NAVIGATION AMONG MOVABLE OBSTACLES : REAL-TIME REASONING IN COMPLEX ENVIRONMENTS

MIKE STILMAN [1]
JAMES J. KUFFNER [1,2]

[1] *Robotics Institute, Carnegie Mellon University,*
5000 Forbes Ave. Pittsburgh, Pennsylvania 15213, USA
robot@cmu.edu, kuffner@cs.cmu.edu

[2] *Digital Human Research Center,*
National Institute of Advanced Industrial Science and Technology (AIST),
2-41-6 Aomi, Koto-ku, Tokyo, Japan 135-0064

In this paper, we address the problem of Navigation Among Movable Obstacles (NAMO): a practical extension to navigation for humanoids and other dexterous mobile robots. The robot is permitted to reconfigure the environment by moving obstacles and clearing free space for a path. Simpler problems have been shown to be P-SPACE hard. For real-world scenarios with large numbers of movable obstacles, complete motion planning techniques are largely intractable. This paper presents a resolution complete planner for a subclass of NAMO problems. Our planner takes advantage of the navigational structure through state-space decomposition and heuristic search. The planning complexity is reduced to the difficulty of the specific navigation task, rather than the dimensionality of the multi-object domain. We demonstrate real-time results for spaces that contain large numbers of movable obstacles. We also present a practical framework for single-agent search that can be used in algorithmic reasoning about this domain.

Keywords: NAMO; navigation; manipulation; planning; motion; movable obstacles.

1. Introduction

Advances in humanoid robotics have brought about the development of software systems for robot interaction with real-world environments. Furthermore, architectures for motion planning have allowed humanoid robots to construct adaptive plans for walking, grasping and other primitive behaviors. These developments pave the way towards higher level planning that enables the exhibition of human-like reasoning in addition to human-like physical action.

In this paper, we explore the domain of *Navigation among Movable Obstacles* (NAMO). A human planning to move through a cluttered space does not simply avoid every obstacle. Rather, the human will often reposition chairs, doors and other movable objects while navigating. These actions open free-space for paths when a plan through the original space is difficult or impossible. For humans, the ability to move objects is an integrated component of navigation. For robots, although numerous algorithms for force control and ZMP-based walking have enabled robots to conform to variations in the environment, little work exists that would allow robots *to conform the environment to the robot's goals.*

Not only is NAMO a natural extension to the current capabilities of humanoid robots, it presents challenging research problems for theoretical motion planning, as well as practical applications ranging from industrial manipulation to urban search and rescue. Pure locomotion or manipulation problems generally address planning among a fixed set of rigid objects. We are concerned with a world rich in objects, some of which may need to be repositioned in order for the robot to achieve its goal. Thus, in addition to planning locomotion, the robot must decide which objects need to be moved, if any, and solve the manipulation problem required to move them. The ability to perform such analysis is not only applicable to small-scale office environments, but also to complicated unstructured navigation. For instance, current work in *urban search and rescue* focuses on small mobile robots or snakes. These robots are highly effective at search, since they can access obstructed environments by avoiding obstacles.[1] However, *rescue efforts* still largely rely on humans entering dangerous scenes. This is precisely due to the human ability to intuitively determine safe methods for clearing paths and our dexterity in carrying out such plans. Although replacing the human in hazardous search and rescue may be a long term initiative, we propose the NAMO domain as an initial step from locomotion towards this fascinating research area. In our work, we examine a practical though simplified formulation of the NAMO problem. We discuss potential variations in the problem statement, the notion of optimality, as well as descriptions of sub-problem classes. Finally, we present a prototype planner that takes advantage of the underlying navigational $\mathcal{C}$-space to construct real-time intuitive solutions to a wide range of NAMO problems.

2. Problem Statement

For simplicity in algorithmic analysis we restrict our domain to a planar projection of a continuous three dimensional environment. The environment contains:

- R - a humanoid or dexterous mobile robot equipped with a gripper arm.
- $\mathcal{L}$ - a set of polygonal *Fixed Obstacles* that the robot must avoid in navigation.
- $\mathcal{M}$ - a set of polygonal *Movable Obstacles* that can be manipulated by the robot through the application of forces at allowable contact points. Each $O_i \in \mathcal{M}$ has planar dynamic properties: *Center of Mass, Mass and Moment of Inertia.*

Given an initial configuration of the robot and environment, the robot's task is to assume a target configuration or *goal.* A solution should consist of a motion plan that iterates walking, grasping and moving obstacles until the robot is at the goal.

We consider that an optimal plan for the NAMO domain will first move the *least number of obstacles* and second *use the least amount of Work.* We define $Work$ in the standard dynamic sense of $Work = Force \times Distance$. There are many possible practical extensions to the definition of optimality. For example, a notion of fragility could be introduced to indicate the risks associated with moving an obstacle.

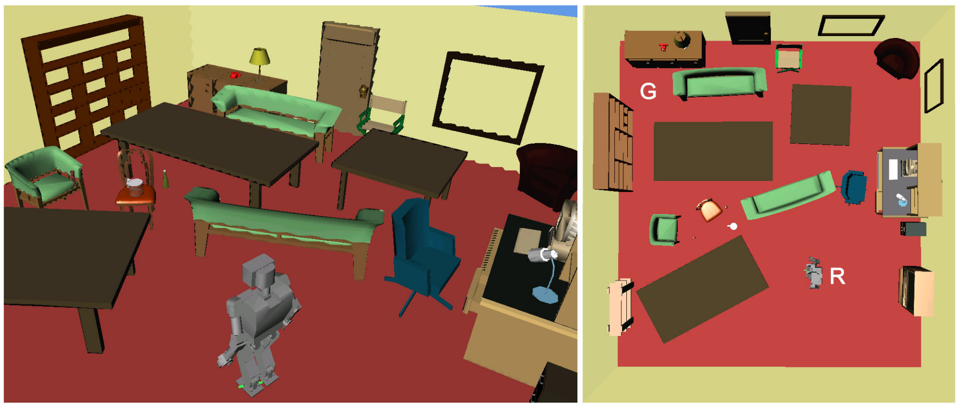

Fig. 1. A sample NAMO domain. All depicted objects are internally represented with polygons.

3. Related Work

At first glance, Navigation among Movable Obstacles is an instance of the *Motion Planning* domain. However, the complexity of this problem is prohibitive to the conventional application of complete motion planning algorithms. Consider a closed, planar world that contains a robot and N obstacles. Suppose we attempt a resolution-complete version of the problem using search over a discretized 2-D planar projection of the 3-D world. Let the configuration of the robot base be represented by $\mathcal{C}_R = (x, y, \theta)$, with resolution m in x, y and c in θ. The generic size of the robot $\mathcal{C}$-space $|\mathcal{C}_R| = O(m^2c)$. Analogously, for each object $|\mathcal{O}_i| = O(m^2c)$. The full motion search space is the product of these subspaces, $\mathcal{C}_R \times \mathcal{O}_1 \times \mathcal{O}_2 \times \ldots \times \mathcal{O}_n$, and therefore has $O(m^{2N+1}c^{N+1})$ world states. Furthermore, for K directions of object motion, the branching factor of a brute force search is NK. If $K = 6$, $N > 5$ (a rather primitive problem), this domain quickly surpasses reasonable limits of computational resources of memory and time. Gordon Wilfong first showed that a simplified variant of this domain is NP-hard.[2] More recent work has demonstrated NP-completeness results for trivial problems where blocks can only be pushed on a planar grid.[3]

The challenges of this domain are not restricted to a large search space. We are also required to consider navigation, environment contact and manipulation of the obstacles. For a comprehensive discussion of NAMO, let us consider works in these fields and the aspects of our problem that they address. In terms of navigation, our robot is asked to work in a configuration space that will change over time. In particular, certain obstacles may move and therefore alter the space.

Previously, obstacles moving along specified trajectories was a problem addressed by bounding the velocities of the obstacles and augmenting the configuration space with time.[4] In doing so, a point in the free space ensures that a configuration is valid at the given time in which it takes place. This approach has been extended to kinodynamic domains,[5] as well as real-time deformable plans.[6] Work along these lines, however, focuses on a world model which the robot cannot control. In that sense, the problems addressed can still be formulated as existence of paths through an expanded state space.

Models of the world that support *robot interaction* are typically in the form of manipulation or assembly planning. For instance, Mason and Lynch developed bounds for stable pushing motions by a mobile robot with edge-edge contact between the robot and a specified object.[7,8] They introduced a search algorithm that would branch on possible motions within these bounds and yield stable plans for repositioning the object among obstacles. Alami et.al. pursued a similar problem with rigid grasping by constructing graphs of transit and transfer(manipulation) paths for robot/object motion.[9] This work was recently extended by Simeon et.al. to handle continuous grasps and placements.[10] Alternatively, researchers in humanoid robotics have shown the potential for generalizing human motion capture in deciding manipulator contact placement and quasi-static manipulation planning.[11] Work along these lines is essential for implementing a NAMO planner on an actual robot, however it does not address the higher-level problem of deciding motions for large numbers of objects as required by our domain.

The majority of research that deals with interactions that involve *multiple objects* is in assembly planning. Yet, the primary focus of this field diverges from the characteristics of NAMO. Assembly planners generally focus on separating a collection of parts and typically ignore the robot/manipulator. Planning motions allow unassembled parts to be removed to “infinity.”[12,13] One of the largest difficulties of the movable obstacle domain is strictly due to the unintended interactions between obstacles and the robot, since both are required to stay within a constrained space. This complication is addressed by a less studied but highly relevant extension of assembly planning known as the *rearrangement problem*.[14] Rearrangement is a number of real-world generalizations of the Sokoban puzzle: a robot is required to move a set of objects between specified initial and final configurations.[15] Although the task of the robot is akin to the earlier mentioned manipulation work, the key difference is that it may not be possible to directly achieve the individual object placement goals. Rearrangement planners typically construct precedence graphs and determine intermediate configurations for obstacles. Since intermediate configurations are not specified in the problem statement, the curse of dimensionality in this problem is very similar to NAMO. The most experimentally successful rearrangement planners employ opportunistic methods in selecting object perturbations that allow for simplified planning.[14,16]

All the mentioned developments in navigation, manipulation and assembly address problems with specified final configurations. To our knowledge, the only existing planner that deals with many movable objects and specifies a single navigation goal was developed by Chen and Hwang (CH).[17] In this work, the authors allow the robot to “*shove_aside*” and “*push_forward*” obstacles as the robot moves towards the goal. Their solution consists of a navigation trajectory which minimizes the associated cost of moving obstacles away from the robot. To generate this solution, they introduce a global planner that heuristically evaluates the cost of moving obstacles away from randomly selected points in the domain. The planner then searches a graph of neighboring points to form a trajectory of least cost. This trajectory is verified by a local planner that can apply manipulation primitives to connect two neighboring points. CH is effective on some examples, yet it has some important drawbacks. For each robot location that is found to be reachable, the algorithm considers only one possible trajectory that led to that event. The authors recognize this lack of backtracking as a cause for incompleteness and show that even slight improvements to their framework would cause exponential growth in complexity. Furthermore, allowing all colliding objects to move jointly narrows the solution space to *plowing* paths. All objects are greedily pushed away from the robot, allowing it to move forward. In particular, since manipulation is restricted to validating the connectivity of *neighboring points*, manipulation tasks that affect distant portions of the world are never considered. For instance, in Figure 5, moving the couch to open a path on the opposite side of the couch would not be addressed by this planner. Although these constraints make the CH algorithm difficult to extend, it serves as a baseline for development in the NAMO domain.

4. Overview

In the following sections, we will introduce the fundamental characteristics of NAMO planning and describe the key tools and methods for approaching this domain. In Section 5, we discuss the choices for contact and actions that define a suitable action-space for the robot. In particular, we focus on developing a space that is well suited for real-world actions, yet can still be analyzed as a planning domain. Section 6 develops a framework for reduced dimensionality analysis of our large action space. We introduce the navigational $\mathcal{C}$-space as both a tool for bounding the complexity of object interactions, and a heuristic for guiding search. Finally, Section 7 presents a simple and informative prototype planner based on the previous discussion.

We show that our planner could be employed in real-time for improved robot navigation. Due to the structure of the planning methodology, our algorithms can easily be extended to handle future developments in multi-object reasoning. We discuss open problems and directions of research that would greatly contribute to the power of planning in NAMO.

5. Contact and Motion Primitives

The primary focus of our work is to develop algorithms that take maximum advantage of the robot's dexterity and the environment structure. In order to simplify the manipulation problem, most manipulation planners restrict the domain to translational motion, pushing or prehensile manipulation.[7,14,16] Yet, while reducing the action space, these planners also reduce the solution space. In fact, many real-world NAMO problems have simpler solutions when the motion of obstacles is not

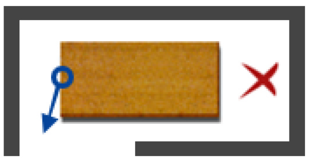

Fig. 2. A large, constrained object requiring manipulation motions that consider dynamics.

restricted.

Figure 2 is one of many examples where a large object, such as a table, is highly constrained in motion. The size of this object makes prehensile manipulation (constraining all the degrees of freedom) difficult. Furthermore, pure translation or pushing motions alone cannot solve this problem. For a human, the intuitive approach is to grasp the table at a convenient location, as shown in Figure 2. Then, *for select directions of pulling/pushing, the natural dynamics of the table will generate the complex motion* that guides it away from the constraints. Our formulation not only adds point contact and pulling, but develops a mapping that projects simple linear robot motion to complex transformations of the object.

To be precise, motion may be defined by sets of point contacts at vectors **P** from the object COM and forces **F** applied by the robot. If we allow only single point contacts to act on the object at a given time, then the mapping from the action space of the robot to the associated planar state-space transitions of obstacles my be modeled as:

$$\dot{\nu} = \frac{\mathbf{F}}{M} \tag{1}$$

$$\ddot{\theta} = \frac{|\mathbf{F} \times \mathbf{P}|}{I} \tag{2}$$

In our simulation we also add viscous friction to the objects. This description of robot motion appears compatible with the planar physical capabilities of a dexterous mobile robot. It does not, however, conform to the statically stable pushing properties discussed by Lynch.[7] We assume that for slow quasi-static motion, a local robot controller will be able to compensate for environment modeling error. The mapping in Eqs. 1,2 highlights the two aspects of object manipulation that we must address. Namely, how do we select the contact point **P** and the force **F**?

5.1. *Contact Points*

There is substantial work in grasping and manipulator/object contact.[18,19] We assume that within the local workspace of the robot base, existing methodology can be used to implement a desired grasp of an obstacle at a desired contact point. Hence, our primary concern is to define valid contacts for a mobile robot whose manipulator configuration space is largely constrained by the free-space of the robot base. We define the following sets of points for each movable obstacle O:

$P(O) = \{\text{The set of all points on the object}\}$

> $CP(O) \subset P(O)$ $\{p \in CP$ can potentially be grasped by the robot$\}$

Natural choices for CP are the edges of a polygonal obstacle, or other line segments that can be identified as graspable for a more involved obstacle definition.
For each $p \in CP(O)$ we define:

> $F(p)$ - the set of all possible forces that can be exerted on point p.
> $F'(p) \subset F$ - the subset of forces on point p that can be exerted without directional slip or loss of contact between the end-effector and the obstacle. $F'(p)$ restricts the set of possible robot interactions with any point of the obstacle. It determines whether the obstacle can be pushed or pulled from a given contact location.
>
> $Reach(p)$ - the set of possible standing configurations for the robot base from which the robot would be able to make manipulator contact with the point p. Analogously, $Reach(O)$ is the set of base configurations that allow the robot to reach some $p \in CP(O)$.

Let $\mathcal{C}$ be the configuration space of the robot base R_{COM}, i.e. the space that we wish to navigate. Let $Acc(R_{COM}) \subset \mathcal{C}_{free}$ be the currently accessible free-space at the time of the grasp. Then, for a given obstacle, the possible grasps are defined as follows:

> $AReach(O) = Reach(O) \cap Acc(R_{COM})$ is the set of all accessible configurations of the base that allow the robot to grasp the obstacle. For a particular base configuration $b \in AReach(O)$, and a point $p \in CP(O)$ where $b \in Reach(p)$, the following holds:
> *The robot with base configuration b can exert forces $F'(p)$ on point $p \in CP(O)$.*

5.2. *Obstacle Motion through Robot Motion Primitives*

In order to search the space of robot interactions with obstacles, we can choose to consider all potential wrenches on the obstacle's COM and then find corresponding robot actions that would make them feasible. Although this would be resolution complete with respect to all motions of the object, it also would be computationally expensive due to the necessary search through possible base configurations and actions for each wrench.

We choose an alternative formulation: sampling the possible contact points in $CP(O)$ and evaluating the intersection $Reach(O) \cap Acc(R_{COM})$. This procedure yields the set $AReach(O)$ of accessible R_{COM} configurations from which O is graspable and the associated grasp points p. The action space is then directly computed by associating robot action forces with each pair (R_{COM}, p).

So far we have defined contacts for robot forces and assumed an existing set of robot actions. Yet, for a high-DOF robot such as a humanoid, the complete set of possible actions is extremely large. Furthermore, many of the robot's actions are redundant with respect to the object and even the end-effector point. This suggests the implementation of a small set of parametrized primitive actions that best span the space of object manipulation.

Consider the way in which humans manipulate objects. For small objects, humans remain stationary and apply torques and forces directly to the object. For large objects, we intuitively move our COM and apply forces using our arms, which act as soft constraints on the motion of the object. In this work, we address the latter case of larger objects with single-point contact. Suppose the robot base is at configuration R_{COM} and the end-effector is attached to point p on the obstacle. We allow purely translational accelerations of the robot's COM in any direction. The robot's manipulator imposes a non-linear spring/damper distance constraint between the robot's COM and the point of contact p. Eq. 4 resembles the shallow exponential stress-strain curve of passive human muscle.[20] $\mathcal{D}$ is the desired distance, $\mathbf{U}(V) = V/|V|$ and k_p, k_v are gains.

$$p_d = R_{COM} + \mathcal{D}\mathbf{U}(p - R_{COM}) \tag{3}$$

$$\mathbf{F} = k_p(e^{p_d - p} - 1) - k_v \nu. \tag{4}$$

The action space is represented by translational motions of R_{COM} in some discrete number of directions. Given an initial (R_{COM}, p), *Eq.* 1-4 can be numerically integrated to determine actions in the domain of obstacle motion. For fixed time steps and accelerations, robot actions are defined by the triple (R_{COM}, p, d), where d is the direction of motion. The actual interactions at point p are restricted to $F'(p)$ and limited to obstacle motions that may have rotational slip but not translational slip between the end-effector and the contact point. Motions for smaller objects can be represented similarly, by directly moving the desired contact point.

This formulation for primitive actions yields intuitive behavior in simulation (Figure 6). On a robot platform, we assume that the spring/damper system can be implemented through force-control and/or natural dynamics of the robot. Still, the problem of selecting *a globally optimal set of actions* that span the manipulation space without redundancy remains an open and interesting topic for future work.

6. NAMO Planning

In the previous section, we formulated a reduced search space that is restricted to feasible real-world actions of a dexterous mobile robot. In our planner we demonstrate how primitive actions can generate interesting solutions for the NAMO domain. Theoretically, we could implement a breadth-first forward planner that would apply the primitive actions. This would be both resolution complete and metric optimal, yet practically intractable due to the large branching factor that results from applying all feasible actions to all reachable obstacles.

This motivates a shift in discussion from feasible actions to *useful* ones, as well as the value of moving particular obstacles. In this section we develop tools

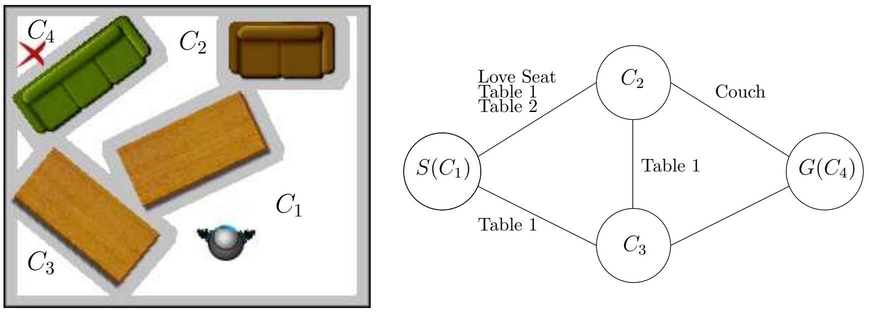

Fig. 3. A simulated representation of the dynamic R_{COM} configuration space. The labeled subspaces C_1-$C_4 \subset \mathcal{C}_{free}$ are disjoint components of free space. $Acc(R_{COM}) \equiv C_1$. On the right is the associated DFG with labeled components of Figure 3. Note: $OL(3, G) \equiv \emptyset$.

for bounding the complexity of NAMO, and heuristic methods for dramatically expediting forward planning.

6.1. *Domain Observations*

In Section 5.1, we defined $\mathcal{C}$ as the configuration space of R_{COM}, the robot base. Suppose the robot can be safely bounded by a polygonal discretization of a circle. We can always project an instance of NAMO into the planar $\mathcal{C}$-space of navigation. This is accomplished by the standard Lozano-Perez convolution of the circular robot bounds and the obstacles.[21] We use a circle to bound the robot because the robot and obstacles may rotate and the robot should preserve a safe distance regardless of configuration. Figure 3 illustrates an instance of NAMO. The obstacle bounds are expanded to represent their associated obstacles in $\mathcal{C}$.

Notice that the navigational free-space is disjoint. Let $\mathcal{C}_{free}$ represent all the space that R_{COM} could occupy. $Acc(R_{COM})$ is the subspace of $\mathcal{C}_{free}$ that can be reached solely by navigation from the current base configuration. In a NAMO domain, the initial $Acc(R_{COM})$ does not contain the goal. This points to an alternative formulation of our problem: *The task of the robot is to manipulate objects such that the goal configuration is added to* $Acc(R_{COM})$.

In Figure 3, $\mathcal{C}_{free}$ consists of five disjoint components (four of which are labeled C_1-C_4). Moving the table and then the couch would respectively add C_2 and C_4 to $Acc(R_{COM})$, accomplishing the NAMO task. The illustrated problem falls into a subclass of NAMO which we label *linear* or LP_1 by analogy to rearrangement planning.[14] LP_1 is the class of problems for which disconnected components of free-space can be connected independently by moving a single obstacle. Let us call a *keyhole* the subproblem of moving one or more objects to connect two components of $\mathcal{C}_{free}$. Our usage of "independently" implies that a given solution to a single *keyhole* does not affect the solvability of any following subproblem.

Although LP_1 covers a large number of real-world problems, it is not difficult to construct examples that lie outside this subclass. These examples are characterized by obstacles that do not directly separate two components of $\mathcal{C}_{free}$ but constrain the motion of the separating obstacle. LP_k would allow moving k interacting obstacles to independently resolve a *keyhole*. It is also possible for obstacle motion that solves one keyhole to interfere with the solution of another by constraining the motion of other obstacles or directly detaching a component of $\mathcal{C}_{free}$.

6.2. *Utilizing the Underlying Navigational Structure*

Section 5 identified a search space for planning the motions of an object. This space is parametrized by the accessible contact points $AReach(O)$ and the motions of the robot after grasping. Directly searching the action space for a solution to NAMO is infeasible. However, the introduction of navigational structure allows us to construct subproblems in the action space that can be solved quickly and optimally. In this section we develop GraphConnect - a simple, resolution complete and metric optimal algorithm that takes advantage of $\mathcal{C}_{free}$ structure to solve problems in LP_1.

For any problem in LP_1, GraphConnect constructs a plan as follows: Define a set DF of disjoint $C_i \subset \mathcal{C}_{free}$ components, where S and G are components containing the robot and goal respectively. We construct a graph DFG where each node is an element of DF. Each edge e, connecting (C_i, C_j) of DFG, is associated with a list $OL(e)$ of obstacles that face both C_i and C_j. We remove all edges in DFG for which $OL(e) \equiv \emptyset$.

Let SFG be a graph that only contains node S. We grow SFG as follows: Select a node C_1 in SFG. Select an edge e in DFG between nodes C_1 and C_2, where $C_2 \notin SFG$. Now, assuming the robot is located in $C_1 \subset \mathcal{C}_{free}$, we try to connect C_1 to C_2. To do this, we individually consider the motions of each obstacle in $OL(e)$ by means of the subprocedure Manip-Search(C_1, C_2, O). If the two components of free-space can be connected, the edge s and node C_2 are added to SFG. Otherwise edge e is removed from DFG. If more than one obstacle $O \in OL(e)$ can be moved successfully, we select the plan requiring the least $Work$. This process is iterated until we successfully add G to SFG, or we fail, noting that all connected nodes of DFG have been added to SFG.

Manip-Search(C_1, C_2, O): This routine is called with a *keyhole* subproblem of NAMO consisting of two disjoint components of $\mathcal{C}_{free}$: C_1, C_2 and an obstacle O. It returns a motion plan for a robot in C_1 that connects C_1 and C_2 by manipulating O, or Nil on failure. As described in Section 5, we can sample $AReach(O)$ and associated contact points $p \in CP(O)$ that are reachable by a robot in C_1. A simple Manip-Search implementation will then conduct a bounded breadth-first search of the robot action space after grasping the object at each p. The sequence of actions connecting C_1 and C_2 yielding the least amount of $Work$ is returned.
Note that a successful Manip-Search changes the state of the world S_W by displacing the robot and an object. The new S_W is returned by Manip-Search

and stored in SFG along with the added node C_2. Further calls to Manip-Search to connect C_2 with some $C_3 \notin SFG$ will be performed in the altered S_W. This ensures the validity of the sequence of motions found by our planner. Although Manip-Search is relatively straight forward, the task of recognizing that the goal has been achieved is non-trivial. We discuss this challenge, and provide one possible solution in Appendix A.

Our description of GraphConnect illustrates the potential for using a $\mathcal{C}$-space decomposition of NAMO problems to construct small subproblems that can easily be solved by a motion planner. Furthermore, the construction of GraphConnect allows for a simple proof of its relationship to the LP_1 problem class.

Lemma 1. GraphConnect *is resolution complete for problems in* LP_1*.*

Proof. Let Π be a solvable problem in LP_1. We show that GraphConnect will find a solution. By definition of LP_1, there exists a sequence Ω of disjoint $\mathcal{C}_{free}$ components starting in S and ending in G. We show that GraphConnect will add G to SFG by induction. In the base case, $S \in SFG$. Assume C_i is in SFG. By the definition of LP_1, a robot in C_i can independently move one obstacle O to connect an Ω-consecutive component C_j. Let e be the edge between C_i and C_j in DFG. Clearly, O faces both C_i and C_j, thus $ob \in OL(e)$. Since Manip-Search(C_i, C_j, O) is complete over the action space, it will find the connecting motion. Therefore, C_j will be added to SFG. By induction, G will be added to SFG.

Trivially, there are finite numbers of edges and nodes in DFG. Every iteration of GraphConnect either adds a node to SFG or removes an edge of DFG. Therefore, the algorithm must either add G to SFG or find that there are no remaining edges e that connect any $C_i \in SFG$ to a $C_j \notin SFG$. Hence GraphConnect terminates in finite time and is resolution complete for problems in LP_1. □

In addition to completeness, we can sketch a proof that GraphConnect with breadth-first-search is optimal for LP_1 problems: Since each edge of SFG corresponds to moving one obstacle, and breadth-first-search will construct an SFG with a minimal number of edges, our solution satisfies the first criterion of NAMO optimality. Furthermore, suppose we extend the search until all edges of solution depth are added. Then if more than one solution is found, Manip-Search can be used to score the $Work$ expended on each path and satisfy the second criterion.

6.3. *Improvement over GraphConnect*

The last optimality result would be most interesting if GraphConnect was feasible for implementation. Yet, although this algorithm dramatically reduces the search space from brute force action-space search, its primary purpose is to convey the utility of the reduced dimensional $\mathcal{C}$-space structure. Practically, a NAMO domain could have a large number K_o objects and N_f disjoint free-space components. Constructing the graph would require an algorithm to determine which objects connect which components of free-space. Furthermore, we would need to call MANIP-SEARCH to verify every potential connection. This seems unreasonable for a navigation planner, since out of K_o objects we may only have to move one or two to reach the goal.

We believe that this observation is critical for developing a real-time system. In particular, the complexity of the problem should depend on the complexity of resolving *keyholes* that lie on a reasonable navigation path, not on the complexity of unrelated components of the world. In other words, *since the purpose of NAMO is navigation, a NAMO problem should only be difficult when navigation is difficult.*

In Section 7, we present an initial answer to this challenge. We introduce a heuristic algorithm that implicitly follows the structure of GRAPHCONNECT without graph construction. To do so, we again turn to the navigational substructure of the problem. Previously, we observed that every NAMO plan is a sequence of actions that alter the accessible free-space $Acc(R_{COM})$. Every plan contains a *path* from the initial robot state to the goal. For plans in LP_1, this path contains no loops (i.e. after entering a component of free-space, it is never advantageous to return). Even LP_k and NLP problems may have large regions that are too far removed from the navigation path to be of any use. In our planner implementation we exploit the notion of *paths* to guide the planner towards reasonable obstacle selection.

7. Planner Prototype

With the tools and results from the previous sections, we now formulate a simple and effective planner for the NAMO domain. The planner follows a greedy, depth-first search to generate fast heuristic plans in LP_1 environments. The employed heuristic, $\mathcal{P}$, is itself a navigation planner with relaxed constraints. It is used to select obstacles that are to be considered for motion. Following the algorithm description, we show that its search space is equivalent to that of GRAPHCONNECT (Section 6.2). With depth-first search, optimality is no longer guaranteed. However, the planner is much more efficient and still resolution complete for problems in LP_1.

7.1. *Implemented Planner*

Sub-planner $\mathcal{P}(R_{COM}, S_W, AvoidList)$ is parametrized by robot location, the world state and a list of (O_i, C_i) pairs to avoid. After generating a path estimate to the goal, it returns the pair (O_1, C_1) of the first obstacle in the path, and the first disconnected free-space. If no such path exists, $\mathcal{P}$ returns NIL.

$\mathcal{P}$ is implemented as follows: a path to the goal is found by means of A$*$ on a dense regular grid.[*] The robot cannot enter cells occupied by fixed $\mathcal{C}$-space obstacles or transition from a movable obstacle to another movable obstacle. Finally, for any pair $(O_i, C_j) \in AvoidList$ the robot cannot transition consecutively from $Acc(R_{COM})$ to cells occupied by O_i and subsequently to cells occupied by C_j. The

[*]Appendix A contains an example of such a grid. Notice that $\mathcal{P}$ constructs paths *through* obstacles.

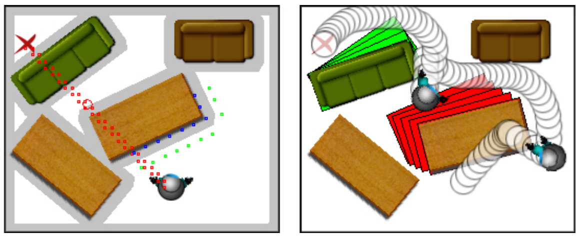

Fig. 4. Path generated by initial $\mathcal{P}$ and final HEURISTIC-PLAN. In this case, no backtracking was necessary. $\mathcal{P}$ selected the table over the love seat because it is defined to have lower mass.

heuristic cost of entering a grid cell **c** is defined by:

$$H_{\mathcal{P}}(c) = (1-\alpha)d_{goal} + \alpha(W). \tag{5}$$

Let W be a positive scalar value proportional to an estimate of the total work (effort) required to move an object that spans **c**. $W = 0$ when **c** is unoccupied. Currently, we base W on the occupying object's mass. The scaling factor α provides a weighting for the relative importance of moving objects. Clearly there may be more effective means for estimating W. However, we must be wary of their complexity since $\mathcal{P}$ is called often on a reconfigurable $\mathcal{C}$-space. The investigation of such heuristics is another interesting topic for future work.

Our implemented NAMO planner makes use of $\mathcal{P}$ and MANIP-SEARCH, as described in Section 6.2. It is a greedy heuristic search with backtracking. The planner backtracks locally when the object selected by $\mathcal{P}$ cannot be moved to connect the selected $C_i \subset \mathcal{C}_{free}$. It backtracks globally when all the paths identified by $\mathcal{P}$ for connecting C_i are unsuccessful. The following pseudo-code details the implementation:

```
HEURISTIC-PLAN($R_{COM}, S_W$)
 1  $AvoidList \leftarrow \emptyset$
 2  $PartialPlan \leftarrow \emptyset$
 3  while $(O_1, C_1) \leftarrow \mathcal{P}(R_{COM}, S_W, AvoidList) \neq$ NIL
 4  do
 5     if $C_1 =$ GOAL
 6        then return ($PartialPlan$ append GOAL)
 7     $(CP, Path, R'_{COM}, S'_W) \leftarrow$ MANIP-SEARCH($Acc(R_{COM}), C_1, O_1$)
 8     if $Path \neq$ NIL
 9        then $PartialPlan$ append $(CP, Path)$
10             $FuturePlan \leftarrow$ HEURISTIC-PLAN($R'_{COM}, S'_W$)
11             if $FuturePlan \neq$ NIL
12                then return ($PartialPlan$ append $FuturePlan$)
13        else $AvoidList$ append $(O_1, C_1)$
14  return NIL
```

For clarity, we have left out the generation of path plans from the current robot position to either the object contact point or the goal. *PartialPlan* is described

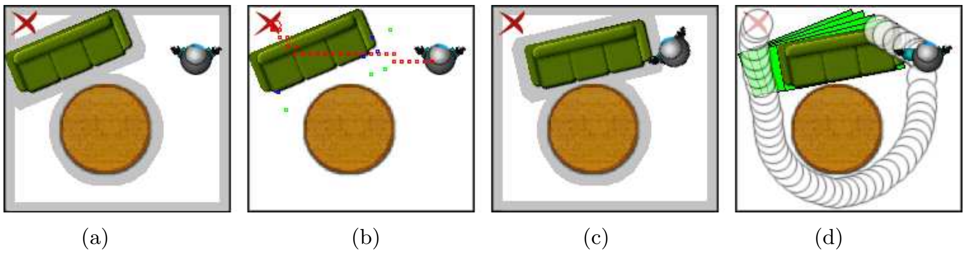

Fig. 5. Walk-through of a generated plan: (a)Problem (b)$\mathcal{P}$ (c)Keyhole solution (d)Final plan

as a list of pairs: $(ContactPoint, ManipulationPath)$. In a full implementation, $PartialPlan$ is a list of triples that includes the $NavigationPath$ to the contact point from R_{COM}.

7.2. *Examples and Experimental Results*

We have implemented the proposed planner in a dynamic simulation environment of NAMO. The intuitive nature of Heuristic-Plan is best illustrated by a sample problem solution generated by the planner. In Figure 5(a), we see that the $\mathcal{C}$-space of R_{COM} is disjoint - making this a NAMO problem. Line (3) of Heuristic-Plan calls $\mathcal{P}$ (the heuristic planner). $\mathcal{P}$ conducts an $A*$ search and finds that the least cost path to the goal lies through the couch. The path is shown in Figure 5(b). $\mathcal{P}$ also determines that the free-space component to be connected is the one containing the goal. Line (6) calls Manip-Search to find a motion for the couch. Figure 5(c) shows the minimum *Work* manipulation path that opens the goal free-space. Finally, a motion plan to the goal completes the procedure (Figure 5(d)). The remainder of the pseudo-code simply iterates this process when the connected free-space is not the goal and backtracks when a space fails to be connected.

Figure 5 is particularly interesting because it demonstrates our use of $\mathcal{C}_{free}$ connectivity. As opposed to the local planner approach employed by CH, Manip-Search does not directly attempt to connect two neighboring points in $\mathcal{C}$. Manip-Search searches all actions in the manipulation space to join the $\mathcal{C}$-space components occupied by the robot and current goal or subgoal. The procedure finds that it is easiest to pull the couch from one side and then go around the table for access. This human-like high-level decision cannot be reached by using existing navigation planners.

Figure 5 also demonstrates a weakness of LP_1 planning. Suppose the couch was further constrained by the table such that there was no way to move it. Although the table is obstructing the couch, the table does not explicitly disconnect any free-space and would therefore not be considered for motion.

Figures 6 and 7 illustrate more complex examples of NAMO planning. While computation time for Figure 5 is $< 1s$, the solutions for 6 and 7 were found in 6.5 and $9s$ respectively (on a Pentium 4 3GHz). Notice that the planning time depends primarily on the number of manipulation plans that need to be generated for a

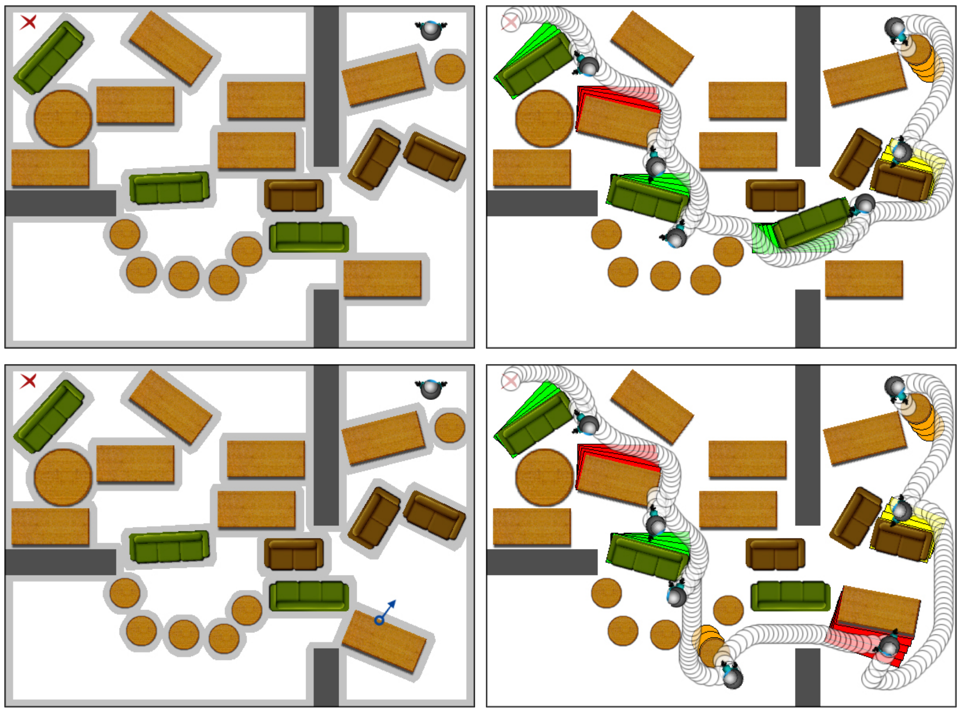
Fig. 6. The generated plan output by our dynamic simulation NAMO planner is illustrated by the time-lapse sequences on the right. In the lower frame, we changed the initial configuration of the table. The initial call to $\mathcal{P}$ still plans through the couch, however MANIP-SEARCH finds that it cannot be moved. The planner backtracks, calling $\mathcal{P}$ again and selects an alternative route.

solution. Although the largest example contains 90 movable obstacles, compared to twenty in Figure 6, there is no sizable increase in the solution time.

7.3. *Theoretical Considerations*

HEURISTIC-PLAN has clear advantages over GRAPHCONNECT in terms of both average computation time and ease of implementation. It is also apparent that HEURISTIC-PLAN is not globally optimal since the algorithm is greedy with a non-admissible (though well informed) heuristic. Note, however, that for each choice of obstacle, the planner still selects a motion requiring the least $Work$. The following Lemmas lead us to a proof of LP_1 completeness for HEURISTIC-PLAN:

Lemma 2. *Let O be a movable obstacle and C_1, C_2 be disjoint regions of $\mathcal{C}_{free}$. For C_1 considered by* HEURISTIC-PLAN *and for each pair (O,C_2): If there exists a path from C_1 to the goal which passes through C_1,O,C_2 consecutively then* HEURISTIC-PLAN *will call* MANIP-SEARCH(C_1, C_2, O).

Proof. Suppose HEURISTIC-PLAN is called with the robot in C_1. Then $\mathcal{P}$ is repeatedly called with the pair $(C_1, AvoidList)$. At each iteration, $\mathcal{P}$ selects some

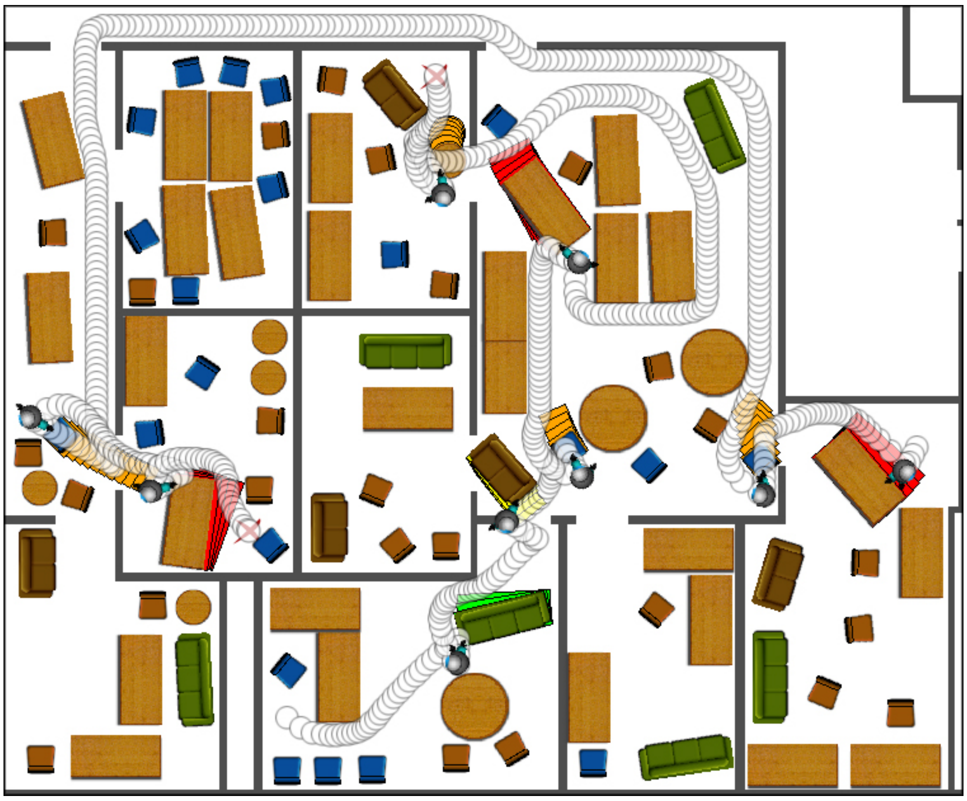

Fig. 7. A larger scale example consisting of 90 movable obstacles. Two separate plans are computed and demonstrated in our dynamic simulation.

reachable object O_i and C_j where $(O_i, C_j) \notin \mathit{AvoidList}$ such that the path considered passes through C_1,C_i,C_j consecutively (by definition of $\mathcal{P}$). This is followed by a call to Manip-Search(C_1, C_j, O_i). After each iteration (O_i, C_j) are added to AvoidList. Since there are finite obstacles and elements of $\mathcal{C}_{free}$, $\mathcal{P}$ will add every pair through which there is a path from C_1 to G. Thus if there exists a path that consecutively passes through C_1,O,C_2 then $\mathcal{P}$ will add $(O$,$C_2)$ and therefore Heuristic-Plan will call Manip-Search(C_1, C_2, O). $\square$

Lemma 3. *If* GraphConnect *finds a solution to a NAMO problem, then* Heuristic-Plan *will find a solution as well.*

Proof. Suppose GraphConnect has found a solution. Then there exists a sequence of connected nodes in SFG that connects S and G. We inductively show that Heuristic-Plan will necessarily find a path before terminating. In the base case, $S = G$ and $\mathcal{P}$ will immediately find a solution. Now let C_i and C_j be two disjoint free-space components associated with consecutive nodes in the SFG solution sequence. Assume Heuristic-Plan has found a plan to C_i from S. Since C_j

is part of SFG with an edge from C_i, there exists an obstacle O that faces both C_i, C_j, for which MANIP-SEARCH(C_i, C_j, O) *returns a valid plan.* We observe:

O faces C_i implies that there exists a path P_1 *from any point in* C_i *to* O that does not enter any other obstacle. O faces C_j implies that there exists a path P_2 *within* O *that enters* C_j. Trivially, there exists some path P_3 *from* C_j *to the goal.*

We conclude that there exists a combined path $Path_i = (P_1, P_2, P_3)$ from a robot in C_i to the goal. Furthermore, $Path_i$ passes through C_i,O,C_j consecutively. Therefore, when considering C_i, HEURISTIC-PLAN will evaluate MANIP-SEARCH(C_i, C_j, O) (Lemma 2). Since we have already established that with these parameters MANIP-SEARCH returns a valid plan, HEURISTIC-PLAN will find a plan from C_i to C_j and therefore from S to C_j (by the inductive hypothesis). Consequently, by induction HEURISTIC-PLAN will find a plan from S to G. □

Theorem 1. HEURISTIC-PLAN *is resolution complete for problems in* LP_1.

Proof. This result follows directly from Lemmas 1,3. HEURISTIC-PLAN always finds a solution whenever GRAPHCONNECT finds a solution and otherwise reports failure in finite time. Because GRAPHCONNECT is resolution complete, HEURISTIC-PLAN is resolution complete. □

8. Concluding Remarks

The problem of Navigation Among Movable Obstacles takes a step towards autonomous robot interaction with complex unstructured environments such as those involved in urban search and rescue applications. One direction of relevant research is the development of robust mechanisms and controls that allow for safe navigation and environment manipulation. In our work, we develop algorithmic solutions that can utilize the dexterity of such mechanisms by formulating real-time high-level manipulation strategies. In particular, such plans should allow robots to reconfigure their environments and solve difficult navigation tasks.

Although NAMO problems involving large numbers of obstacles are practically intractable for traditional AI planning algorithms, these are problems that humans solve regularly with relative ease. We observe that even *seemingly unstructured* environments contain significant structural components that humans take advantage of. The primary contribution of our work is the development of a planning methodology that utilizes an intuitive decomposition of the NAMO state-space, as well as the underlying structure of the navigation problem. The theoretical ideas introduced in this paper have been realized in the form of a heuristic motion planner that is resolution complete for a well defined subclass of NAMO problems. We experimentally demonstrated that such a planner can be implemented to efficiently solve relatively complex NAMO problems. In addition, the modular structure of the planning tools developed in this work yields an adaptable basis for future developments in robot/environment interaction.

In the class of LP_1 problems solved by our planner, many questions remain for future research. For implementing our algorithm on an actual robot, we would need to address the effects of partial observability in both the static structure and the dynamics of the environment. Additionally, though navigation is generally planar, three-dimensional contact placement and large object grasping should be considered. Furthermore, the problem of selecting the most practical action space for a robot that can affect its environment remains an open and fascinating topic for research.

In terms of planning, future work should address larger classes of NAMO problems. These include consideration for obstacles that do not directly disconnect the state-space but interfere with the motions to reconnect it. Three-dimensional effects due to object stacking and partial support should also be considered. It is possible that rearrangement planning methods that find intermediate obstacle configurations could be used to complement our algorithm after the search has been narrowed to smaller sets of relevant movable objects. We expect that rigorous study of these and other methods for state-space decomposition will lead to future progress in the capabilities of humanoid robots that includes a greater capacity for human-like reasoning in complex domains.

Acknowledgements

We thank C. G. Atkeson for his helpful advice and support. We are grateful to S. Siddiqi for many discussions and his contribution to the planning simulation. This research was partially supported by NSF grants DGE-0333420, ECS-0325383, ECS-0326095, and ANI-0224419.

Appendix A. Dynamic $\mathcal{C}$-space Modification

The two algorithms described in our work make judicious use of the MANIP-SEARCH(C_1, C_2, O) routine as defined in Section 6.2. MANIP-SEARCH is required to search the available robot action space in manipulating O(Section 5). The search should terminate when $\mathcal{C}_{free}$ contains a path for the robot to transition from C_1 to C_2. Conceptually this is a clear objective that can easily be extended to higher dimensional spaces. In terms of implementation, however, deciding whether the goal has been satisfied is an interesting challenge.

Suppose the robot applies action a to obstacle O. We need to quickly determine whether a path (C_1, C_2) exists. In terms of implementation and computational difficulty it seems daunting to formulate/maintain a complex model of all C_i. We avoid this by depicting the entire R_{COM} $\mathcal{C}$-space as a planar grid, where $\mathcal{C}$-space obstacles are represented by rasterizing their perimeter.[22] Since the robot embodies a cell in the $\mathcal{C}$-space, we can trivially test for connectivity between C_1 and C_2 by

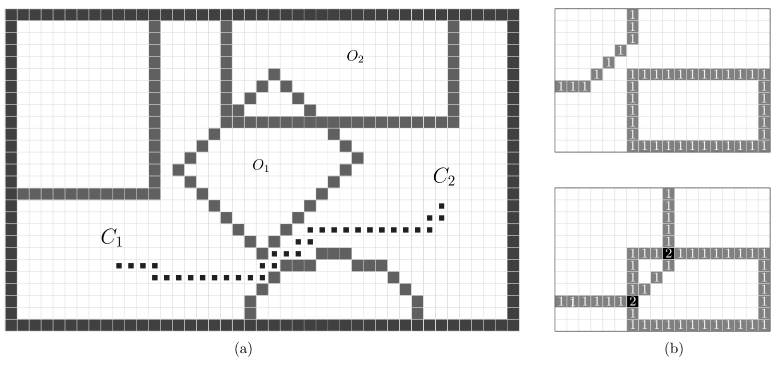

Fig. 8. (a) coarsely depicts a grid implementation of R_{COM} $\mathcal{C}$-space. A free path is illustrated confirming the connectivity of C_1,C_2. Notice that O_1 and O_2 have overlapping projections in $\mathcal{C}$. (b) shows an update of dynamic $\mathcal{C}$-space by obstacle reference counting.

locally searching for a path between any two grid cells $p_1 \in C_1$ and $p_2 \in C_2$.*

We are left with the problem of updating the $\mathcal{C}$-space in a way that would facilitate goal testing. Updating the discrete grid involves rasterizing the perimeter of the obstacle in its original configuration to *remove* it, and rasterizing it again in the new configuration. The difficulty lies in the fact that $\mathcal{C}$-space obstacles can overlap. *Removing* an obstacle requires us to clear the cells that belong to the obstacle. Due to overlap, however, a single grid cell may belong to more than one obstacle and should not be cleared when only one obstacle is removed. A simple binary grid provides no tools for determining if there are other objects occupying a given cell. A naive approach to this problem can be highly inefficient. Maintaining lists of obstacles for each grid cell can add another dimension to the $\mathcal{C}$-space. Alternatively, testing other obstacles for inclusion involves unnecessary search.

We solve this problem as follows: rather than maintaining a binary grid to represent the $\mathcal{C}$-space, we keep an integer grid. Each cell is represented by a counter of the number of obstacles to which it belongs. *Removing* an obstacle decrements the cell counters during rasterization, and *adding* the obstacle increments them. The robot can then reach all accessible cells of 0 count. This algorithm requires no added complexity in implementation and allows us to perform $\mathcal{C}$-space updates in time linear to the perimeter of the manipulated obstacle.

References

1. A. Wolf, H. Ben Brown Jr., R. Casciola, A. Costa, M. Schwerin, E. Shammas, and H. Choset. A mobile hyper redundant mechanism for search and rescue task. In *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pages 2889–2895, 2003.

*There exist more efficient methods for this test. We simply show that testing connectivity is not difficult in the discrete $\mathcal{C}$-space.

2. G. Wilfong. Motion panning in the presence of movable obstacles. In *Proc. ACM Symp. Computat. Geometry*, pages 279–288, 1988.
3. E. Demaine and et. al. Pushpush and push-1 are np-complete. Technical Report 064, Smith, 1999.
4. J. H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proc. 26th Annual Symposium on Foundations of Computer Science*, pages 144–154, 1985.
5. D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Workshop on the Algorithmic Foundations of Robotics*, 2000.
6. Oliver Brock and Oussama Khatib. Elastic strips: Real-time path modification for mobile manipulation. In *International Symposium of Robotics Research*, pages 117–122. Springer Verlag, 1997.
7. K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. Journal of Robotics Research*, 15(6):533–556, 1996.
8. M.T. Mason. *Mechanics of Robotic Manipulation.* MIT Press, 2001.
9. R. Alami, J.P. Laumond, and T. Sim'eon. Two manipulation planning algorithms. In *Workshop on the Algorithmic Foundations of Robotics*, 1994.
10. T. Simeon, J. Cortes, A. Sahbani, and J.P. Laumond. A manipulation planner for pick and place operations under continuous grasps and placements. In *Proc. IEEE Int. Conf. on Robotics and Automation,*, 2002.
11. N. Pollard and J.K. Hodgins. Generalizing demonstrated manipulation tasks. In *Workshop on the Algorithmic Foundations of Robotics*, 2002.
12. M. H. Goldwasser and R. Motwani. Complexity measures for assembly sequences. *Int. J. of Computational Geometry and Applications*, 9:371–418, 1999.
13. R. Wilson. *On Geometric Assembly Planning.* PhD thesis, Department of Computer Science, Stanford University, 1992.
14. O. Ben-Shahar and E. Rivlin. Practical pushing planning for rearrangement tasks. *IEEE Trans. on Robotics and Automation*, 14(4):549–565, 1998.
15. A. Junghanns and J. Schaffer. Sokoban: A challenging single-agent search problem. In *IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch*, pages 27–36, 1997.
16. J. Ota. Rearrangement of multiple movable objects. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1962–1967, 2004.
17. P.C.Chen and Y.K.Hwang. Pracitcal path planning among movable obstacles. In *Proc. IEEE Int. Conf. Robot. Automat.*, pages 444–449, 1991.
18. Andrew T. Miller. *GraspIt!: A Versatile Simulator for Robotic Grasping.* PhD thesis, Department of Computer Science, Columbia University, 2001.
19. R. Pelossof, A. Miller, P. Allen, and T. Jebara. An svm learning approach to robotic grasping. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 3215–3218, 2004.
20. D. G. Thelen. Adjustment of muscle mechanics model parameters to simulate dynamic contractions in older adults. *Journal of Biomechanical Engineering*, 2003.
21. T. Lozano-Perez. Spatial planning: a configuration space approach. *IEEE Trans. Comput.*, pages 108–120, 1983.
22. J. Lengyel, M. Reichert, B.R. Donald, and D.P. Greenberg. Real-time robot motion planning using rasterizing computer. *Computer Graphics, ACM*, 24(4):327–335, 90.