Combining Planning and Motion Planning

Jaesik Choi and Eyal Amir

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 planning and motion planning. Our approach integrates motions of a robot with other (non-physical or external-to-robot) actions to achieve a goal while manipulating objects. It differs from previous, hierarchical approaches in that (a) it considers kinematic constraints in configuration space (C-space) together with constraints over object manipulations; (b) it automatically generates high-level (logical) actions from a C-space based motion planning algorithm; and (c) it decomposes a planning problem into small segments, thus reducing the complexity of planning.**

I. 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). However, the complexity of such planning algorithm is exponentially proportional to the dimension of the space (the degree-of-freedom (DOF) of the robot and the number of objects) [1]. It was shown that planning with movable objects is P-SPACE hard [2], [3], [4]. Nonetheless, previous works examined such planning in depth [5], [6], [7], [8], [9], [4] because of the importance of manipulating objects. The theoretical analysis gave rise to some practical applications [9], [10], [4], [11], but general purpose manipulation remains out of reach for real-world-scale applications.

Most of current algorithms (or architectures) for manipulating objects [12], [9], [4] divide the planning problem into high-level reasoning and low-level motion planning. Given propositions and abstract actions, a high-level planner finds an abstract plan that leads to a goal. Then, the low-level motion planner finds plans for each set of actions in the abstract plan. The low-level motion planner considers obstacles and the kinematic constraints in C-space.

Unfortunately, previous algorithms [12], [9] use manually encoded interfaces between the high-level layer and the low-level layer. That is, all possible actions (eg. push the button and insert a key) in the high-level reasoning system should be manually defined in the low-level motion planner. This manual encoding is difficult and imprecise many times, leading to costly and error-prone products, especially in complex domains and tasks.

J. Choi and E. Amir are with the Department of Computer Science, University of Illinois at Urbana Champaign, Urbana, IL, 61874 USA (e-mail: {jaesik,eyal}@cs.uiuc.edu)

We minimize manual encodings using the reachability of objects. That is, logical actions are extracted from a tree (planned by a motion planning algorithm), if the actions change the reachability of objects (i.e. a switch can be reachable by opening a door).

Our algorithm provides a path of a robot given following inputs: configurations of a robot and objects; constraints between objects; an initial state; and a goal condition.[1] We use logical expressions to represent both spatial constraints in C-space (e.g. collision) and constraints in state space (we define them formally in section IV). We automatically build a set of actions from a motion planner, while it was done by hands in previous works.

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 abstract plan from the logical domain, and (3) decoding it into C-space. It extracts PDDL actions [13] from a tree constructed by a motion planner in C-space. Then, it combines extracted actions with a given KB_{object} (Knowledge Base) that has propositions, axioms (propositional formulae) and abstract PDDL 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 [14], [15] which are designed for the decomposed domain. In decoding, a motion plan is found from the abstract plan.

We argue that the complexity of a planning problem is bounded by the treewidth of the encoded KB. One may think some analogy between the treewidth of KB in this paper and the number of mutually-interfering objects in the motion planning literature. However, the treewidth is more general expression because KB has more expressive power than the conventional C-space. In addition, this work proposes two improvements in terms of efficiency. One improvement is to use a factored planning algorithm for the decomposed domain. The other is to encode actions on behalf of workspace which is much smaller than C-space.

This approach is a unique decomposition-based path planning algorithm. We minimize manual encodings which are required to manipulate objects. Both (kinematic) constraints of the robot, and constraints of manipulating object are considered in our planning. It is efficient because its efficiency depends only on the workspace (2D or 3D), when appropriate

[1] For each object, we provide a function which maps from a configuration to discrete states (labels) of objects, if discrete states are required for the provided constraints of objects (KB_{object}).

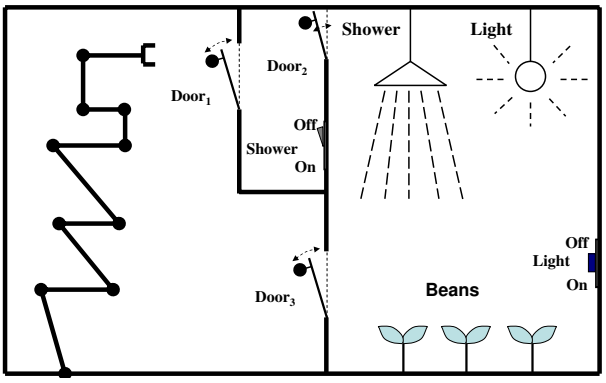

Fig. 1: This figure shows an example of manipulating objects with a robotic arm. The goal is to take care of beans in a glasshouse. Beans require water and light everyday. The robot will provide water and light for beans. To accomplish this goal, the arm needs to manipulate objects such as doors and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

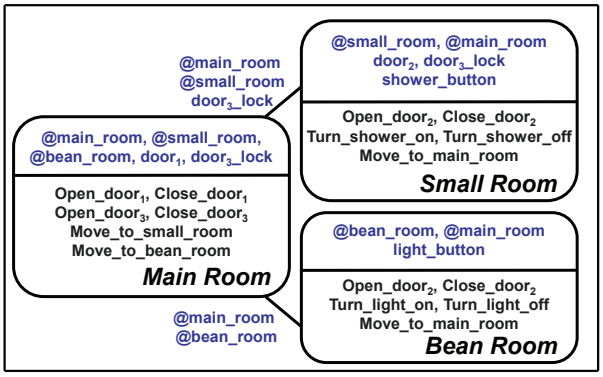

Fig. 2: This is a possible tree decomposition for the toy problem of figure 1. The shared propositions appear on edges between subgroups. For example, a proposition ('@$door_3\_lock$') is shared by two subgroups ('Main Room' and 'Small Room') because the proposition is used by actions of two subgroups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

conditions are met. Moreover, our method calculates actions of a robot once and can reuse them for other tasks.

Section II presents related works. Section III provides a motivational example. Section IV explains our encoding to build a KB. Sections V and VI show our algorithm. Finally, section VII provides experimental results followed by the conclusion in section VIII.

II. 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.

[9] presents a well-integrated robot architecture which controls multiple robots. It uses logical representations in higher level planners and C-space based motion planners in lower-level planning. However, the combination of two planners is rather naive (manual).

Recently, [11] provides an improved way to combine the *Linear Temporal Logic (LTL)* to control continuously moving cars in the simulated environment.[2] 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). [4] suggests such a planning algorithm based on a heuristic planner [2] which efficiently relocates obstacles to reach a goal location. Recently, it was extended to embed constraints over objects into the *C-space* [16]. In fact, the probabilistic roadmap method [7] 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. [17] solves a motion planning problem focused on safety with logical constraints represented with LTL . [18] focuses on learning actions for manipulating objects based on the explanation based learning [19]. They use a classical hierarchical planner in planning. [20] provides an idea that extracts the propositional symbols from a motion planner. The symbols are used to check the satisfiability of the planning problems. [21] 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.

III. 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 C-space 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 C-space. 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

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

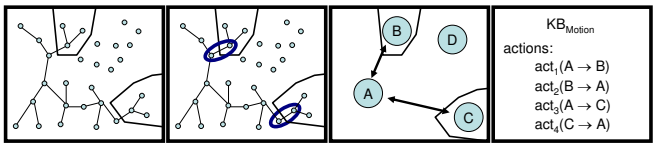

Fig. 3: This figure illustrates a process to encode a motion plan into KB_M. The process is follows: (1) a motion plan (a tree) is built by a motion planning algorithm; (2) actions which changes the states of objects are found; (3) propositions are generated (and grouped) based on the found actions; and (4) a KB_M 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.

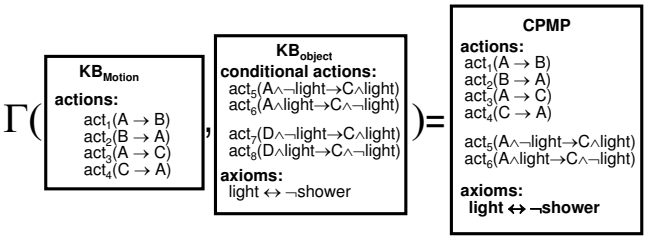

Fig. 4: This shows an operation (or algorithm) to combine the extracted KB_M with pre-existing KB_O. KB_O is independently given in a general form to a robot. Thus, KB_O can be reusable for robots with different configurations space. Meanwhile, KB_{MP} is specific to a robot. Thus, some actions (e.g. act_7 and act_8) in KB_O are invalidated by the KB_M.

hierarchical planner may need to play with the large number of propositions ($O(exp(DOF_{robot}))$=|*discretized C-Space*|) , 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 [22], [23], [9], [11], [17]. However, these are either hard to build due to manual encodings, or infeasible to conduct complex tasks due to the curse of dimensionality of expanded C-space. The size of C-space of a hybrid planner exponentially increases with additional movable objects and given propositions. Thus, solving a complex problem may require extensive search.

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.[3] Thus, high-level planner can start to plan based on the actions extracted by a motion planner.[4] The number of actions and states can be different according to constraints of the robot.

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 [24], [25].

A factored planner [14] 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 C-space 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.

IV. Problem Formulation

A. Combining C-space and State Space

Here we suggest new problem formulation to combine C-space of an object-manipulating robot and KB (defined in the next paragraph) of objects and propositions. An object, located in a specific workspace, generates propositions into KB. Other axioms (propositional formulae) and actions (PDDL[13]) are given for the propositions. We will call this KB as **CPMP** (Combining Planning and Motion Planning).

Definition CPMP (Combining Planning and Motion Planning) is a Knowledge Base which is composed of propositions for states of a robot and objects, axioms of a robot and objects, and actions of a robot. It encodes a set of points in C-space into a proposition (p_c) in the *CPMP*. Actions of a robot are encoded into actions of the *CPMP*. A set of propositions and actions can be constrained each other by an axiom (a propositional formula).

[3]Here, we assume that we know states of objects without uncertainty as in [11].

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

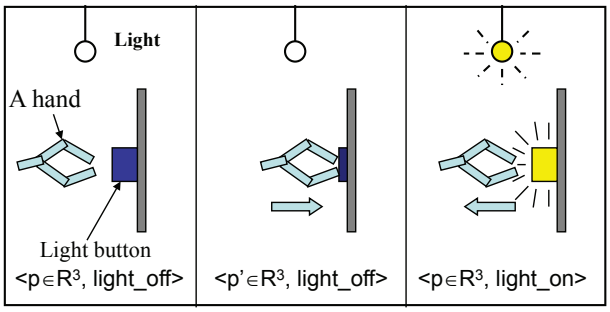

Fig. 5: This example shows a situation in which one position in the workspace can correspond to two different states in the combined space (CPMP). Although the physical locations of the arm and button are same in the workspace, an internal state (eg. light is on) is different. The situation can be represented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

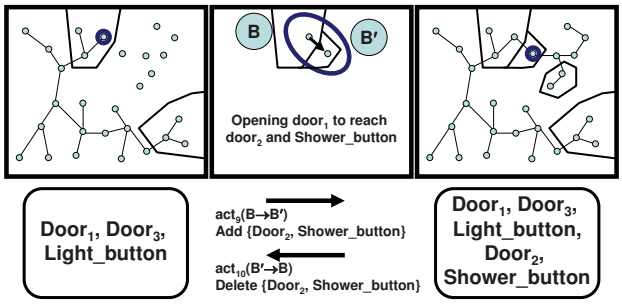

Fig. 6: This figure represents an action which changes states of the object ($Door_1$) to change the reachable set of objects. Before doing the action ($act_9 B \rightarrow B'$), the set of reachable objects are { $Door_1$, $Door_3$, and $Light\_button$ }. After the action, { $Door_2$, $Shower\_button$ } are also included in the reachable set.

A CPMP is composed of propositional symbols, axioms, and actions. The propositional symbols (P) represent states in binary values. The axioms ($Axiom$) are propositional formulae. The actions ($Action$) represent the pair of preconditions and effects of a robot motion. It has a set of propositions that represents states of a robot, objects and internal states. *External states* are propositions in KB_M extracted from C-space. *Internal states* are propositions explicitly given in KB_O.

It also include a set of axioms. The axioms (logical formula) represent relations among states of objects. When a state of an object (o^i) is o_1^i (e.g. light), the state of another object (o^j) is constraints o_1^j (e.g. $\neg$shower).[5] It is represented as follows.

$$o_1^i \leftrightarrow o_1^j$$

In CPMP, a set of actions, KB_M, is generated from a tree (or graph) in C-space built by a motion planning algorithm as shown in figure 3. In detail, two points (p_1 and p_2) in the network are connected by a edge (an action of the robot). This can be simply encoded as follows.

$$\begin{aligned} Action: \quad & Move(p_1, p_2) \\ & \text{Precond} : p_1 \\ & \text{Effect} : p_2 \wedge \neg p_1 \end{aligned}$$

When the action changes the internal state of an object (o) from o_1 to o_2, the action can be encoded as follows.

$$\begin{aligned} Action: \quad & MoveObject(p_1, p_2, o_1, o_2) \\ & \text{Precond} : p_1 \wedge o_1 \\ & \text{Effect} : p_2 \wedge o_2 \wedge \neg p_1 \wedge \neg o_1 \end{aligned}$$

Figure 5 represents the expressive power of $CPMP$. It represents a situation which can be described in $CPMP$ but C-space. The same physical locations are different states in $CPMP$ because the state of the light is changed.

[5]Such axioms are manually encoded. However, the encodings are independent of a specific robot. Thus, the encodings can be reusable to other types of robot. Moreover, there are algorithms [26], [27] which can generate such axioms with a sensor-mounted robot.

A $CPMP$ has following properties.

- A $CPMP$ has more expressive power than a C-space, if no two configurations in C-space can distinguish the two internal states.[6]
- It may reduce the number of propositions in $CPMP$, if spatial locations of end-effector are well-defined into disjoint sets. In each disjoint set, all spatial locations of end-effector have an identical internal state. Thus, any edge between the two disjoint sets changes some of the internal state.

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. ■

B. Encoding with Mapping Functions and Reachability

Here, we suggest an automatic encoding for moving objects with maintaining consistent states given mapping functions[7] and reachability of objects. When a robot manipulates movable objects, it changes C-space of the robot. Hybrid systems[22], [23], [9] consider each C-space as a mode. Then, each manipulation connects two distinct modes. However, the size of the space is exponentially proportional to the number of objects and the number of joints. To address this issue, we group a set of modes based on the states of propositions and reachability of objects as shown in figure 3 and 6.

We register an action (an edge between two points extracted from a motion planner) into CPMP in following two cases. The first case is when two points have different states in CPMP with a mapping function as shown in figure 3. We validate abstract PDDL actions which are realized by the action. The second case is when an edge changes the set of reachable objects or one of mapping functions as

[6]C-space normally takes into account configurations which only consider spartial locations of a robot or objects.

[7]A mapping function provides a state of a proposition (eg. object) given a configuration of objects and a robot.

shown in figure 6.[8] Thus, we build a hypergraph whose nodes are sets of modes (C-space) which have the same state in terms of mapping functions and reachable objects. Our algorithm extensively searches actions with a *resolution complete* motion planner until no new action is found in the hypergraph given a specific resolution.

Lemma 2: The size of the discretized C-space 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 C-space 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 C-space of object $O(exp(|objects|))$, we can choose an object $O(|objects|)$ to change states. ■

V. 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. 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 C-space.

Algorithm: NaiveSolution
Input: r(a robot), KB_O(KB of objects), s_{start}(initial state), and s_{goal}(goal condition)
Output: $path_{concrete}$(Solution)
$KB_M \leftarrow$ FindActionFromMP(r)
$CPMP$ = $\Gamma(KB_M, KB_O)$
$path_{abstract} \leftarrow$ 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 C-space by a motion plan ($LocalMotionPlan$).

B. 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 [24], [25].

Theorem 4: The complexity of planning in CPMP is bounded by $O(exp(tw(CPMP)))$ if the tree-decomposition is given.[9]

Proof: Proofs in [15], [25] can be easily modified to prove this theorem. ■

[8]The reachable objects are added to preconditions and effects respectively.

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

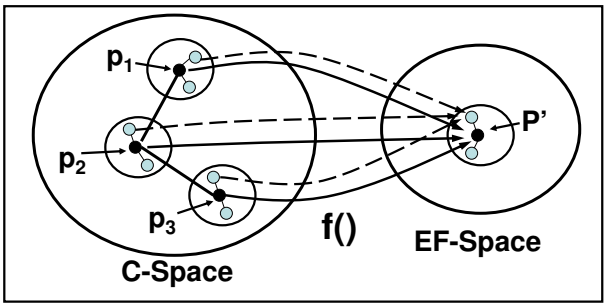

Fig. 7: This figure shows a mapping function (f()) from a C-space to an EF-Space. p_1, p_2, and p_3 in C-space are mapped into p' in EF-Space. The connected lines ((p_1, p_2) and (p_2, p_3)) represent the first condition of Theorem 3. The circles represent the second condition.

C. From Exponential C-space to Polynomial EF-Space

In this section, we provide a generalized method which project C-space into much smaller workspace. It is an extension of our previous work [28]. it efficiently finds a solution when the projection method is applicable. Here, we want to transform C-space into a smaller space, EF-Space, using a mapping function $f()$. The function ($f()$) maps each point (p) in C-space 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 C-space;
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

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

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 C-space are mapped into 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 C-space. 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 C-space.

Moreover, one can find another EF-Space considering topological shape of robot [28]. 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, two groups of configurations are divided by an island in right and left sides. 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 C-space and rather larger than the workspace.

VI. 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), KB_O(KB of objects), s_{start}(initial state), s_{goal}(goal condition)
Output: $path_{concrete}$(Solution)
$KB_M \leftarrow$ FindActionFromMP(r)
$CPMP$ = $\Gamma(KB_M, KB_O)$
$KB_{Tree} \leftarrow$ 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 C-space. 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.

A. FindActoinFromMP

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

B. 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: FindActionFromMP

Input: r(a robot)
Output: KB_M(extracted actions)
$MP_{Tree} \leftarrow$ a random tree in C-space 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**
    $KB_M \leftarrow KB_M \bigcup \{$
    $act_{ij}(state(p_i) \wedge p_i \rightarrow state(p_j) \wedge p_j \wedge \neg p_i)$ $\}$
    $KB_M \leftarrow KB_M \bigcup \{$
    $act_{ji}(state(p_j) \wedge p_j \rightarrow state(p_i) \wedge p_i \wedge \neg p_j)$ $\}$
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.

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*.

VII. An Experiment in Simulation

In this preliminary 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 KB_O.

To build KB_M, we build a tree from a randomized algorithm with 80000 points in C-space. With a labeling function that returned the states of buttons, we found 33 edges in the tree[10]. They are encoded into 8 actions in KB_M 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

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

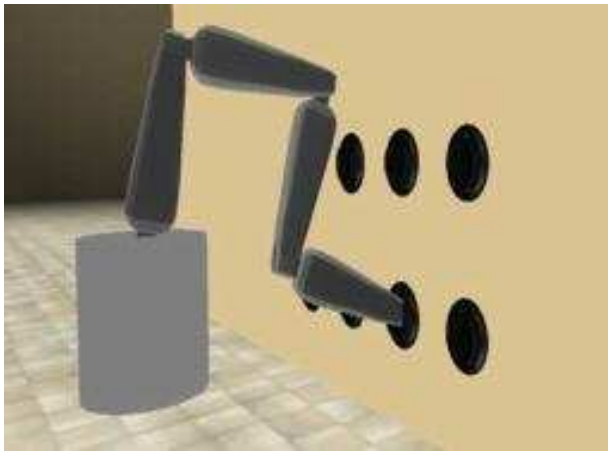

Fig. 8: This is a capture of the motion of push button in the wall in experiments. The robot has 5 DOFs (rotational joints on the base and 4 revolute joints on the arm). We do experiment with increasing the number of joints from 2 to 9.

of motion plan. Figure 8 is a snapshot of the simulation.[11]

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 [14], [15]. We run our simulation on a general purposed planner [29]. Thus, the *NaiveSolution* algorithm is used in this simulation.

VIII. 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 C-space and state space gives an doubly exponential search problem. Moreover, we can loss the geometric motion planning information, if we translate everything to PDDL [13] without a motion planner.

We combine the C-space 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.

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

IX. 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

[1] J. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1987.
[2] P. Chen and Y. Hwang, "Motion planning for a robot and a movable object amidst polygonal obstacles," in *ICRA'91*, 1991, pp. 444–449.
[3] B. Dacre-Wright, J.-P. Laumond, and R. Alami, "Motion planning for a robot and a movable object amidst polygonal obstacles," in *ICRA'92*, vol. 3, 12–14 May 1992, pp. 2474–2480.
[4] M. Stilman and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *International Journal of Humanoid Robotics*, vol. 2, no. 4, pp. 479–504, December 2005.
[5] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* search with provable bounds on sub-optimality," in *NIPS'03*, 2003.
[6] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning." in *ICRA'00*, 2000.
[7] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Trans. on Rob. and Auto.*, vol. 12, no. 4, pp. 566–580, 1996.
[8] O. Brock and O. Khatib, "Real-time replanning in high-dimensional configuration spaces using sets of homotopic paths." in *ICRA'00*, 2000, pp. 550–555.
[9] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *International Journal of Robotics Research*, vol. 17, no. 4, pp. 315–337, 1998.
[10] J. Cortés, "Motion planning algorithms for general closed-chain mechanisms," Ph.D. dissertation, Institut National Polytechnique de Toulouse, Toulouse, France, 2003.
[11] H. C. A. R. David C. Conner, Hadas Kress-Gazit and G. J. Pappas, "Valet parking without a valet," in *IROS'07*, 2007.
[12] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Trans. on Rob. and Auto.*, vol. 2, no. 1, pp. 14–23, March 1986.
[13] D. McDermott, "The planning domain definition language manual." 1998.
[14] E. Amir and B. Engelhardt, "Factored planning." in *IJCAI*, 2003, pp. 929–935.
[15] R. I. Brafman and C. Domshlak, "Factored planning: How, when, and when not." in *AAAI*, 2006.
[16] M. Stilman, "Task constrained motion planning in robot joint space," in *IROS'07*, 2007.
[17] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: From verification to falsification by combining motion planning and discrete search," *Formal Methods in System Design*, 2008.
[18] R. D. M. Pardowitz, R. Zollner, "Incremental acquisition of task knowledge applying heuristic relevance estimation," in *IROS'07*, 2007.
[19] G. Dejong and R. Mooney, "Explanation-based learning: An alternative view," *Mach. Learn.*, vol. 1, no. 2, pp. 145–176, 1986.
[20] M. O. J. Van den Berg, "Kinodynamic motion planning on roadmaps in dynamic environments," in *IROS'07*, 2007.
[21] R. G. S. Hart, "Natural task decomposition with intrinsic potential fields," in *IROS'07*, 2007.
[22] R. Alami, T. Siméon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks," in *Proceedings International Symposium on Robotics Research*, 1989, pp. 113–119.
[23] R. Alami, J.-P. Laumond, and T. Siméon, "Two manipulation planning algorithms," in *Algorithms for Robotic Motion and Manipulation*, J.-P. Laumond and M. Overmars, Eds. Wellesley, MA: A.K. Peters, 1997.
[24] A. Becker and D. Geiger, "A sufficiently fast algorithm for finding close to optimal junction trees," in *UAI*, 1996, pp. 81–89.
[25] E. Amir, "Efficient approximation for triangulation of minimum treewidth," in *UAI*, 2001, pp. 7–15.
[26] E. Amir and S. J. Russell, "Logical filtering," in *IJCAI*, 2003, pp. 75–82.
[27] D. Shahaf and E. Amir, "Logical circuit filtering," in *IJCAI*, 2007, pp. 2611–2618.
[28] J. Choi and E. Amir, "Factor-guided motion planning for a robot arm," in *IROS'07*, 2007.
[29] M. Fourman, "Propplan," Software, Dec 2007. [Online]. Available: http://homepages.inf.ed.ac.uk/mfourman/tools/propplan