Path Planning among Movable Obstacles: A Probabilistically Complete Approach

Jur van den Berg, Mike Stilman, James Kuffner, Ming Lin, and Dinesh Manocha

Abstract. In this paper we study the problem of path planning among movable obstacles, in which a robot is allowed to move the obstacles if they block the robot's way from a start to a goal position. We make the observation that we can decouple the computations of the robot motions and the obstacle movements, and present a *probabilistically complete* algorithm, something which to date has not been achieved for this problem. Our algorithm maintains an explicit representation of the robot's configuration space. We present an efficient implementation for the case of planar, axis-aligned environments and report experimental results on challenging scenarios.

1 Introduction

In this paper we consider the problem of path planning among movable obstacles. This involves an environment with static and *movable* obstacles, and the task is for a robot to plan a path from some start position s to some goal position g, whereby the robot can move the movable obstacles out of its way. The robot and the movable obstacles may not collide with other obstacles.

This problem is more complex than typical robot path planning among only static obstacles. The computational challenge is similar to games like Sokoban [6] where it is easy to design puzzle scenarios that are difficult even for experienced human players. Not surprisingly, the problem is known to be NP-hard [19].

Jur van den Berg, Ming Lin, and Dinesh Manocha
Department of Computer Science, University of North Carolina at Chapel Hill
e-mail: {berg,lin,dm}@cs.unc.edu

Mike Stilman
School of Interactive Computing, Georgia Tech
e-mail: mstilman@cc.gatech.edu

James Kuffner
School of Computer Science, Carnegie Mellon University
e-mail: kuffner@cs.cmu.edu

Due to the complexity of our problem, previous works have focused on heuristics [5, 10, 15], and have given completeness results only for subclasses of the problem [1, 13, 14, 19]. No complete algorithms are known that cover the entire problem domain. In contrast, for other path planning problems generally applicable algorithms have been proposed that rely on *probabilistic completeness*— a weaker form of completeness that given infinite time guarantees to find a solution to any problem for which a solution exists [8, 9].

In this paper, we present a *probabilistically complete* algorithm that covers the entire domain of planning among movable obstacles. Our approach is based on the observation that we can *decouple* the computation of the robot's motion from the computation of the obstacle movements, if we maintain an explicit representation of the robot's free configuration space and keep track of which connected component the robot configuration resides in. We present an efficient data structure to maintain a representation of the robot's configuration space for the specific case in which both the robot and obstacle geometry can be represented by translating axis-aligned rectangles. We implemented our algorithm and data structure and present experimental results on challenging problem scenarios.

The rest of this paper is organized as follows. In the next section, we give an overview of previous work. In Section 3, we formally define our path planning problem. We present our approach and prove its probabilistic completeness in Section 4. In Section 5 we describe the implementation of our algorithm and a data structure to maintain the robot's configuration space, and discuss results in Section 6. We conclude the paper in Section 7.

2 Related Work

Achieving completeness in planning among movable obstacles has proven extremely challenging. The problem was shown to be NP-hard by Wilfong [19] and addressed with heuristic methods by Chen and Hwang [5]. Stilman and Kuffner [14] introduced (resolution-) completeness to this domain by showing that a subclass of problems called L_1 could be solved within a practical amount of time. The class was broadened to monotone problems in [15]. These approaches are particularly relevant to practical scenarios where an efficient method is required to identify blocking obstacles and restore connectivity in the robot's free space. However, they cannot solve movable obstacle problems outside the given subclasses.

Constructing an efficient, generally complete algorithm is difficult even for the standard path planning problem of a single robot moving among static obstacles [3]. Recently, probabilistic completeness has become an alternative standard for path planning problems. Sampling-based planners such as PRM [8] and RRT [9] have proven to be very successful in a domains ranging from single and multiple robots [12, 17] to dynamic environments with non-holonomic constraints [7]. This success prompted Nieuwenhuisen et. al. [10] and Stilman et. al. [16] to apply sampling based planning in the movable obstacle domain. However, in both cases the expansion of search trees for individual obstacle movements was bounded to ensure proper backtracking over alternative obstacle choices. In order to ensure probabilistic completeness an algorithm must explore all possibilities and allow these search trees to grow indefinitely.

We now propose an algorithm that allows indefinite exploration over all obstacle movements. The algorithm is proven to be probabilistically complete. Not only is this result new for the domain of planning among movable obstacles, but also for related domains such as rearrangement planning [2, 4, 11] or manipulation planning [1, 13] where the goal is specified in terms of goal configurations for the obstacles, rather than for the robot. While we do not directly address this variant of the problem, the observations in this paper can be applied to the design of probabilistically complete algorithms that span these domains as well.

3 Problem Definition

The problem we discuss in this paper is defined as follows. We are given a robot R and a two- (or three-) dimensional workspace containing a set of static obstacles O and a set of n (rigid) movable obstacles $\{M_1,\dots,M_n\}$. We denote the configuration space of R, i.e. the set of all possible configurations of the robot, by C_R (e.g. if R is a "free-flying" robot in the plane, then $C_R = \mathbb{R}^2 \times [0,2\pi)$), and we similarly denote the configuration space of each of the movable obstacles M_i by C_{M_i}. A movable obstacle cannot move by itself, but can be moved by R if R first *grasps* the obstacle.

Given a start configuration $s \in C_R$ and a goal configuration $g \in C_R$ for the robot, and initial configurations $(c_1,\dots,c_n) \in C_{M_1} \times \cdots \times C_{M_n}$ for the movable obstacles, the task is to find a collision-free path for the robot R from s to g. The robot is allowed to move the movable obstacles, but only one at a time, and only if the robot is grasping the obstacle. During the movement of an obstacle, both the robot and the obstacle should be collision-free with respect to other obstacles. We generally

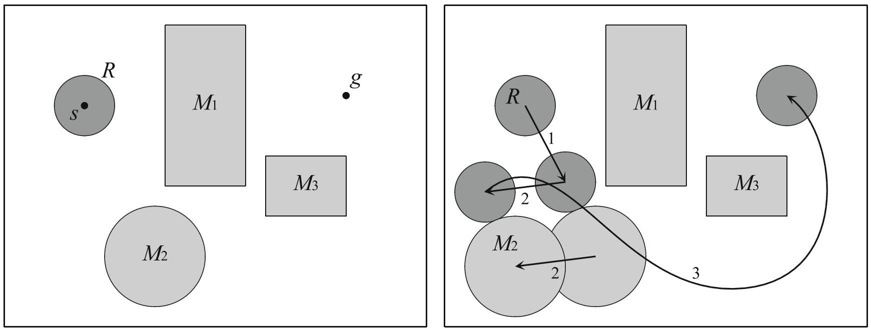

Fig. 1. (a) The initial situation of an example problem with three moving obstacles M_1, M_2 and M_3. The dark grey disc R is the robot. **(b)** An alternating sequence of navigation actions (1 and 3) and manipulation actions (2) that solves the problem.

define that the robot can grasp (and move) a movable obstacle when it is *touching* that obstacle.

The problem is thus defined as finding a *sequence of actions*, alternating between *navigation actions*, in which a robot moves by itself from a configuration on the boundary of one moving obstacle to a configuration on the boundary of another moving obstacle, and *manipulation actions*, in which the robot rigidly attaches itself to a movable obstacle and moves as a composite body from one position to the other. The first and last actions of the sequence are navigation actions that begin in the robot's start configuration s, and end in the robot's goal configuration g, respectively. The navigation actions in the sequence may include *regrasps* of the same obstacle, in which the robot moves to another configuration on the boundary of the moving obstacle to grasp it there. In Fig. 1 we show a sequence of actions that solve an example problem.

4 Approach

The problem has traditionally been approached by finding an alternating sequence of navigation actions and manipulation actions [1, 10, 13, 14]. This formulation, however, makes it difficult to devise a (probabilistically) complete planner. This is because each of the navigation and manipulation actions lie in a sub-dimensional "slice" of the composite configuration space $C_R \times C_{M_1} \times \cdots \times C_{M_n}$ of the robot and the obstacles. There is an infinite number of such slices, and each of these slices have zero probability to receive a sample in a probabilistic planner. Previous works have circumvented this problem by constraining the problem to a finite set of possible obstacle positions and grasps [1], or by dealing with only one movable obstacle [13].

In this paper, we discuss the general continuous problem with any number of movable obstacles. The key to our approach is that the problem should not be defined in terms of finding an alternating sequence of navigation and manipulation actions, but that one should abstract from the precise motions of the robot, and focus on the movements of the obstacles.

In our approach we are looking for a sequence of obstacle movements. The precise robot motions that lead to such obstacle movements are not explicitly computed; *we only make sure that the robot is somehow able to validly execute those movements*. In order to test whether movements of the movable obstacles are executable by the robot, we maintain an explicit representation of the *free configuration space* of the robot. Each of the static and movable obstacles induce a *C-obstacle* in the robot's configuration space, consisting of robot configurations in which the robot is in collision with that obstacle. The free configuration space is the space of configurations in which the robot is collision-free. This free configuration space consists of multiple *connected components*, whose boundaries consists of boundaries of C-obstacles (see Fig. 2a). If the robot is on the boundary of the C-obstacle of one of the movable obstacles, it is touching that obstacle. As defined above, it is then able to move that movable obstacle. So, if the robot is in a free connected component N, it is able to move the movable obstacles whose C-obstacles are adjacent to N. When

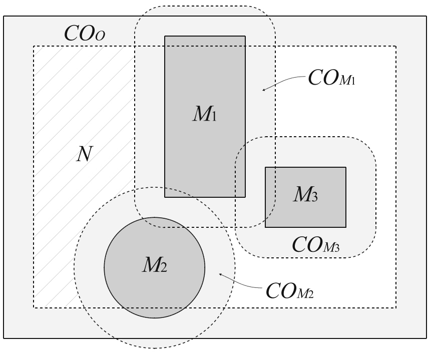

Fig. 2. The configuration space of the robot in the situation of Fig. 1a. The C-obstacles are shown light gray. There are two connected components in the robot's free configuration space. The dashed region labeled N is the one the robot is in.

an obstacle is moved, the configuration space of the robot and the connected component N change their shape, but as long as the C-obstacle of the movable obstacle remains adjacent to N, the robot is able to execute that movement. So, it is necessary only to keep track of which connected component the robot is in, rather than the exact configuration of the robot. This observation is central to our approach.

The task is now to find a sequence of obstacle movements that results in a situation in which the robot's goal configuration g is in the same connected component N as the robot. Once we have found such a sequence, we can (relatively easily) find the actual motions of the robot that execute these movements as a post-processing step.

In the remainder of this section, we will formalize the above observation and introduce the state space of the problem (Section 4.1). We then present a simple random-search algorithm (Section 4.2), and show that this algorithm is probabilistically complete (Section 4.3). Note that we do not make any assumption about the nature and dimensionality of the configuration spaces of both the robot and the movable obstacles.

4.1 State Space

Let us denote the robot R configured at $c_R \in C_R$ by $R(c_R)$, and similarly a movable obstacle M_i configured at $c_i \in C_{M_i}$ by $M_i(c_i)$.

Each of the movable obstacles generates a C-obstacle in the configuration space C_R of the robot (see Fig. 2). Given a specific configuration $c_i \in C_{M_i}$ of a movable obstacle M_i, its C-obstacle is given by $CO_{M_i}(c_i) = \{c_R \in C_R \,|\, M_i(c_i) \cap R(c_R) \neq \emptyset\}$. Similarly, the static obstacles generate a C-obstacle CO_O in C_R. Now, given specific configurations $(c_1, \ldots, c_n) \in C_{M_1} \times \cdots \times C_{M_n}$ of all movable obstacles, the *free configuration space* of the robot, i.e. the set of all configurations of the robot for which it is collision-free, is given by $C_R^{\text{free}}(c_1, \ldots, c_n) = C_R \setminus (CO_O \cup \bigcup_i CO_{M_i}(c_i))$. Note that the shape of the free space of the robot changes when an obstacle is moved.

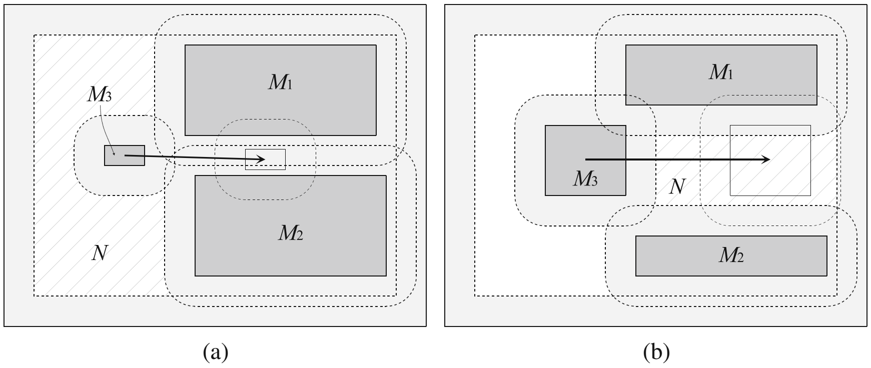

Fig. 3. Invalid obstacle movements. **(a)** The movement of obstacle M_3 is not executable by the robot, because at some point M_3's C-obstacle will not be adjacent to the connected component N anymore. **(b)** The movement of obstacle M_3 causes the connected component N of the robot to disappear, so there does not exist a collision-free motion for the robot to execute this movement.

At any time, the robot's free configuration space consists of one or more *connected components* (see Fig. 2), and the robot must be residing in one of them. We denote the connected component in which the robot resides by N.

In configurations on the *boundary* of N, the robot is *touching* some static or movable obstacle. If it is touching a movable obstacle M_i, the robot's configuration is on the boundary of the C-obstacle of M_i as well, and in that case the robot is able to move M_i. This leads to the following observation.

Definition 4.1 (Manipulable obstacle). *Given the configurations $(c_1,\ldots,c_n)$ of the movable obstacles and the connected component N of the robot's free configuration space that contains the robot, we define a movable obstacle M_i to be* manipulable *if its C-obstacle is adjacent to N, i.e. $\partial CO_{M_i} \cap \partial N \neq \emptyset$, where ∂ refers to the boundary of a set.*

Lemma 4.1. *Given initial configurations $(c_1,\ldots,c_n)$ of the movable obstacles and the connected component N of the robot's free configuration space that contains the robot, the movement of a movable obstacle M_i over a path $\pi : [0,1] \to C_{M_i}$, with $\pi(0) = c_i$, is valid and can be executed by the robot if (see Fig. 3 for examples of* in*valid obstacle movements):*

- *The movable obstacle M_i is* collision-free *with respect to the other obstacles at all times during the movement, i.e. $(\forall t \in [0,1] :: M_i(\pi(t)) \cap O = \emptyset \wedge (\forall j \neq i :: M_i(\pi(t)) \cap M_j(c_j) = \emptyset))$.*
- *The movable obstacle M_i is* manipulable *at all times during the movement, i.e. $(\forall t \in [0,1] :: \partial CO_{M_i}(\pi(t)) \cap \partial N \neq \emptyset)$. (Note that the shape of N changes during the movement of M_i over π.)*

Proof. Let the robot initially be in some configuration $c_R \in N$. As N shares part of its boundary with the boundary of CO_{M_i}, there exists some collision-free path within

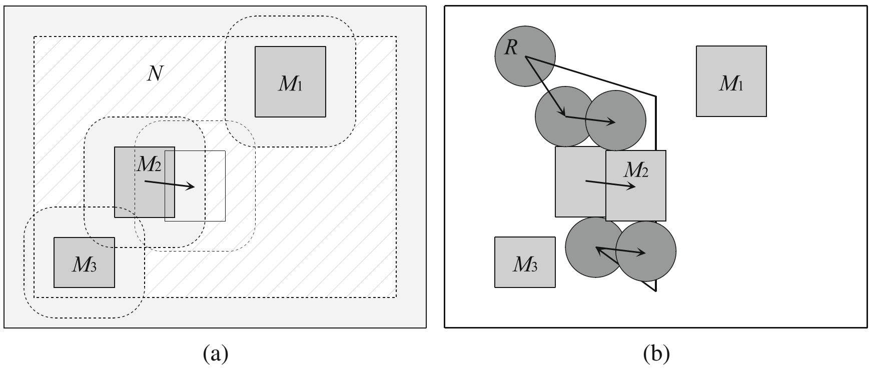

Fig. 4. (a) A situation in which the movement of obstacle M_2 splits the connected component N. **(b)** Two motions of the robot executing the same movement of M_2, but ending up in either of the two connected components formed by the split.

N for the robot to arrive at a point p on the boundary of CO_{M_i}. Let us rigidly attach this point p to CO_{M_i}. Now, the robot is able to move M_i along π. As M_i moves the point p moves, so p might leave the boundary of N at some moment. At the instant that this happens, the robot must regrasp, and find a new point p' that is both on the boundary of N and on the boundary of CO_{M_i}. As there is a part of the boundary of CO_{M_i} that is also on the boundary of N (see the second requirement), such a point p' must exist. As both p and p' are in N at that moment, there exists a free path for the robot that arrives at p'. Now the robot can continue moving M_i, and the above process can repeat until M_i arrives at configuration $\pi(1)$. □

As mentioned, the connected component N of the robot changes its shape during the movement of an obstacle. In some cases the obstacle movement may lead to a *split* of the connected component N into *two* new connected components (see Fig. 4a). In such a case, the robot may be in either of the two newly formed connected components after the split (see Fig. 4b), so we have to choose which component the robot will next be in.

Based on Lemma 4.1, we can define the *state space* our problem "lives" in. A *state* x is defined as a tuple $\langle c_1, \ldots, c_n, N \rangle$, where $c_1, \ldots, c_n$ are the configurations of the movable obstacles, and N is the connected component of the robot's free configuration space in which the robot resides (note that the definition of a state does not include any information regarding the specific configuration of the robot). The *state space* X is consequently defined as the set of all states. Given the robot's goal configuration g, the *goal region* $X_{\text{goal}} \subset X$ is given as the set of all states $x \in X$ for which $g \in N$. The initial state $x_{\text{init}} \in X$ is given by the initial configurations of the movable obstacles, and the connected component containing the robot's start configuration s.

We define an *action* u as a tuple $\langle M_i, \pi, \chi \rangle$, in which movable obstacle M_i is moved over path $\pi : [0,1] \rightarrow C_{M_i}$, and choices as given in χ are made with respect to the robot's connected component in cases of component splits encountered during

Algorithm 1. RANDOMTREE$(x_{\text{init}}, X_{\text{goal}})$

```
1: $T \leftarrow \{x_{\text{init}}\}$.
2: while true do
3:     Pick a random state $x \in T$ from the tree.
4:     $x' \leftarrow$ EXPAND$(x)$.
5:     $T \leftarrow T \cup \{x'\}$.
6:     if $x' \in X_{\text{goal}}$ then
7:         Path found! Terminate.
8:     end if
9: end while
```

Algorithm 2. EXPAND$(x) : X$

```
1: Pick a random movable obstacle $M_i$ that is manipulable in $x$.
2: Pick a random configuration $c_i'$ from $M_i$'s configuration space $C_{M_i}$.
3: return MOVEOBSTACLE$(x, M_i, c_i')$.
```

Algorithm 3. MOVEOBSTACLE$(x = \langle c_1, \ldots, c_n, N \rangle, M_i, c_i') : X$

```
1: while $M_i$ is manipulable and $M_i$ is collision-free and $M_i$ is not at $c_i'$ do
2:     Move $M_i$ toward $c_i'$, and keep track of the robot's connected component $N$.
3:     if $N$ splits into two components during the movement of $M_i$ then
4:         $N \leftarrow$ a component randomly chosen among the two formed by the split.
5:     end if
6: end while
7: return the resulting state $x'$.
```

M_i's movement over π. An action is valid if the movement of M_i over π is valid according the requirements of Lemma 4.1. Applying an action u to a state $x \in X$ results in a new state $x' \in X$. Below, we present a simple algorithm that finds a sequence of valid actions that when applied to the initial state x_{init} gives a final state in X_{goal}.

4.2 Algorithm

Our algorithm randomly builds a tree of states that are connected by actions. The tree is rooted in the initial state x_{init}. In each iteration, we randomly pick a state from the tree, and *expand* it by applying a randomly chosen action to that state. The newly created state is then added to the tree. This repeats until a state has been reached that is in X_{goal} (see Algorithm 1). In the algorithm, we continuously keep track of an explicit representation of the robot's configuration space.

We only consider actions that move an obstacle along a *straight line* in the obstacle's configuration space (see Algorithm 2). The obstacle is moved toward a randomly chosen configuration as long as the movement is valid, or until the picked configuration is reached (see Algorithm 3). We next prove that our algorithm is *probabilistically complete*.

4.3 Probabilistic Completeness

In the following, a problem solution is defined as a sequence of k *straight-line* actions $u_1, \ldots, u_k$, where $u_j = \langle M_{i_j}, \pi_j, \chi_j \rangle$ and each π_j is a straight-line path, that transform the initial state into a state in X_{goal}. Notice that any sequence can be approximated with one consisting of straight-line paths. A solution $u_1, \ldots, u_k$ has *clearance* ε if any alternative sequence $u'_1, \ldots, u'_k$ —where $u'_j = \langle M_{i_j}, \pi'_j, \chi'_j \rangle$ such that the endpoint of each π'_j deviates no more than $\frac{\varepsilon}{k}$ from the endpoint of π_j (i.e. $\|\pi'_j(1) - \pi_j(1)\| < \frac{\varepsilon}{k}$ for all $j \in 1..k$) *and* the correct choices χ'_j are made in case of component splits— is also a solution to the problem. Applying all these alternative sequences $u'_1, \ldots, u'_k$ to the initial state x_{init} gives a sequence $X_0 = \{x_{\text{init}}\}, X_1, \ldots, X_k$ of sets of states, such that $X_k \subset X_{\text{goal}}$. The following establishes probabilistic completeness for the random tree planner.

Theorem 4.1. *If there exists a solution with clearance $\varepsilon > 0$ then the the probability that* RANDOMTREE *will find a solution approaches 1 as the number of states in the tree approaches ∞.*

Proof. Assume that the random tree contains state $x_{j-1} \in X_{j-1}$ after some finite number $z-1$ of iterations. In the next iteration, each state in the tree has a probability $1/z$ to be selected for expansion (see line 3 of Algorithm 1). If x_{j-1} is chosen as the state to expand, there exists a second probability greater than some $q > 0$ that an action $\langle M_{i_j}, \pi'_j, \chi'_j \rangle$ with $\|\pi'_j(1) - \pi_j(1)\| < \frac{\varepsilon}{k}$ is chosen that results in a state $x_j \in X_j$ (see lines 1-2 of Algorithm 2; c'_i needs to be picked such that $\|c'_i - \pi_j(1)\| < \frac{\varepsilon}{k}$). Hence, the probability of 'success', i.e. that the next step in the solution sequence is constructed, in the z'th iteration is q/z.

Now, let random variable Y_z denote the number of successes we have had after z iterations. The expected value and the variance of Y_z are given by:

$$E(Y_z) = \sum_{i=1}^{z} \frac{q}{i} = q(\psi_0(z+1) + \gamma) \tag{1}$$

$$\mathrm{Var}(Y_z) = \sum_{i=1}^{z} \left[\frac{q}{i}(1 - \frac{q}{i})^2 + (1 - \frac{q}{i})(\frac{q}{i})^2 \right] = E(Y_z) + q^2(\psi_1(z+1) - \tfrac{\pi^2}{6}) \tag{2}$$

where $\psi_n(x)$ is the n'th polygamma function, and γ the Euler-Mascheroni constant (the closed form for $\mathrm{Var}(Y_z)$ was obtained using Maple).

To construct a solution sequence to a state $x_k \in X_k \subset X_{\text{goal}}$, we need k times success. The expected number of successes $E(Y_z)$ approaches infinity as the number of iterations z approaches infinity (i.e. $\lim_{z\to\infty} E(Y_z) = \infty$), so $E(Y_z) - k$ is positive for sufficiently large z. In these cases, the probability $\Pr(Y_z < k)$ that after z iterations a solution sequence has *not* been found is upper bounded by the Chebyshev inequality:

$$\Pr(Y_z < k) \;<\; \Pr(|E(Y_z) - Y_z| > E(Y_z) - k) \;\leq\; \frac{\mathrm{Var}(Y_z)}{(E(Y_z) - k)^2} \tag{3}$$

$\Pr(Y_z < k)$ approaches zero as the number of iterations z approaches infinity, as

$$\lim_{z\to\infty} \Pr(Y_z < k) \;\leq\; \lim_{z\to\infty} \frac{\mathrm{Var}(Y_z)}{(E(Y_z) - k)^2} \;=\; 0 \tag{4}$$

Hence, the probability $1 - \Pr(Y_z < k)$ that a solution *has* been found approaches 1 as the number of states in the tree approaches infinity. □

5 Implementation

The challenging part of the above algorithm is to explicitly maintain the robot's configuration space, and detect events (such as connected component splits) critical for the algorithm. Below, we present a data structure for the specific case of all obstacles and the robot being *axis-aligned rectangles* that can *translate* in the *plane*. This data structure enables us to efficiently perform the checks of lines 1 and 3 of Algorithm 3, in an *exact* and *continuous* manner, so we do not have to rely on approximations taking small discrete steps.

5.1 Data Structure

We maintain two data structures: the *workspace*, in which we make sure that movable obstacles will not collide with other movable or static obstacles, and the *configuration space* of the robot, in which we keep track of the connected component the robot is in.

As both the robot and the obstacles are axis-aligned rectangles that translate, the C-obstacles the movable obstacles induce are also axis aligned rectangles. Now, we look at all vertical and horizontal lines which are incident to the boundaries of the workspace obstacles and the C-obstacles, and use these lines to form which we call a *rectangular map* for the workspace and the configuration space of the robot, respectively (see Fig. 5).

For both the workspace and the configuration space of the robot, we store two lists of lines: one list for horizontal lines and one list for vertical lines. Both of these lists are *ordered*, the vertical lines from left to right and the horizontal lines from bottom to top. With the lines, we store their coordinate, to which obstacle it belongs, and whether it is the bottom or top line (horizontal lines), or the left or right line (vertical lines) of the particular obstacle.

For the configuration space of the robot, we maintain some additional information in the data structure. The rows and columns in between the lines are overlapping zero or more obstacles. If, for a horizontal row, the bottom line of an obstacle is below the row, and the top line of the obstacle is above the row, the row contains the particular obstacle. We store these associations with the rows and columns. In Fig. 5b, we encoded these associations using three bits, one bit for each obstacle. If first

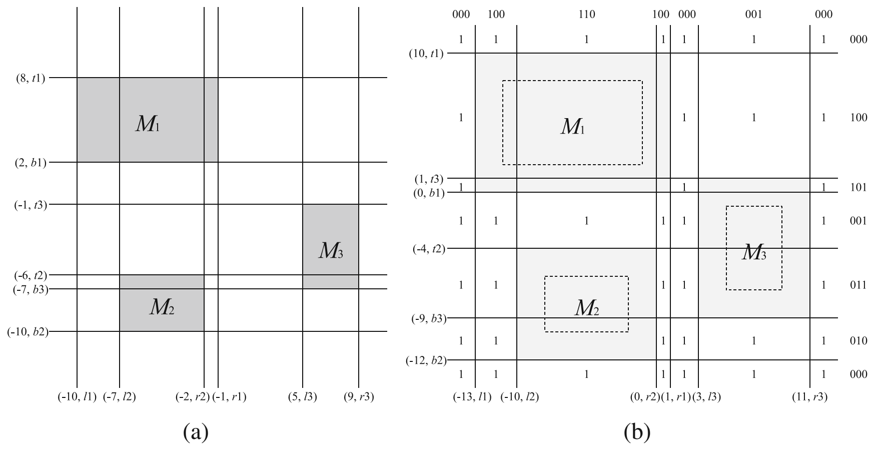

Fig. 5. **(a)** The rectangular map of the workspace. The coordinate and the type (l_i, r_i, b_i and t_i mean the left, right. bottom and top line of obstacle M_i, respectively) are stored with the horizontal and vertical lines. **(b)** The rectangular map of the configuration space of the robot. Here, we additionally store with the rows and columns what obstacles they overlap (see right and top), and for each empty cell in the map whether or not it belongs to the robot's connected component.

bit is 1, then the row (or column) contains the obstacle M_1. If the second bit is 1, it also contains obstacle M_2, etc. Now, to determine the status of a cell in the rectangular map, we can simply "and" the bits stored at the row and the column of that particular cell. If the result is 0, the cell is free, otherwise, the bits determine by which obstacles the cell is occupied.

In addition, we maintain in the configuration space data structure in which connected component the robot is. Given the initial position of the robot, we can quickly find in which cell of the (configuration space) map it is. Then, we "flood fill" the empty cells of the rectangular map from the cell containing the robot's position, and store with each free cell a flag indicating whether or not the cell belongs to the robot's connected component. In the example of Fig. 5b, all free cells belong to the robot's connected component, as there is only one connected component.

5.2 Events

Given the initial configuration space and workspace as constructed as explained above, we can manipulate the environment by moving the movable obstacles. While doing so, we need to keep track of the changes made in the configuration space data structure and the workspace data structure. We can only move an obstacle if it is manipulable. This is when in the rectangular map of the robot's configuration space, a cell in the connected component of the robot is adjacent to a cell occupied by the C-obstacle of the movable obstacle. Suppose we have selected a movable obstacle to be moved along some straight line. While moving the obstacle, the coordinates of the horizontal lines and the vertical lines associated with the obstacle in both the configuration space and the workspace change. As long as the ordering of the vertical and horizontal lines remains the same, we only need to update the coordinates stored at the vertical and horizontal lines. However, at the moment two vertical lines or two horizontal lines swap position in either the workspace or the configuration space, we need to (1) check if that swap is allowed (does the obstacle lose its manipulability? does the obstacle collide with another obstacle?), and if so (2) update the the data structure.

Given a proposed straight-line movement of an obstacle M_i, we can compute using the coordinates of the lines what the first *event* is that will be encountered, i.e. the first occasion in which we potentially swap two lines. Below we iterate over all possible events the can be encountered, and show for each of them how we check whether the event is allowed. We only discuss events that occur when M_i moves directly to the *right* (all other directions can be handled symmetrically). So, each event involves the left or the right line of obstacle M_i, and the left or the right line of another obstacle, say M_j, in either the workspace or the configuration space (C-space) of the robot.

- *Workspace*: *left* line of M_i, *left* line of M_j. The event is always allowed.
- *Workspace*: *left* line of M_i, *right* line of M_j. The event is always allowed.
- *Workspace*: *right* line of M_i, *left* line of M_j. The event potentially causes M_i to start colliding with M_j. The event is allowed if the bottom line of M_i is above the top line of M_j, or if the top line of M_i is below the bottom line of M_j.
- *Workspace*: *right* line of M_i, *right* line of M_j. The event is always allowed.
- *C-space*: *left* line of M_i, *left* line of M_j. The event potentially causes M_i to stop being manipulable (similar to the situation of Fig. 3a). The event is *not* allowed if all free cells of the connected component of the robot that are adjacent to the C-obstacle of M_i are below the top line of M_j, above the bottom line of M_j, and left of the left line of M_j.
- *C-space*: *left* line of M_i, *right* line of M_j. The event is always allowed.
- *C-space*: *right* line of M_i, *left* line of M_j. The event potentially causes the connected component of of the robot to disappear (similar to the situation of Fig. 3b). The event is *not* allowed if all free cells of the connected component of the robot that are adjacent to the C-obstacle of M_i are below the top line of M_j, above the bottom line of M_j, and right of the right line of M_i. The event may also cause a connected component *split* (similar to the situation of Fig. 4a); we will discuss this below.
- *C-space*: *right* line of M_i, *right* line of M_j. The event is always allowed.

If the event we encountered is allowed, we need to update the data structure. If the event happened in the workspace, we only need to swap the involved lines in the ordered list of lines stored in the data structure, and update the coordinate of the lines of the moved obstacle M_i.

If the event happened in the configuration space of the robot, we first swap the involved lines in the configuration space data structure, and update the coordinate of the lines of M_i. Second we need to update the information stored with the column

Algorithm 4. EXPAND$(x = \langle c_1, \ldots, c_n, N \rangle) : X$

```
1: for $K$ times do
2:     Pick a random movable obstacle $M_i$ that is manipulable in $x$.
3:     Pick a random direction $\theta \in \{0, \pi/2, \pi, 3\pi/2\}$, and a random distance $r$.
4:     $x \leftarrow$ MOVEOBSTACLE$(x, M_i, c_i + (r\cos\theta, r\sin\theta))$.
5: end for
6: return $x$.
```

(or row) that is in between the two lines that have swapped. Given the bit-representation, we simply need to "x-or" the stored value with the bits of both of the involved obstacles M_i and M_j.

Further, we need to update the component information of the free cells; i.e. set the flag whether or not they belong to the connected component of the robot. Only the cells in the column between the two swapped lines may have changed status from "occupied by a C-obstacle" to "free" or vice versa. Initially, we set the flag of all cells in that column to 0 (i.e. not belonging to the robot's connected component). Then we use a flood fill from free cells flagged 1 that neighbor the column to set all the flags of the free cells belonging to the robot's connected component. However, if the event involves the right line of M_i, and the left line of M_j, the connected component of the robot may have split into two components (which would both be flagged 1 after the above flood-fill). Whether this is the case can be checked using another flood fill. If the connected component has indeed split, the robot can in the new situation be in either of the newly created components. In our algorithm, we randomly pick one, and set the flags of the cells in the other component to 0.

After the event has been handled, we can compute what the next event is that is encountered when moving M_i. This repeats until M_i has reached its destination, or until an event is encountered that is not allowed.

5.3 Algorithm

We implemented the algorithm of Section 4.2 using the data structure presented above. For each state in the tree, we store both the rectangular map of the workspace and the rectangular map of the configuration space of the robot. As this is quite memory-intensive —the space complexity of the data structure is $O(n^2)$— we slightly changed Algorithm 2 (see Algorithm 4). Instead of storing the state after each obstacle movement, we let the state be expanded by a random sequence of K obstacle movements, where K is a parameter of the algorithm. This does not affect the probabilistic completeness: Theorem 4.1 also holds for a sequence of sequences of actions. Further, also without loss of probabilistic completeness, we let the obstacles only move along axis-aligned paths (see line 3 of Algorithm 4).

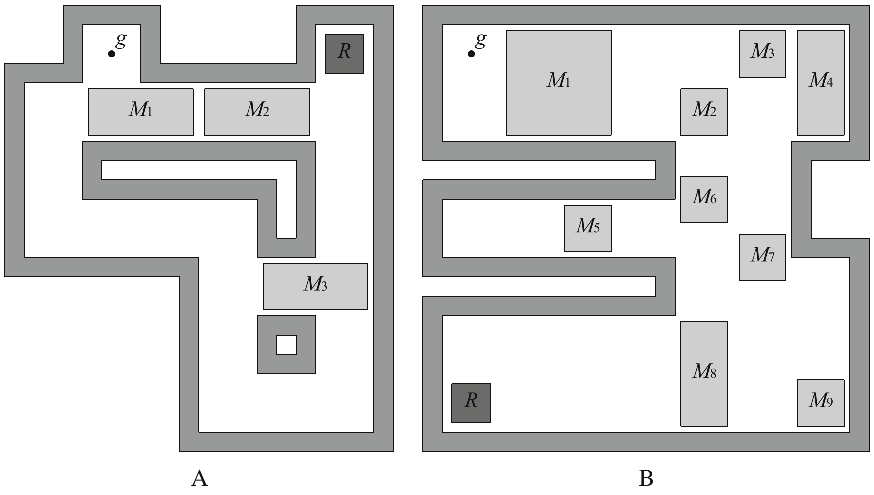

Fig. 6. The initial situation of two scenarios A and B we perform experiments on. The movable obstacles are light gray, the static obstacles are dark gray, and the robot is even darker gray. The robot is shown in its start configuration.

6 Results

As a proof-of-concept, we report results of experiments performed on two scenarios (see Fig. 6). Scenario A only contains three movable obstacles, but is a difficult problem mainly because of the "lock" created by obstacle M_3. This problem cannot be solved by the algorithm of [15] since it belongs to the class of non-monotone problems. The planner of [10] does not take into account indirect obstacle interactions, and therefore would not consider moving obstacle M_2 before moving through the lock of obstacle M_3. Scenario B is more complex in the sense that it contains more obstacles. It is an axis-aligned version of the problem experimented on in [15]. The big obstacle M_1 must be moved out of the way of the robot, but before this is possible other obstacles have to be moved out of the way of M_1 first.

We performed our experiments on a 1.66 GHz Intel T5500 processor with 1 GByte of memory. Our algorithm solves scenario A in 0.01 seconds, and scenario B in 0.38 seconds. The solutions contain 214 and 23899 obstacle movements, respectively.

The solutions produced by our algorithm are not optimal; they include large amounts of unnecessary obstacle movements. This is because of the purely random nature of our algorithm. However, the solutions are found very fast (although the comparison is not entirely fair, [15] reports a running time of

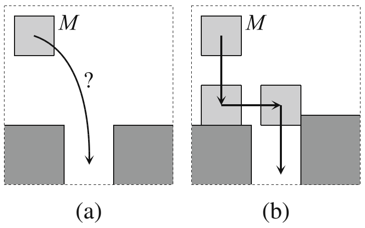

Fig. 7. **(a)** A narrow passage for movable obstacle *M* formed by two static obstacles. **(b)** *M* bumps against the obstacles to find its way through the narrow passage.

2.08 seconds on scenario B). This is explained by the efficiency of the rectangular map data structure, and our algorithm's inherent advantage in handling narrow passages: As the movable obstacles only move along axis-aligned paths, they implicitly use *compliance* to the static obstacles (see Fig. 7). Even if the boundaries of two (static) obstacles are collinear (Fig. 7a), there is an explicit ordering of the lines in the rectangular map data structure, so one is above the other. This is topologically equivalent to the situation of Fig. 7b, which the movable obstacles exploit to find their way through narrow passages.

7 Conclusion and Future Work

In this paper, we have discussed the problem of path planning among movable obstacles. We have made the observation that if we maintain an exact representation of the configuration space of the robot and the connected component the robot is in, we can *decouple* the computation of the obstacle movements, and the robot motions that lead to these obstacle movements. This approach to the problem enabled us to devise the first *probabilistically complete* algorithm for this domain.

We have presented a data structure called the *rectangular map* to maintain an exact representation of the robot's configuration space in case all obstacles and the robot are translating axis-aligned rectangles. We have implemented the algorithm and the data structure, and used it to solve problems that could not be solved by previous work.

The requirement to maintain an explicit representation of the robot's configuration space limits the practical applicability of our algorithm to robots with two or three degrees of freedom. Note, however, that the number of degrees of freedom of the movable obstacles is not constrained. Future work includes the implementation of a data structure to maintain the robot's configuration space in the more general case of polygonal and circular obstacles that can both translate and rotate. The arrangement package of CGAL [18] may provide most of the functionality required. Another possibility to address this limitation is to maintain the robot's connected component and its connectivity using sampling-based techniques, without sacrificing probabilistic completeness. This remains subject of future study.

The fact that our algorithm is probabilistically complete shows that we have characterized the problem correctly, but it does not necessarily say much about the performance of the algorithm. In fact, the algorithm that we have presented performs a rather uninformed brute force search. Enhancing the algorithm with *heuristics* to focus the search, such as the ones used in [10, 14, 15], might drastically improve the performance without losing probabilistic completeness. It may improve the quality of the produced solutions as well.

References

1. Alami, R., Laumond, J.-P., Siméon, T.: Two manipulation planning algorithms. In: Proc. Workshop on Algorithmic Foundations of Robotics, pp. 109–125 (1995)
2. Ben-Shahar, O., Rivlin, E.: Practical pushing planning for rearrangement tasks. IEEE Trans. on Robotics and Automation 14(4), 549–565

3. Canny, J.: The complexity of robot motion planning. MIT Press, Cambridge (1987)
4. Chadzelek, T., Eckstein, J., Schömer, E.: Heuristic motion planning with movable obstacles. In: Proc. Canadian Conf. on Computational Geometry, pp. 211–219 (2000)
5. Chen, P., Hwang, Y.: Practical path planning among movable obstacles. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 444–449 (1991)
6. Demaine, E., Demaine, M., O'Rourke, J.: Pushpush and Push-1 are NP-hard in 2D. In: Proc. Canadian Conf. on Computational Geometry, pp. 211–219 (2000)
7. Hsu, D., Kindel, R., Latombe, J.-C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. Int. J. of Robotics Research 21(3), 233–255 (2002)
8. Kavraki, L., Švestka, P., Latombe, J.-C., Overmars, M.: Probabilistic roadmaps for path planning in high dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)
9. LaValle, S., Kuffner, J.: Rapidly-exploring random trees: progress and prospects. In: Proc. Workshop on Algorithmic Foundations of Robotics (2000)
10. Nieuwenhuisen, D., van der Stappen, F., Overmars, M.: An effective framework for path planning amidst movable obstacles. In: Proc. Workshop on Algorithmic Foundations of Robotics (2006)
11. Ota, J.: Rearrangement of multiple movable objects. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 1962–1967 (2004)
12. Sánchez, G., Latombe, J.-C.: Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 2112–2119 (2002)
13. Siméon, T., Laumond, J.-P., Cortés, J., Sahbani, A.: Manipulation planning with probabilistic roadmaps. Int. J. of Robotics Research 23(7-8), 729–746 (2004)
14. Stilman, M., Kuffner, J.: Navigation among movable obstacles: real-time reasoning in complex environments. Int. J. of Humanoid Robotics 2(4), 479–504 (2005)
15. Stilman, M., Kuffner, J.: Planning among movable obstacles with artificial constraints. In: Proc. Workshop on Algorithmic Foundations of Robotics (2006)
16. Stilman, M., Schamburek, J.-U., Kuffner, J., Asfour, T.: Manipulation planning among movable obstacles. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 3327–3332 (2007)
17. Švestka, P., Overmars, M.: Coordinated path planning for multiple robots. Robotics and Autonomous Systems 23(3), 125–152 (1998)
18. Wein, R., Fogel, E., Zukerman, B., Halperin, D.: 2D Arrangements. CGAL User and Reference Manual, ch. 20 (2007)
19. Wilfong, G.: Motion planning in the presence of movable obstacles. Annals of Mathematics and Artificial Intelligence 3, 131–150 (1991)