensions which become inputs to BAT of Situation Calculus. C_{Motion} is the cross product of dimensions which are used only in motion planning. Thus, some dimensions become an input to BAT of the Situation Calculus. For theses dimensions, we write Fluents as follows.

$$P(c) \equiv P'(s)(c \in C_{Shared})$$
$$f(c) = f'(s)(c \in C_{Shared})$$

P and P' are predicates. f and f' are functions. P and f include dimensions of CSpace, although P' and f' replace the dimensions with a situation s. c is a configuration. Thus, such predicates and functions relate configurations in CSpace with situations in Situation Calculus.

Definition D_{map}: is defined with following axioms. With a configuration c and a situation s, we build a predicate as follows

$$P_{consistent}(c, s) \equiv \left(\bigwedge_i P_i(c) = P'_i(s) \right)$$

When i is the index for each predicate. In addition, we call the set of axioms with two Fluents as D_{map}.

Thus, our goal is to find a path which achieves the goal conditions while satisfying the action theory and avoiding collision in CSpace.

Combining Planning and Motion Planning (CPMP)

Action Theory with Motion (ATM)

Definition D_{motion}: We build a new theory based on the graph extracted from a motion planning algorithm. A resolution-complete motion planner builds a connectivity graph in CSpace. Based on the graph (V, E), we build two Predicates as follows.

$$P_{free}(c) \equiv \top \textit{ if } c \in V$$
$$P_{free}(c) \equiv \bot \textit{ otherwise}$$

$$P_{move}(c, c') \equiv \top \textit{ if } (c, c') \in E$$
$$P_{move}(c, c') \equiv \bot \textit{ otherwise}$$

$$P_{stable}(c, c') \leftrightarrow$$
$$(\forall s(P_{consistent}(c, s) \leftrightarrow P_{consistent}(c', s))$$
$$\wedge (P_{move}(c, c') \vee \exists c''(P_{stable}(c, c'') \wedge P_{stable}(c'', c'))))$$

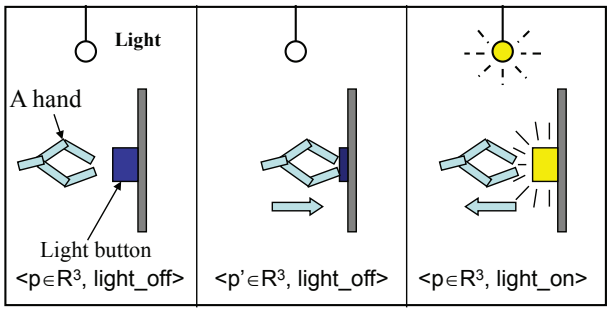

Figure 5: This example shows a situation in which one position in the workspace can correspond to two different states in the combined space (CPMP). Although the physical locations of the arm and button are same in the workspace, an internal state (eg. light is on) is different. The situation can be represented when CSpace and state space in KB are combined (CPMP), even though it is not possible to represent in the classical CSpace alone.

$$\forall c\,(P_{stable}(C_{init}, c) \rightarrow P_{realize}(c, S_0))$$

C_{init} is the initial configuration. S_0 is the initial situation in the BAT.

We call these sets of predicates as D_{motion}.

Definition D^*_{ap}: We change the set of precondition axioms in D_{ap}. Suppose that we have a following precondition axiom for an action act in a situation s.

$$Poss(act, s) \equiv \varphi_{poss}$$

We change it into following

$$Poss_{motion}(act, s) \equiv \varphi_{poss}$$
$$\wedge \exists c, c'(P_{realize}(c, s) \wedge P_{consistent}(c', do(act, s))$$
$$\wedge P_{move}(c, c') \wedge P_{free}(c) \wedge P_{free}(c'))$$

We call the set of modified precondition axioms as D^*_{ap}.

Definition $D^*_{\textit{eff}}$: We add the set of effect axioms in $D_{\textit{eff}}$. Suppose that we have a following effect axiom for an action act in a situation s.

$$Poss(act, s) \rightarrow \varphi_{\textit{eff}}$$

We change it into following axiom.

$$Poss_{motion}(act, s) \rightarrow \varphi_{\textit{eff}}$$
$$\wedge (\forall c, c', c''(P_{realize}(c, s) \wedge P_{consistent}(c', do(act, s)))$$
$$\rightarrow (P_{realize}(c', do(act, s))$$
$$\wedge (P_{stable}(c', c'') \rightarrow P_{realize}(c'', do(act, s)))))$$

We call the set of added effect axioms as $D^*_{\textit{eff}}$.

We define the unified actions theory, (Combining Planning and Motion Planning)

$$CPMP = (\Sigma, D_{S0}, D_{ss}, D_{una}, D_{motion}, D_{map}, D^*_{ap}, D^*_{\textit{eff}})$$

Lemma 1. *The complexity of planning problem in the $CPMP$ is as hard as P-SPACE.*

Proof. Any motion planning problem (P-SPACE hard) with movable objects can be reduced to a planning problem in $CPMP$. Suppose that $CPMP$ includes only external propositions which are extracted from the motion planning algorithm. □

Building Actions

We register an action (an edge between two points extracted from a motion planner) into CPMP in when two points have different states in CPMP with regard to mapping function as shown in figure 3. We validate abstract PDDL actions which are realized by the action. Thus, we build a hypergraph whose nodes are sets of modes (CSpace) which have the same state in terms of mapping functions and reachable objects. Our algorithm extensively searches actions with a *resolution complete* motion planner (i.e. PRM) until no new action is found in the hypergraph given a specified resolution.

Lemma 2. *The size of the discretized CSpace for a robot manipulating n objects with given propositions in CPMP is bounded by $O(exp(|objects| + n + p))$, when $|objects|$ is the number of objects, n is the DOF (Degree of Freedom) of the robot, and p is the number of propositions.*

Lemma 3. *The number of possible actions (edges) in the discretized CSpace for objects is only bounded by $O((|objects|) \cdot exp(|objects|))$, when the robot moves one object with an action.*

Proof. From a point in CSpace of object $O(exp(|objects|))$, we can choose an object $O(|objects|)$ to change states. □

Finding a Solution in $CPMP$

We provide a naive algorithm that solves a task in CPMP. Then, we provide two improvements: (1) that solves the problem in the (smaller) factored KBs; and (2) that reduces the number of propositions in $CPMP$ using workspace.

A Naive Solution

Given a task of CPMP, *NaiveSolution* finds a solution. It may use a general purpose planner ($GeneralPlanner$) to find an abstract solution. Then, ($LocalMotionPlan$) encodes a path in CSpace.

Algorithm: NaiveSolution
Input: r(a robot), BAT(Basic Action Theory), s_{start}(initial state), and s_{goal}(goal condition)
Output: $path_{concrete}$(Solution)
$ATM \leftarrow$ FindActionFromMP(r)
$CPMP = \Gamma(ATM, BAT)$
$path_{abstract} \leftarrow$ GeneralPlanner($CPMP$, s_{start}, s_{goal})
$path_{concrete} \leftarrow$ LocalMotionPlan($path_{abstract}$)

Algorithm 1: *NaiveSolution* provides a path for a robot. It uses a general planner ($GeneralPlanner$) to find an abstract solution. Then, it is encoded into the path in the CSpace by a motion plan ($LocalMotionPlan$).

Tree Decomposition of KB with Objects

Given a KB, finding a tree-decomposition of the minimum treewidth is a NP-hard problem. However, the complexity is only bounded by the treewidth of CPMP, if a tree-decomposition is found by an efficient heuristic (Becker and Geiger 1996; Amir 2001).

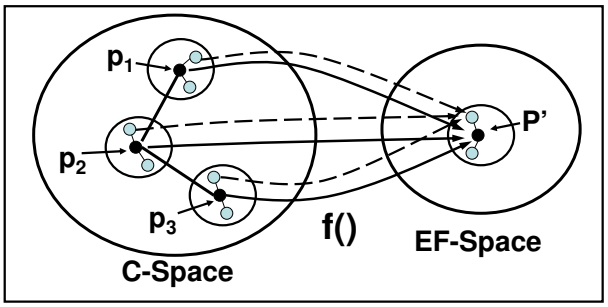

Figure 6: This figure shows a mapping function (f()) from a CSpace to an EF-Space. p_1, p_2, and p_3 in CSpace are mapped into p' in EF-Space. The connected lines ((p_1, p_2) and (p_2, p_3)) represent the first condition of Theorem 3. The circles represent the second condition.

Theorem 4. *The complexity of planning in CPMP is bounded by $O(exp(tw(CPMP)))$ if the tree-decomposition is given.*[4]

Proof. Proofs in (Brafman and Domshlak 2006; Amir 2001) can be easily modified to prove this theorem. □

From Exponential CSpace to Polynomial EF-Space

In this section, we provide a generalized method which project CSpace into much smaller workspace. It is an extension of our previous work (Choi and Amir 2007). it efficiently finds a solution when the projection method is applicable. Here, we want to transform CSpace into a smaller space, EF-Space, using a mapping function $f()$. The function ($f()$) maps each point (p) in CSpace into a point (p') in EF-Space with satisfying following conditions.

1. When P is a set of points whose image are p' in EF-Space ($f(p) = p'$), any pair of two elements (p_1, $p_2 \in P$) is connected each other in CSpace;
2. When two points (p and q) are mapped into two points (p' and q') in EF-Space. p and q are connected neighbor if and only if p' and q' are connected neighbor.

Two points are connected neighbor means when they are directly connected in the space.

Theorem 5. *The complexity of motion planning in EF-Space is bounded by following*

$$O(\textit{EF-Space}) \cdot O(max_{ep \in \textit{EF-Space}}(ball(P_{ep}))).$$

P_{ep} is a set of points whose image is ep. (That is, $P_{ep} = \{p | f(p) = ep\}$) The $ball(P)$ is volume of the ball which includes P.

Proof. Given a motion planning problem (an initial configuration and goal one), a path in EF-Space can be found in O(EF-Space) with a graph search algorithm. Given the path in EF-Space, one needs to search the whole ball in worst case. □

One simple example of EF-Space is the workspace of end-effector. Suppose that the points in CSpace are mapped into

[4] tw(KB) is the treewidth of KB.

the points of end-effector in workspace. One can build an algorithm that finds all the neighboring points from the innermost joint (or wheel) to the outermost joint with a dynamic programming. If points of the previous joint are connected to all neighboring points, the neighboring points of the current joint are found by a movement of current joint (current step) or a movement of any previous joint (previous steps). The found connected points in workspace satisfy the second conditions, if the first condition holds in the workspace.

In worst case, the first condition is hard to satisfy. In the environment, the mapping function (f) should be bijective. Thus, the EF-Space is nothing but the CSpace. However, the first condition holds in many applications where the distance between obstacles (or objects) and the robot is far enough. That is the theoretical reason why the planning problem in the sparse environment is easy even in CSpace.

Moreover, one can find another EF-Space considering topological shape of robot (Choi and Amir 2007). In the space, two points (p_1 and p_2) are mapped into the same point p'_1 if two configurations (p_1 and p_2) are homotopic, and they indicate the same end point. Otherwise, another point p'_2 is generated in the EF-Space. In 2D, an island obstacle divides configurations into two groups for each side (left or right). Thus, the EF-Space is exponentially proportional to the number of island obstacles. However, EF-Space itself is bounded by the workspace whose size is polynomial to the number of joints. Thus, it is much smaller than the CSpace and rather larger than the workspace.

A Unified Motion Plan

We present our algorithms in this section. The main algorithm , *UnifiedMotionPlanner* (Algorithm 2), is composed of three parts: *FindActionFromMP* (Algorithm 3); *FactoredPlan* (Algorithm 4); and *LocalPlanner*. The goal of *UnifiedMotionPlanner* is to find a solution to achieve a goal situation.

Algorithm: UnifiedMotionPlanner

Input: r(a robot), BAT(Basic Action Theory), s_{start}(initial state), s_{goal}(goal condition)
Output: $path_{concrete}$(Solution)
$ATM \leftarrow$ FindActionFromMP(r)
$CPMP$ = $\Gamma(ATM, BAT)$
$KB_{Tree} \leftarrow$ PartitionKBtoTree($CPMP$)
$path_{abstract} \leftarrow$ FactoredPlan(KB_{Tree}, s_{start}, s_{goal})
$path_{concrete} \leftarrow$ LocalPlan($path_{abstract}$)
return $path_{concrete}$

Algorithm 2: *UnifiedMotionPlaner* finds all the reachable locations and actions in each location with FindActionFromMP. A motion planner is embedded in FindActionFromMP to extract abstracted actions in CSpace. Then, PartitionKBtoTree partitions the $CPMP$ into a tree. FactoredPlan finds a solution given the pair of initial and goal condition in the partitioned tree domain. The LocalPlan finds a concrete path for the robot.

FindActoinFromMP

FindActionFromMP searches all the reachable locations and actions in CSpace or EF-Space. In both cases, it has a dramatically reduced space.

Algorithm: FindActionFromMP

Input: r(a robot)
Output: ATM(extracted actions)
$MP_{Tree} \leftarrow$ a random tree in CSpace built by a motion planner (e.g. Probabilistic Roadmap, Factored-Guided Motion Planning)
for *each edge* $(e_{ij}) \in MP_{Tree}$ **do**
  if $state(p_i) \neq state(p_j)$ **then**
    $KB_M \leftarrow KB_M \bigcup D^*_{ap}$ (as in section
    $KB_M \leftarrow KB_M \bigcup D^*_{eff}$ (as in section
return KB_M

Algorithm 3: . *FindActionFromMP* finds all abstract actions for a robot. A motion planner (eg. *FactorGuidedPlan* or *RoadmapMethod*) recursively finds all the reachable locations and actions. Then, the algorithm insert actions of each configuration (c_{ij}) of objects in the workspace. It assume that the object is in the configuration (c_{ij}). Thus, the condition (configuration of objects) is combined into the actions (act_{ij}). The union of all actions becomes the KB_M.

FactoredPlan

FactoredPlan finds a solution after factoring the domain (the space of end-effector in workspace) into small domains. It decomposes the domain into a tree in which each partitioned group becomes nodes, and shared axioms appear on a link between nodes. Then, it finds partial plans for a node and its children nodes with assuming that the parents nodes may change any shared states in between. After all, it finds a global solution in the root node.

Algorithm: FactoredPlan

Input: KB_{Tree} (partitioned KB as a tree), s_{start} (initial states), s_{goal} (goal condition)
Output: $path_{abstract}$ (An abstract plan)
depth $\leftarrow$ (predefined) number of interaction between domains.
for *each node*(KB_{part}) *in* KB_{Tree} *from leaves to a root* **do**
  $Act_{ab} \leftarrow$ PartPlan(KB_{part}, depth) .
  SendMessage(Act_{ab}, the parent node of KB_{part})
$path_{ab} \leftarrow$ a solution from s_{init} to s_{goal} in the root node of KB_{tree}
return $path_{ab}$

Algorithm 4: *FactoredPlanning* algorithm automatically partitions the domain to solve the planning problem (from s_{init} to s_{goal}). It iterates domains from leaves to the root node without backtracks. In each node, *PartPlan* finds all possible actions that change shared states in the parents node. *PartPlan* assumes that the parent node may change any states in the shared states in between. The planned actions in the subdomain become an abstract action in the parent node. They are sent by *SendMessage*.

Related Works

Here, we review the related works in two aspects: (1) using logical representation in robot planning; and (2) modifying the motion planning algorithm to achieve complex task (eg.

manipulating objects). One may see the former way as top-down and the latter way as bottom-up.

(Alami et al. 1998) presents a well-integrated robot architecture which controls multiple robots. It uses logical representations in higher level planners and CSpace based motion planners in lower-level planning. However, the combination of two planners is rather naive (manual).

Recently, (Conner et al. 2007) provides an improved way to combine the *Linear Temporal Logic (LTL)* to control continuously moving cars in the simulated environment.[5] However, their model is a nondeterministic automata, while our model is deterministic. Due to the intractability of nondeterministic model, their representation is restricted to a subset of LTL to achieve a tractable (polynomial time) algorithm. Experiments are focused on controlling cars instead of manipulating objects.

Motion planning research has a long-term goal of building a motion planning algorithm that finds plans for complex tasks (eg. manipulating objects). (Stilman and Kuffner 2005) suggests such a planning algorithm based on a heuristic planner (Chen and Hwang 1991) which efficiently relocates obstacles to reach a goal location. Recently, it was extended to embed constraints over objects into the *CSpace* (Stilman 2007). In fact, the probabilistic roadmap method (Kavraki et al. 1996) of the algorithm is highly effective in manipulating objects. However, we argue that our algorithm (factored planning) is more appropriate in terms of generality and efficiency than a search-based (with backtracks) heuristic planner.

Other works also make efforts in this direction to build a motion planning algorithm for complex tasks. (Plaku, Kavraki, and Vardi 2008) solves a motion planning problem focused on safety with logical constraints represented with LTL . (M. Pardowitz 2007) focuses on learning actions for manipulating objects based on the explanation based learning (Dejong and Mooney 1986). They use a classical hierarchical planner in planning. (J. Van den Berg 2007) provides an idea that extracts the propositional symbols from a motion planner. The symbols are used to check the satisfiability of the planning problems. (S. Hart 2007) uses a potential field method to achieve complex tasks with two arms. However, the main interests of these works are not planning algorithm, or are limited to the rather simpler tasks.

An Experiment in Simulation

We build our algorithm for a task that pushes buttons to call numbers. There are 8 buttons in total. 4 buttons ($key1(P1)$, $key2(P2)$, $unlock(P3)$, and $lock(P4)$) are used to lock (and unlock) the buttons. Other 4 buttons ($\#A(P5)$, $\#B(P6)$, $\#C(P7)$ and $Call(P8)$) are used to make phone calls. Initially, the button is locked, the robot needs to push unlock buttons after pushing both key buttons ($P1$ and $P2$). Then, the robot can make a phone call with pushing the $Call$ button ($P8$) after selecting an appropriate number among $\#A(P5)$, $\#B(P6)$, and $\#C(P7)$. After a call, the buttons are automatically unlocked. We encode such constraints and action in a PDDL.[6]

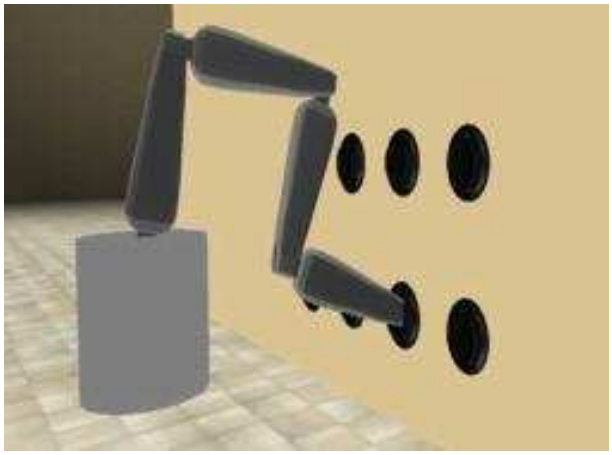

Figure 7: This is a capture of the motion of push button in the wall in experiments. The robot has 5 DOFs (rotational joints on the base and 4 revolute joints on the arm). We do experiment with increasing the number of joints from 2 to 9.

We build a tree from a randomized algorithm with 80000 points in CSpace. With a labeling function that returned the states of buttons, we found 33 edges in the tree[7]. They are encoded into 8 actions for 8 buttons. Then, the combined KB ($CPMP$) is used to find a goal (calling all numbers ($\#A$, $\#B$, and $\#C$). The returned abstract actions are decoded into a path on the tree of motion plan. Figure 7 is a snapshot of the simulation.[8]

In this experiment, we focus on extracting actions from a motion planning algorithm, because the factored planer itself is not a contribution of this paper. Theoretical and experimental benefits of *FactoredPlan* is shown in the previous papers (Amir and Engelhardt 2003; Brafman and Domshlak 2006). We run our simulation on a general purposed planner (Fourman 2007). Thus, the *NaiveSolution* algorithm is used in this simulation.

Conclusions and Future Research

We present an algorithm that combines the general purpose (logical) planner and a motion planner. Our planner is designed to manipulate objects with robot. To solve the problem, previous works used a hierarchical planner (high-level) and a motion planner (low-level). Most of them used manual encodings between two layers. That was one of technical hardness of this problem.

Theoretically, combining such planners is hard for the following reasons: (1) hierarchical planner is hard and not feasible sometime; and (2) direct combination of CSpace and state space gives an doubly exponential search problem.

[5] Any *First Order Logic (FOL)* sentences can be reduced to *Linear Temporal Logic (LTL)*. Thus, LPL is a superset of FOL.

[6] Situation Calculus encoding is not impleted yet

[7] We simplify the manipulations for attaching and detaching buttons

[8] The details of encoded actions and movies are available at http://reason.cs.uiuc.edu/jaesik/cpmp/supplementary/.

Moreover, we can loss the geometric motion planning information, if we translate everything to PDDL (McDermott 1998) without a motion planner.

We combine the CSpace and state space in a KB, CPMP (Combining Planning and Motion Planning). Moreover, we provide the computational complexity of the problem. We also argue that the treewidth of CPMP determines the hardness of a manipulation task.

However, the suggested algorithm still has some limitations that need to be improved in future research. The exploration steps in *FindActionFromMP* may take long time due to the large cardinality of state space ($O(n + |objects| + p)$ as in lemma 2. Assumptions of EF-space would inappropriate for cluttered environments where $O(max_{ep \in \text{EF-Space}}(ball(P_{ep}))$ of theorem 5 are intractable.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 05-46663. We also thank UIUC/NCSA Adaptive Environmental Sensing and Information Systems (AESIS) initiative for funding part of the work.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research* 17(4):315–337.

Alami, R.; Laumond, J.-P.; and Siméon, T. 1997. Two manipulation planning algorithms. In Laumond, J.-P., and Overmars, M., eds., *Algorithms for Robotic Motion and Manipulation*. Wellesley, MA: A.K. Peters.

Alami, R.; Siméon, T.; and Laumond, J.-P. 1989. A geometrical approach to planning manipulation tasks. In *Proceedings International Symposium on Robotics Research*, 113–119.

Amir, E., and Engelhardt, B. 2003. Factored planning. In *IJCAI*, 929–935.

Amir, E. 2001. Efficient approximation for triangulation of minimum treewidth. In *UAI*, 7–15.

Becker, A., and Geiger, D. 1996. A sufficiently fast algorithm for finding close to optimal junction trees. In *UAI*, 81–89.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *AAAI*.

Brock, O., and Khatib, O. 2000. Real-time replanning in high-dimensional configuration spaces using sets of homotopic paths. In *ICRA'00*, 550–555.

Chen, P., and Hwang, Y. 1991. Motion planning for a robot and a movable object amidst polygonal obstacles. In *ICRA'91*, 444–449.

Choi, J., and Amir, E. 2007. Factor-guided motion planning for a robot arm. In *IROS'07*, 27–32.

Choi, J., and Amir, E. 2009. Combining planning and motion planning. In *ICRA'09*, 238–244.

Conner, D. C.; Kress-Gazit, H.; Choset, H.; Rizzi, A.; and Pappas, G. J. 2007. Valet parking without a valet. In *IROS'07*.

Cortés, J. 2003. *Motion Planning Algorithms for General Closed-Chain Mechanisms*. Ph.D. Dissertation, Institut National Polytechnique de Toulouse, Toulouse, France.

Dacre-Wright, B.; Laumond, J.-P.; and Alami, R. 1992. Motion planning for a robot and a movable object amidst polygonal obstacles. In *ICRA'92*, volume 3, 2474–2480.

Dejong, G., and Mooney, R. 1986. Explanation-based learning: An alternative view. *Mach. Learn.* 1(2):145–176.

Fourman, M. 2007. Propplan. Software.

J. Van den Berg, M. O. 2007. Kinodynamic motion planning on roadmaps in dynamic environments. In *IROS'07*.

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

Kuffner, J. J., and LaValle, S. M. 2000. RRT-connect: An efficient approach to single-query path planning. In *ICRA'00*.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* search with provable bounds on sub-optimality. In *NIPS'03*.

M. Pardowitz, R. Zollner, R. D. 2007. Incremental acquisition of task knowledge applying heuristic relevance estimation. In *IROS'07*.

McCarthy, J., and Hayes, P. J. 1987. *Some philosophical problems from the standpoint of artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

McDermott, D. 1998. The planning domain definition language manual.

Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2008. Hybrid systems: From verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design*.

Reiter, R. 2001. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. Cambridge, Mass.: MIT Press. The frame problem and the situation calculus.

S. Hart, R. G. 2007. Natural task decomposition with intrinsic potential fields. In *IROS'07*.

Stilman, M., and Kuffner, J. 2005. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics* 2(4):479–504.

Stilman, M. 2007. Task constrained motion planning in robot joint space. In *IROS'07*.