Mobile Manipulation: Encoding Motion Planning Options Using Task Motion Multigraphs

Ioan A. Şucan, Lydia E. Kavraki

Abstract— **This paper introduces the concept of a task motion multigraph, a data structure that can be used to reveal a difficulty specific to mobile manipulation: the possibility of planning in different state spaces in order to achieve the same goal. The different options reflect the mobile manipulator's ability to use different hardware components to perform a required task. For instance, a humanoid robot can open a door with its left arm or with its right arm. Thus, motion planning can be performed in the left arm's state space or in the right arm's state space. Given the specification of a task, it is shown how to encode the available motion planning options in a task motion multigraph. An algorithm that computes sequences of motion plans for mobile manipulators using the newly introduced notion is presented and evaluated. The algorithm makes use of information from the task motion multigraph to prioritize the spaces for which motion plans are computed. Experimental results show that reduced planning times can be obtained when considering the available planning options.**

I. Introduction

Mobile manipulation is a problem that has benefited from increasing interest over the last decade [1]–[7]. For the purposes of this paper we consider mobile manipulators to be robotic devices that have one or more arms and a means to move their base within the environment [8]–[10]. Such robotic systems can be used as service robots, assistive robots, search and rescue robots, etc.

A simple typical task for such a robotic system is fetching a cup of coffee or placing a book on a shelf. These example tasks, even though very simple, require the robot to reason about the sequence of actions it needs to perform in order to achieve its goal. For instance, when fetching a cup of coffee, the robot knows the map, its initial position, the location of the coffee cup and the desired destination for the coffee cup. The task can thus be decomposed into reaching, grasping and delivering the cup. The reaching and delivering actions may require moving the base of the robot as well, if, for instance, the arm is not sufficiently long to reach the cup. The reaching sub-task is further complicated by the fact that the cup can be grasped in multiple ways. This leads to questions such as whether there is a grasping pose that is better than others, or which grasping pose the robot should attempt to use. If we now consider the problem of placing a book on a shelf, there are again a number of possible ways to grasp a book. However, not all of these grasping modes allow placing the book on the shelf. Thus, the way the reaching action is performed influences whether the delivery action is even feasible. One could imagine several ways to address this problem; for example, if the system has multiple arms, it could attempt re-grasping. To make the scenario even more complex, there could be objects that need to be moved out of the way, for the task to be performed successfully.

The scenarios above represent instances of mobile manipulation tasks, a term that is used widely today but still lacks a formal definition [11]. For mobile manipulation in particular, motion planning is complicated by the availability of multiple planning options: e.g., the robot can choose to simply use its arm to grasp an object, it can choose to use its base to move closer and then use the arm, but it can also choose to use both its arm and its base at the same time.

Contributions and Structure of the Paper. In order to state our contributions we first need to clarify some terms. *Task planning* is the process of computing a high level plan (task plan), or sequence of actions, that the robot needs to perform in order to achieve its goal. *Motion planning* is the computation of lower level plans to be used by controllers. Many of the actions from the higher level task plan are actually computation of motion plans. A *grasp reasoning system* is a system capable of proposing grasping poses for objects we are interested in manipulating.

This paper presents an approach which computes sequences of motion plans that solve a given task. Related work is discussed in Section II. Our work needs as input a task graph, as generated by a task planner [12]–[14]. The nodes of this graph can be annotated, when appropriate, with one or more grasping poses of the robotic system. These poses can be computed by a grasp reasoning system (e.g., [15]). We ignore issues that arise from uncertainty. All the assumptions we make are detailed later in Section III-A.

First, we introduce the notion of a *task motion multigraph* and show how it can be used to represent information pertinent to mobile manipulation (Section III). This is a formalization of available motion planning options when solving a task – a representation of the robot's capabilities in a form that can be used by motion planning.

Second, we present an algorithm that uses task motion multigraphs to compute a sequence of motion plans for solving a given task (Section IV). This algorithm attempts to reduce the total time spent planning motions by leveraging information from the task motion multigraph. Considering multiple planning options is computationally possible due to the advances in speed of computation for sampling-based motion planners [16], [17]. To demonstrate the utility of task motion multigraphs we apply the introduced algorithm to planning sequences of motions for the PR2 from Willow Garage, and compare it with a version of itself that only uses graphs. Conclusions follow in Section V.

II. Related Work

There has been a significant amount of research in motion planning (e.g., [16], [17]), grasping (e.g., [15], [18]), task planning (e.g., [12], [13]) and manipulation planning (e.g., [19]–[21]) that has lead to the current state of the art in mobile manipulation. In this section we discuss two lines of work and attempt to synthesize a pattern for each.

a) Work that builds upon the notion of manipulation graph. The methods included here combine motion and task planning to produce a system that solves given tasks by reducing them to a sequence of transit and transfer paths (e.g., [1]–[7], [19]–[22]). These paths are defined as two different types of edges in a *manipulation graph* [19]:

- *Transit path*: path executed by the robot without carrying any objects, avoiding collisions with the environment and with itself.
- *Transfer path*: path executed by the robot while carrying an object. The object is considered fixed to the robot along such a path and is treated as part of the robot for collision avoidance purposes.

b) Work that relies on the idea of guiding continuous exploration with discrete paths. This line of work performs motion planning in the state space of the robot and uses paths in the task graph as discrete guides (e.g., [23], [24]). The idea of considering discrete plans in the computation of lower level plans is used in the algorithm we propose. Similar concepts have been used for planning foot steps for humanoid robots (e.g., [25]) and climbing robots (e.g., [26]).

The line of work mentioned in *a)* is closer to our work. Each of the systems that falls under that category employs a representation of a *task graph* that encodes sequences of actions the robot could possibly perform to achieve its goal (see Figure 1-Left for an example). Typically, nodes in this graph denote states, and edges denote actions. Many of these actions require motion planning – computation of a transit or a transfer path. For our work, the distinction between transfer and transit paths is not important. We refer to it only for a more clear connection to previous work.

One of the most complex systems that follows the pattern described in *a)* is aSyMov [4]. We present this system in more detail, as it is representative. Planning can be performed for multiple robots, with potentially different capabilities, under symbolic and geometric constraints in an environment with movable objects. This is a very general form of the problem and allows for solving complex tasks. aSyMov uses a STRIPS-like [12] language to represent tasks. Because probabilistically complete [16] motion planners are used, maximum runtime bounds are imposed in order to make the algorithm terminate. Although performance is good for tasks that involve one robot operating on one object, the aSyMov paper shows the success rate of the method drops to 15% for tasks that involve two robots moving two objects, and computation can take more than a hundred seconds on current modern machines [4]. Even though the latter task is seemingly still simple, the number of sub-tasks and individual motion plans that have to be computed is high, which increases the running time.

The approach aSyMov follows is based on constructing Probabilistic RoadMaps (PRMs) [16], [27] for every robot and every movable object in the environment. A roadmap R_1 is connected to a roadmap R_2 by identifying a milestone in R_1 and creating a corresponding milestone in R_2. Use of these connected roadmaps effectively allows the extraction of transit and transfer paths.

aSyMov is not the only system capable of task and motion planning. Over the years, a number of other systems were proposed (e.g., [1]–[3], [5]–[7], [19]–[22]), with varying levels of interaction between task and motion planning. In [1], motion planning is used as a subroutine and information about its progress is not used at the task planning level. In [2], roadmaps are used in a similar fashion to [4]: they are connected by explicitly sampling the intersection of the state spaces they correspond to.

The work in [6] uses a hierarchical representation of tasks. At the lowest level of the hierarchy there are primitive actions. Among these actions there can also be algorithms that compute motion plans. This hierarchical representation speeds up the search at the task level. Information about the length of the paths produced by motion planning is used to provide optimal solutions, at the task level. A hierarchical representation for performing motion planning with temporal goals is shown in [28].

One of the contributions of [20] is that of showing how to compute a sequence of motion plans along a path from an input manipulation graph, in a manner that reduces computation along the path segments that are hard to plan for. This idea is further explored and generalized in this paper.

III. Representing Motion Planning Options

This section introduces the concept of a *task motion multigraph*, a data structure that represents explicitly the state spaces in which motion planning can be performed for a mobile manipulator in order to achieve its goal. This data structure is defined in terms of an intermediate notion, namely *task motion graphs*. A core issue not captured when representing tasks as graphs rather than multigraphs is that there are multiple ways of performing the same operation when the manipulator is mobile. For example, the robot can use its manipulator alone, it can move its base and then use its manipulator, or it can move both its base and its manipulator simultaneously. The experiments included in this paper show that it is possible to use information from task motion multigraphs and produce fast algorithms that compute sequences of motion plans necessary for solving given tasks.

A. Assumptions

We consider a single mobile manipulator – a robotic system with a mobile base and one or more arms.

We assume a task specification is available, in any of the variants suggested by previous work (LTL [29], STRIPS-like [12], etc.). Based on this specification, a task planner can construct an explicit task graph – a directed acyclic

graph that encodes the low level sub-tasks (actions) the robot performs. Since many variants of such graphs are present in the literature, we describe a generic version here. The graph has one root that represents the robot's initial state. Leaves of the graph are goals of the task planner. In some cases, the explicit construction is possible only if the horizon of possible actions is bounded. We believe this is a reasonable assumption for mobile manipulators operating in human environments. For simplicity, assume the only actions the robot can perform are `grip` (close gripper), `release` (open gripper) and `move_to` (plan a motion). The `grip` and `release` actions are very simple ones and do not include the computation of grasp poses. It is assumed that if grasp poses are necessary (which is typically the case), a grasp reasoning system (e.g., [15]) is employed at the time the task graph is generated and grasp poses are included in the graph's nodes. It is further assumed that such grasp poses, when specified, can be converted to states, or sets of states. Such computation can be performed, for instance, with inverse kinematics [30], [31].

B. Motion Planning Actions

Figure 1-Left shows an example task graph. This example encodes the task of delivering a book, while accounting for the possibility of having to move a cup of coffee out of the way, if delivering the book directly is not possible. In this paper, the `move_to` actions are of special interest since these are the ones that require motion planning. Contracting the edges that correspond to `grip` and `release` actions in the task graph leads to a simplification as shown in Figure 1-Right. We refer to this simplification as the *task motion graph*. This does not mean that the `grip` and `release` actions are not going to be performed in the execution of the task plan. Removing these edges is only a simplification that allows us to focus on the motion planning actions. The task motion graph encodes the different sequences of motions that would lead the robot to its goal. This is an intermediate notion used to define *task motion multigraphs*.

C. Task Motion Graphs and Task Motion Multigraphs

Let the mobile manipulator consist of a set of joints J. This implicitly defines a state space $\mathcal{X}_J$. All motion planning actions in the task motion graph are performed in some projection of $\mathcal{X}_J$.

Definition 3.1: Task Motion Graphs.

A task motion graph (TMG) for mobile manipulation is a directed, acyclic graph $G = (V, E)$ such that:

- $V = \{v | Q(v) \subset \mathcal{X}_J\}$. Every vertex v is associated with a set of states $Q(v) \subset \mathcal{X}_J$. $Q(v)$ can be explicitly specified as a set of states or implicitly specified in a manner that allows computation of states in $Q(v)$ (e.g., end-effector poses, which can be converted to states using inverse kinematics [30]).
- $E = \{e = (v_i, v_j) | v_i \neq v_j,\ v_i, v_j \in V\}$, and there exists an edge labeling function $label(e) = (Act, Env_e, \mathbf{A}_e)$. Act specifies an action that requires motion planning. For the purposes of this paper, Act is always `move_to`. At the start of the action, the robot is at a state $x \in Q(v_i)$ and at the end of the action, the robot is at a state $x' \in Q(v_j)$. Env_e defines the environment in which motion plans for edge e are to be computed. $\mathbf{A}_e = \{A_{e,1}, \ldots, A_{e,k} | A_{e,\cdot} \subseteq J, k \leq 2^{|J|}\}$ defines the possible sets of joints to plan for when computing motion plans along edge e.

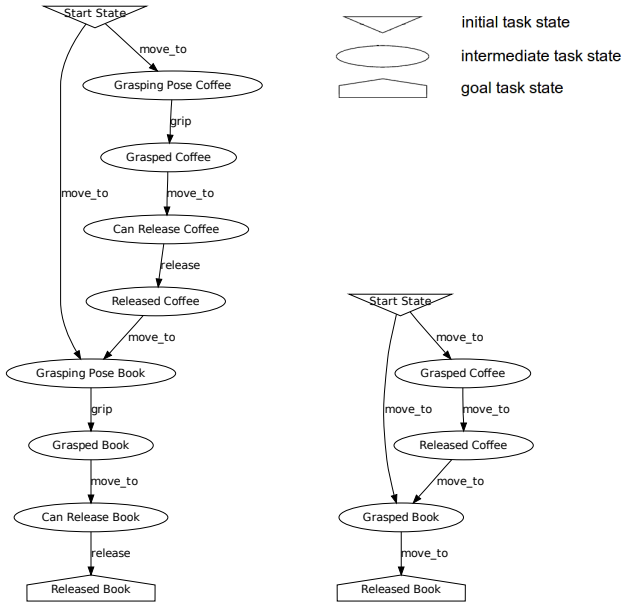

Fig. 1. Left: The task graph for delivering a book. It may be necessary for a cup of coffee to be moved out of the way to deliver the book. Right: The task motion graph – only actions that require motion planning are kept.

It is important to note that there can be multiple sets of joints that can be planned for when computing motion plans along an edge e. $\mathbf{A}_e$ is an input specified by the user. For example, if moving an arm's end-effector is intended, $A_{e,1}$ can be defined to be the minimal set of joints usually required to perform the operation: the joints in the actual arm. For more complex problems, it may be necessary to move the robot's base as well. For this reason, additional sets of joints ($A_{e,2}, \ldots$) can be included in $\mathbf{A}_e$. The intention is that planning can be attempted in any state space $\mathcal{X}_L$ (projection of $\mathcal{X}_J$), for $L = \bigcup_{j \in A} j$, $A \subseteq \mathbf{A}_e$.

Example: Consider a mobile manipulator with an arm with 7 joints and an omni-directional base that moves in plane. We define J_{arm} to be the set of joints in the arm and J_{base} to be a virtual joint with 3 degrees of freedom that corresponds to the $SE(2)$ state space ($\mathcal{X}_{J_{base}} = SE(2)$). We thus have $\mathcal{X}_{J_{arm}}$ 7-dimensional, $\mathcal{X}_{J_{base}}$ 3-dimensional and $\mathcal{X}_J$ 10-dimensional, $J = J_{arm} \cup J_{base}$. A simple TMG $G = (V, E)$ can have $V = \{v_{start}, v_{goal}\}$, $Q(v_{start}) = \{x_{start}\}$, $Q(v_{goal}) = \{x_{goal}\}$, defines the two vertices of the TMG: a start state and a goal state. $E = \{e = (v_{start}, v_{goal})\}$, $label(e) = (\texttt{move\_to}, Env, \{J_{arm}, J_{base}\})$. This specification defines a TMG that requires the computation of a motion plan from a specified start state $x_{start} \in \mathcal{X}_J$ to a state $x_{goal} \in \mathcal{X}_J$. The motion plan is to be computed in either $\mathcal{X}_{J_{arm}}$, $\mathcal{X}_{J_{base}}$ or $\mathcal{X}_{J_{arm} \cup J_{base}}$. Planning may not be feasible for all possible spaces. Env represents the environment considered for determining the validity of states (e.g., collision checking).

Definition 3.2: Task Motion Multigraphs.

Given a TMG, $G = (V, E)$, we define a task motion multigraph (TMM), $G_M = (V_M, E_M)$ as follows:

- $V_M = V$
- for every $e = (v_i, v_j) \in E$, $label(e) = (Act, Env_e, \mathbf{A}_e)$, let $E_{M,e}$ be a multiset, $E_{M,e} = \{e_{m,k} = (v_i, v_j) | k \in \{1, \ldots, 2^{|\mathbf{A}_e|}\}\}$ and $label_M(e_{m,k}) = (Act, Env_e, J_{e,k})$, for $J_{e,k} = \bigcup_{j \in \mathbf{a}(k)} j$, $\mathbf{a} : \{1, \ldots, 2^{|\mathbf{A}_e|}\} \rightarrow 2^{\mathbf{A}_e}$ is a bijection, $2^{\mathbf{A}_e}$ is the power set of $\mathbf{A}_e$.
- $E_M = \bigcup_{e \in E} E_{M,e}$.

In essence, a TMM is a TMG where all possible sets of joints used in motion planning are explicitly specified for each edge. The conversion, which can be done automatically, is fairly straightforward and only requires addition of edges. See Figure 2 for an example. The TMM reveals an additional layer of complexity for mobile manipulation: the need to decide which state spaces to plan in. This is a different type of decision to be made, in addition to other decisions such as selection of grasping poses. We introduce TMMs in an attempt to expose the need to consider which state spaces to plan in. This observation has been made in previous work as well [2], [4], but we present the first formalization. Since there is a combinatorial explosion in the construction of a TMM from a TMG (in terms of number of edges), it is desirable that the TMG is defined in a manner amenable to the robotic system this notion is used for. For instance, a robotic system with a mobile base and two arms may define three sets of joints to be used for planning: J_{left}, J_{right}, J_{base}, corresponding to the joints in the respective arms and the base. Edges in the TMG could then use $\mathbf{A}_e = \{J_{left}, J_{right}, J_{base}\}$. The definition of $\mathbf{A}_e$ implies that the option of planning in the full state space, $\mathcal{X}_{J_{left} \cup J_{right} \cup J_{base}}$, is included in the edges of the TMM, allowing in this case even the use of control theoretic techniques, if available.

Definition 3.3: A motion plan for a TMM.

A motion plan for a TMM $G = (V, E)$ is an ordered sequence of edges $P = \{e_1, \ldots, e_k\}, P \subseteq E$ such that for every edge $e = (v_a, v_b) \in P$ there exists a motion plan between some state $x_a \in Q(v_a)$ and some state $x_b \in Q(v_b)$. Furthermore, the motion plans for any two consecutive edges $e_i, e_{i+1}, 1 \leq i < k$ from P can be *connected*. The two motion plans are said to be connected if there exists a well-defined method (e.g., a controller) to move from the last state $x_{i,L}$ of the motion plan for e_i to the first state $x_{i+1,F}$ of the motion plan for e_{i+1}. For the purposes of this work the condition $x_{i,L} = x_{i+1,F}$ was imposed for connectivity to be achieved.

A motion plan $P = \{e_1, \ldots, e_k\}$ is a solution in a TMM $G = (V, E)$ if $v_b \in V$ is a leaf in G, with $e_k = (v_a, v_b)$.

Remark: Given a motion plan in a TMM, it is easy to see that a task plan in the original task graph can be constructed: the actions from the task graph that are not present in the TMM do not require motion planning. Only `grip` and `release` actions (closing and opening the end-effector) need to be re-inserted.

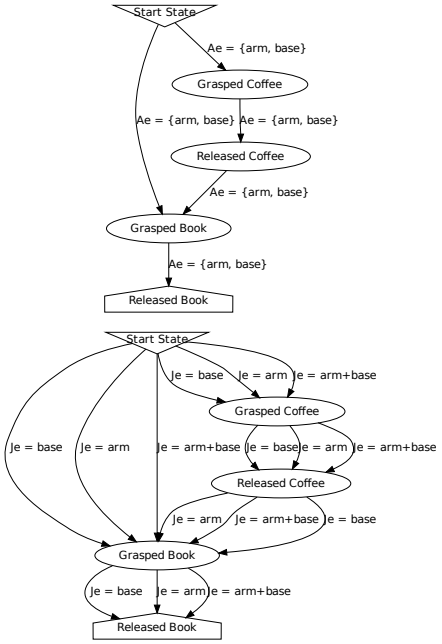

Fig. 2. Top: The task motion graph for delivering a book, defining the groups of joints $\mathbf{A}_e$. Bottom: The task motion multigraph. Edges define J_e.

IV. Computing Motion Plans on Task Motion Multigraphs

A. Intended Use

The intended use scenario of task motion multigraphs is as follows. The task planner generates a task graph with bounded horizon. This work does not concern itself with how this task planner operates or how the task graph is constructed: the existence of these components is assumed (e.g., [14]). A corresponding TMG is then constructed by discarding the actions that do not require motion planning. The TMG is converted to a TMM and a motion plan is computed for the TMM.

This paper provides a formalization of the available motion planning options in a manner that can aid with computation. The following section shows an example method that uses information contained in the TMM to compute motion plans. The availability of the TMM at the time of motion plan computation helps with the identification of less expensive but feasible sequences of motion plans. This information is used to attempt to reduce the amount of time spent planning motions.

B. Algorithm

Computing motions plans for certain edges may be more time consuming than for others: the complexity of the environments can vary, the set of joints J_e to plan for (and implicitly, the dimensionality of $\mathcal{X}_{J_e}$) may also vary. Planning motions along some edges may not even be feasible. These considerations make it obvious that computing the sequence of motion plans for some paths in the TMM can be much more computationally intensive than for others. To address this issue, we attempt to compute motion plans for the path that appears to be the cheapest. Which path appears to be the cheapest changes as the computation progresses.

The cost of a path is the sum of the costs of its edges. The cost of an edge e, $label(e) = (Act, Env_e, J_e)$ is:

$$cost(e) = \exp\left(1 + \frac{dim(\mathcal{X}_{J_e})}{\max_J dim(\mathcal{X}_J)}\right) \cdot scost(e)$$

$$scost(e) = \begin{cases} 1 & \text{if } sol \\ s \cdot (1+t) \cdot \left(1 + \frac{d_L(e)}{d_R(e)+d_L(e)}\right) & \text{if not } sol, \end{cases}$$

where $dim(\mathcal{X}_{J_e})$ is the dimension of the state space to be used for planning along edge e, $\max_J dim(\mathcal{X}_J)$ is the dimension of the largest state space considered by the TMM, s represents the number of times e was selected for motion planning (starts at 1), t is the number of seconds already spent planning motions along e, $d_L(e)$ represents the number of edges from e to the nearest leaf, and $d_R(e)$ represents the number of edges from e to the root. If motion plans along edge e are already available (sol is true), the cost of e relates only to the dimensionality of the state space, thus making edges that actuate fewer joints preferable. If no motion plans for e are available (sol is false), the cost of e is increased proportionally to the number of times e was selected for planning. Furthermore, the closer e is to a possible goal (a leaf), the fraction $d_L(e)/(d_R(e) + d_L(e))$ decreases, thus decreasing the cost of edges closer to possible goals.

The definition for the cost of an edge is heuristically determined, with the purpose of approximating how expensive paths are. The provided formula is intended as a guide and represents what worked well in our experiments. This is a topic for further investigation and several other approaches may be applicable (e.g., [20]). Determining better edge costs is an open issue that can further affect performance.

The computation of a task plan in a TMM proceeds as described in Algorithm 1. The body of the algorithm is an iterative process that runs motion planners on different edges of the TMM for short periods of time.

At every iteration, the set of segments that make up the path of least cost from the root to a leaf in G_M is computed using Dijkstra's algorithm (line 4). The edge along this path that is closest to the goal and that has no motion plan associated to it is then selected deterministically by *selectEdgeFromPath()* (line 5). Such an edge will exist as long as no complete solution has been found for the TMM. Motion planning is performed on the selected edge for a short duration (line 6). Repeated selections of the same edge continue the computation of the motion plan rather than restart it. Unless a motion corresponding to the selected edge is successfully computed, motion planning is executed on another edge, one selected from the remaining set of edges in the TMM (lines 8, 9). With low probability (10% in our work) *selectEdge()* selects an edge randomly; otherwise, priority is given to edges that have fewest number of computed motions, then to edges that have lowest cost. The intention is to be greedy and follow the path of least cost to the goal but at the same time allow the exploration of other options. Planning is performed for short durations Δt to allow updating edge costs more often.

For a vertex $v \in V$, let the reached states in $Q(v)$ be denoted by $R(v) \subseteq Q(v)$. A state x is reached if there exists a motion plan in the TMM that ends at x (see Definition 3.3). Initially, $R(v) = \emptyset$ for all vertices, except for the root: $R(root) = Q(root)$. For a new motion plan to be computed for an edge $e = (v_1, v_2)$, a starting state needs to be identified in $Q(v_1)$. A state $x' \in Q(v_1)$ can be used as a starting state for a motion plan along e if some state $x \in R(v_1)$ has been previously reached by a motion plan in the TMM, and x can be connected to x' using a well-defined means (in this work, $x = x'$). When a motion plan for an edge e is computed, the reached state in $Q(v_2)$ is added to $R(v_2)$. A diagram showing the computation of a motion is in Figure 3. If $R(v_1) = \emptyset$, no motion plan is computed for e. This is done to avoid computing motion plans that start at unreachable states.

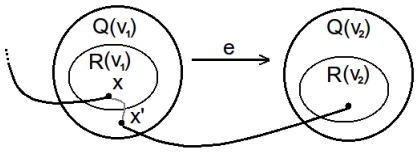

Fig. 3. Diagram showing a computed motion along edge $e = (v_1, v_2)$. The motion starts at state $x' \in Q(v_1)$, reaches a state in $R(v_2)$ and there exists a means to connect to x' from $x \in R(v_1)$.

Algorithm 1 TMM-Computation($G_M = (V_M, E_M)$)

```
1:  done = false
2:  noProgressIter = 0
3:  while not done and noProgressIter < H do
4:      P = shortestPath(G_M)
5:      edge = selectEdgeFromPath(P)
6:      solved = motionPlan(edge, Δt)
7:      if not solved then
8:          nextEdge = selectEdge(E_M \ {edge})
9:          solved = motionPlan(nextEdge, Δt)
10:     if solved then
11:         noProgressIter = 0
12:     else
13:         noProgressIter++
14:     done = haveCompleteSolution(G_M)
15: if done then
16:     return extractSolution(G_M)
17: else
18:     return nil
```

Termination for the algorithm is possible in two ways: either $done$ becomes true (line 14) and a solution is found, or sufficiently many iterations pass without any motion plan being computed (lines 3, 10–13), in which case the algorithm returns failure. We chose to limit the number of consecutive iterations that produce no motion plans to $H = 5$ in this paper. Similarly, an upper limit on computation time could have been imposed as well.

C. Experimental Results

In order to show the usefulness of TMMs, we compare the algorithm described in Section IV-B with a version of itself that only uses graphs. To obtain the graph version of the algorithm, we construct a TMM$^-$. The only change is that in Definition 3.2, instead of adding a multiset of edges

$E_{M,e}$ to the TMM$^-$ for every edge in the TMG e, $label(e) = (Act, Env_e, \mathbf{A}_e)$, a single edge is added to the TMM$^-$: e', $label(e') = (Act, Env_e, J_e)$, $J_e = \bigcup_{A \in \mathbf{A}_e} A$. On one hand, the graph version of the algorithm needs to consider the worst case scenario – plan for all the possible degrees of freedom – in order to offer the same generality as the multigraph version. On the other hand, the graph version of the algorithm has much fewer constructed edges.

We consider the task of moving two objects from their initial states to specified goal states using the method described in Section IV-B, in an environment with obstacles. Each object can be grasped in four different ways (see Figure 4 for a graphical representation). The robotic system we consider is the PR2 from Willow Garage. This robot includes two 7 degree-of-freedom (DOF) arms and an omni-directional mobile base. The PR2 needs to move from its initial position to grasp `object1` or `object2` with either of its arms. The PR2 then moves to the corresponding destination of the grasped object (`dest-object1` or `dest-object2`) and releases the object. The next step is to go back to the remaining object (`object1` or `object2`), grasp it, take it to its corresponding destination (`dest-object1` or `dest-object2`), and release it. Let J_{left} and J_{right} be the sets of joints in the respective arms and J_{base} be a virtual joint corresponding to the $SE(2)$ state space.

The sets of states for the nodes in the task graph are defined in terms of end-effector poses. Using inverse kinematics, each of these poses is converted to a number of states (the arm is 7-DOF, so there are multiple inverse kinematics solutions for a given pose). The poses are supplied by hand in this experiment, but in practice grasp reasoning systems that can propose such poses exist [15], [18].

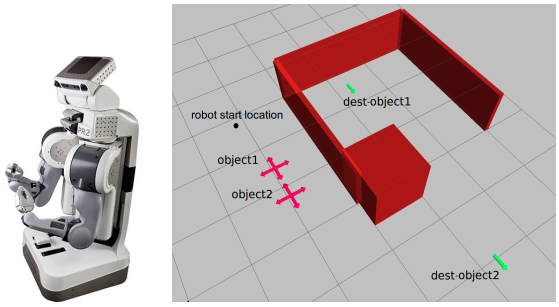

Fig. 4. Left: The used robot (PR2). Right: An environment with two objects that can each be grasped in four ways (in red) and the destinations of the objects (in green). The simulation is done using ROS.

Figure 5 represents the TMG for the problem described above. As actions can be performed with either the left or the right arm, $\mathbf{A}_e = \{J_{left}, J_{right}, J_{base}\}$ for all edges. While the TMG is sufficiently simple to produce by hand, the automatically constructed TMM is too large to include[1]. The motion planner used to solve this task (implementation of the *motionPlan()* routine) is the Probabilistic RoadMap (PRM) [27]. PRM is a proven technique for computing motion plans in high-dimensional spaces and has also been the choice of

[1]The TMM defines 160 ($20 \cdot 2^3$) distinct motion planning instances. The interested reader can view this TMM using our online resources: `http://kavrakilab.org/data/ICRA2011TMM/`

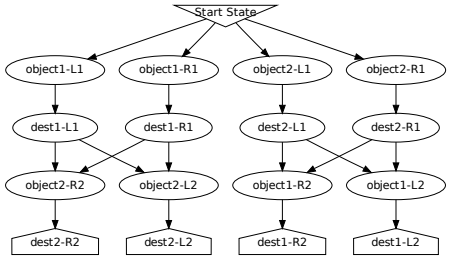

Fig. 5. The TMG for the task described in Section IV-C. Each edge is a `move_to` edge. "object$i$-Aj" denotes object i (1 or 2) being grasped by arm A (L=left, R=right) as the j^{th} object (first or second). dest$i$-Aj denotes object i being placed at its destination using arm A, as the j^{th} object.

previous related work. The implementation of PRM we used is from `OMPL` [32]. Table I shows the runtime of Algorithm 1, averaged over 30 runs, for the environment in Figure 4, with and without using a multigraph. The only parameter we vary between runs is the time allowed for motion planning along edges (Δt seconds). As we can see, the version of the algorithm using a TMM is faster by 20% or more, finding solutions in as low as 0.40 seconds (on average). The success rate was 100% for both the TMM and the graph version, when $\Delta t = 0.5s$, $\Delta t = 0.2s$, $\Delta t = 0.1s$. For $\Delta t = 0.05s$, the success rate was 93% for the TMM version and 83% for the graph version. Due to the probabilistic nature of the algorithms used to plan motions along edges in the TMM, time outs will be reached when no solutions exist. As TMMs encode multiple options for performing the same action, more planer instances will need to time out, thus increasing the time taken to report failure.

For very small values of Δt it is possible for the algorithm to give up the search for a solution even though one exists. This can occur since the probability of having H consecutive iterations that produce no motion plan for any edge increases as Δt decreases. For very low values of Δt it may be necessary to increase the value of H in the algorithm (used as 5 in this paper). For practical applications of this system, it is probable an adaptive scheme for selecting Δt is appropriate.

Increasing the value of Δt has no significant influence over runtime when a solution exists. This is to be expected since the *motionPlan()* routine returns when a solution is found (does not wait for Δt to elapse).

TABLE I

RUNTIME (IN SECONDS) FOR COMPUTING A MOTION PLAN FOR THE TMG IN FIGURE 5 IN THE ENVIRONMENTS FROM FIGURE 4.

$\Delta t = 0.05s$		$\Delta t = 0.1s$		$\Delta t = 0.2s$		$\Delta t = 0.5s$	
TMM	*graph*	*TMM*	*graph*	*TMM*	*graph*	*TMM*	*graph*
0.51	0.63	0.52	0.66	0.50	0.68	0.40	0.70

The main factor that allows reducing total planning times is that all motion planning options can be considered simultaneously. This information is available in the TMM. Having this information at the motion planning level creates the opportunity of quickly switching the sequence of motion plans that is presumed to solve the task, without having to compute complete motion plans for all edges in the TMM. Once a sequence of motion plans is computed entirely, a solution has been found.

It is important to remark that no effort was made to tune the algorithm using TMMs and no problem specific knowledge was used. For every edge in the TMG, 2^3 edges were created in the TMM, although it is certainly possible to use fewer options. Furthermore, faster algorithms that are not probabilistically complete (such as decoupled planners) could be used as well. If knowledge about the task to be solved is known in advance, this can be used to bias the search for paths in the TMM. In effect, a multitude of avenues exist to improve the results presented in this work.

V. Conclusions

In this paper, we introduced the notion of a task motion multigraph (TMM). TMMs are used to formalize an inherent difficulty of mobile manipulation and encode it in a form that can be used when computing motion plans. Mobile manipulators have the ability to perform tasks in multiple ways, using different hardware components. This implies that motion planning can be performed in different state spaces – projections of the complete state space of the robot – leading to the same goal.

We have also presented an algorithm that can quickly compute sequences of motion plans using information from the TMM. We have shown that even with simple definitions of costs assigned to the edges of the TMM, the use of multigraphs leads to definite performance improvements as compared to using graphs.

TMMs can provide a coherent framework that allows better understanding of available motion planning options when computing task plans. For future work, it would be interesting to integrate the proposed method with a task planner to produce a complete planning system. This would also allow for a more extensive experimental evaluation of the approach.

Acknowledgements

Work on this paper by Ioan Şucan and Lydia Kavraki was supported in part by NSF IIS 0713623, NSF DUE 0920721, NSF CCF 1018798, a Sloan Fellowship to LK and Rice University funds. The experimental part of this work was supported in part by the Shared University Grid at Rice funded by NSF under Grant EIA-0216467, and a partnership between Rice University, Sun Microsystems, and Sigma Solutions, Inc.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *Intl. Journal of Robotics Research*, vol. 17, pp. 315–337, 1998.

[2] K. Hauser and J.-C. Latombe, "Integrating task and PRM motion plannning," in *Intl. Conf. on Automated Planning and Scheduling*, 2009, workshop on Bridging the Gap between Task and Motion Planning.

[3] J. Guitton and J.-L. Farges, "Taking into account geometric constraints for task-oriented motion planning," in *Intl. Conf. on Automated Planning and Scheduling*, 2009, workshop on Bridging the Gap between Task and Motion Planning.

[4] S. Cambon, R. Alami, and F. Gavrot, "A hybrid approach to intricate motion, manipulation and task planning," *Intl. Journal of Robotics Research*, vol. 28, pp. 104–126, 2009.

[5] J. Choi and E. Amir, "Combining planning and motion planning," in *IEEE Intl. Conf. on Robotics and Automation*, Japan, 2009, pp. 238–244.

[6] J. Wolfe, B. Marthi, and S. J. Russell, "Combined task and motion planning for mobile manipulation," in *Intl. Conf. on Automated Planning and Scheduling*, 2010.

[7] L. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *IEEE Intl. Conf. on Robotics and Automation*. Alaska: Workshop on Mobile Manipulation, 2010.

[8] C. Borst, C. Ott, T. Wimbock, B. Brunner, F. Zacharias, B. Baeum, U. Hillenbrand, S. Haddadin, A. Albu-Schaeffer, and G. Hirzinger, "A humanoid upper body system for two-handed manipulation," in *IEEE Intl. Conf. on Robotics and Automation*, Italy, 2007, pp. 2766–2767.

[9] S. Srinivasa, D. Ferguson, M. V. Weghe, R. Diankov, D. Berenson, C. Helfrich, and H. Strasdat, "The Robotic Busboy: Steps Towards Developing a Mobile Robotic Home Assistant," in *Intl. Conf. on Intelligent Autonomous Systems (IAS-10)*, July 2008.

[10] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey, and E. Berger, "Autonomous door opening and plugging in with a personal robot," in *IEEE Intl. Conf. on Robotics and Automation*, Alaska, 2010.

[11] "Technical committee on mob. manip." http://mobilemanipulation.org.

[12] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice-Hall, Englewood Cliffs, NJ, 2003.

[13] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, 2004.

[14] J. Boren and S. Cousins, "The SMACH high-level executive," in *IEEE Robotics & Automation Magazine*, December 2010, vol. 17, pp. 18–20.

[15] A. Miller and P. K. Allen, "GraspIt!: a versatile simulator for robotic grasping," *IEEE Robotics and Automation Magazine*, vol. 11, 2004.

[16] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, June 2005.

[17] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[18] M. Ciocârlie, K. Hsiao, E. G. Jones, S. Chitta, R. Rusu, and I. Şucan, "Towards reliable grasping and manipulation in household environments," in *Intl. Symposium on Experimental Robotics*, 2010.

[19] R. Alami, J.-P. Laumond, and T. Siméon, "Two manipulation planning algorithms," in *Intl. Workshop on the Algorithmic Foundations of Robotics*, 1994.

[20] C. L. Nielsen and L. E. Kavraki, "A two level Fuzzy PRM for manipulation planning," in *Intl. Conf. on Intelligent Robots and Systems*, Takamatsu, Japan, 2000, pp. 1716–1722.

[21] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *Intl. Journal of Robotics Research*, vol. 23, 2004.

[22] T.-Y. Li and J.-C. Latombe, "On-line manipulation planning for two robot arms in a dynamic environment," *Intl. Journal of Robotics Research*, vol. 16, pp. 144–167, 1997.

[23] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *IEEE Intl. Conf. on Robotics and Automation*, Alaska, 2010.

[24] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE Intl. Conf. on Robotics and Automation*, Alaska, 2010, pp. 2689–2696.

[25] J. Chestnutt, M. Lau, K. M. Cheung, J. Kuffner, J. K. Hodgins, and T. Kanade, "Footstep planning for the Honda ASIMO Humanoid," in *IEEE Intl. Conf. on Robotics and Automation*, Spain, April 2005.

[26] K. Hauser, T. Bretl, J.-C. Latombe, and B. Wilcox, "Motion planning for a six-legged lunar robot," in *Intl. Workshop on the Algorithmic Foundations of Robotics*, New York City, July 2006.

[27] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, August 1996.

[28] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, pp. 343–352, 2009.

[29] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, 2000.

[30] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005.

[31] J. M. Ahuactzin and K. Gupta, "The kinematic roadmap: a motion planning based global approach for inverse kinematics of redundant robots," *IEEE Transactions on Robotics and Automation*, vol. 15, pp. 653–669, 1999.

[32] "The Open Motion Planning Library," http://ompl.kavrakilab.org.