OBPRM: An Obstacle-Based PRM for 3D Workspaces

Nancy M. Amato, *Texas A&M University, College Station, TX, USA*
O. Burchan Bayazit, *Texas A&M University, College Station, TX, USA*
Lucia K. Dale, *Texas A&M University, College Station, TX, USA*
Christopher Jones, *Texas A&M University, College Station, TX, USA*
Daniel Vallejo, *Texas A&M University, College Station, TX, USA*

Recently, a new class of randomized path planning methods, known as Probabilistic Roadmap Methods (PRMs) have shown great potential for solving complicated high-dimensional problems. PRMs use randomization (usually during preprocessing) to construct a graph of representative paths in C-space (a roadmap*) whose vertices correspond to collision-free configurations of the robot and in which two vertices are connected by an edge if a path between the two corresponding configurations can be found by a local planning method.*

This work describes and evaluates various node generation and connection strategies for one such PRM, the obstacle-based probabilistic roadmap method (OBPRM), in cluttered 3-dimensional Workspaces. Various node generation strategies are evaluated in terms of their ability to produce nodes in difficult regions of C-space; our results include recommendations for selecting appropriate node generation strategies for different types of objects, and a default strategy for use when objects cannot be classified easily. We also propose and analyze a multi-stage strategy for connecting the roadmap nodes; the use of different local planners at different stages is shown to enhance the connectivity of the resulting roadmap significantly.

1 Introduction

Automatic motion planning has application in many areas such as robotics, virtual reality systems, and computer-aided design. Although many different motion planning methods have been proposed, most are not used in practice since they are computationally infeasible except for some restricted cases, e.g., when the robot has very few degrees of freedom (dof) [11, 18]. Indeed, there is strong evidence that any complete planner (one that is guaranteed to find a solution or determine that none exists) will require time that is exponential in the number of dof of the robot [21]. For this reason, attention has focussed on randomized or probabilistic motion planning methods. Notable among these are randomized potential field methods (e.g., RPP [4]), which work very well when the configuration space (C-space) is relatively uncluttered, but unfortunately there also exist simple situations in which they are not successful [5, 15].

Recently, a new class of randomized motion planning methods has gained much attention (see, e.g., [1, 3, 10, 15, 17, 19, 20]). These methods, known as *probabilistic roadmap methods* (PRMs), use randomization (usually during preprocessing) to construct a graph in C-space (a *roadmap* [18]). Roadmap nodes correspond to collision-free configurations of the robot. Two nodes are connected by an edge if a path between the two corresponding configurations can be found by a local planning method. Queries are processed by connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points.

PRMs: PROBABILISTIC ROADMAP METHODS

I. PREPROCESSING: ROADMAP CONSTRUCTION
   1. NODE GENERATION (find collision-free configurations)
   2. CONNECTION (connect nodes to form roadmap) (repeat as desired)

II. QUERY PROCESSING
   1. CONNECT START/GOAL TO ROADMAP
   2. FIND PATH IN ROADMAP BETWEEN CONNECTION NODES

PRMs have been shown to perform well in practice. In particular, after the roadmap is constructed during preprocessing, many difficult planning queries can be answered in fractions of seconds [3, 17]. Although PRMs are particularly suitable when multiple queries will be answered in the same static environment, the general PRM strategy can be used to solve single queries by only constructing 'useful' portions of the roadmap [10, 20].

Node Generation. Node generation strategies are the methods used to select collision-free robot configurations to be used as nodes in the roadmap. A good node generation strategy will produce nodes that can be connected to form a roadmap that is representative of the connectivity and complexity of C-free. Ideally, the roadmap should contain nodes in every C-space crevice and corridor. However, guaranteeing this requires the costly computation of the constraint surfaces — which is what randomized methods seek to avoid.

The first PRMs [15, 17] use uniform sampling in C-space to generate roadmap candidate nodes (collision-free configurations are retained); roadmaps are enhanced by further sampling in 'difficult' regions. These methods perform well for general many-dof robots. However, their effectiveness decreases as the environments become more cluttered since uniform sampling of C-space is unlikely to yield configurations in narrow regions of C-space. To obtain improved roadmaps in crowded situations, some PRMs use information about the environment to guide node generation. Examples include executing random reflections at C-obstacle surfaces [9], and a technique called geometric node adding [20] for generating configurations of non-articulated robots near Workspace obstacle boundaries.

Connection. After the collision-free roadmap candidate nodes are generated, they must be connected to form the roadmap. The basic idea is to attempt to connect selected pairs of roadmap nodes using some local planning method(s); each successful connection identifies an edge in the roadmap. To save space, the paths found in this stage are not recorded since they can be re-generated quickly when processing queries.

The methods by which a PRM determines which (and how many) nodes to attempt to connect, and the local planner(s) selected to make those connections can crucially impact both the quality of the resulting roadmap and the running time of the PRM. Indeed, even though most PRMs greatly limit the number of connections attempted (say, to ten for each node), they still typically spend more than 95% of their preprocessing time in the connection phase [3, 17].

The general strategy of PRMs is to first make as many of the 'easy' and 'cheap' connections as possible, and then to use more sophisticated techniques to improve the roadmap's quality. For example, the PRM of [15, 17] first tries to connect each node to the k (a parameter) closest nodes (as determined by some distance metric) using the common straight-line in C-space local planner, and then attempts to enhance the roadmap by sampling more nodes in identified 'difficult' regions and/or by using more sophisticated local planners such as RPP [4].

1.1 Our Results

In this paper we consider an obstacle-based PRM (OBPRM) [3, 23] which samples points on or near C-obstacle surfaces. Even though the prototype implementation of OBPRM for planar articulated robots employed only the simplest node generation and connection strategies, it established that OBPRM was a promising method for planning in cluttered environments. Here, we describe several more sophisticated strategies for the node generation and connection phases, and provide an evaluation of a more mature implementation of the method for cluttered 3-dimensional Workspaces, typical, e.g., of mechanical designs [6]. The moving objects (robots) are rigid, non-articulated objects yielding six-dimensional C-spaces. Although we concentrate on OBPRM, we believe the techniques proposed here will be useful for PRMs in general, and for other motion planning approaches as well.

We propose and study various heuristics for generating nodes on or near C-obstacle surfaces and evaluate their effect on roadmap quality. The results of our study include recommendations for selecting appropriate combinations of node generation strategies for use with objects of certain general shapes and compositions (e.g., symmetrical with surfaces described by triangles of roughly equivalent area). We identify a default strategy for use with unclassified objects.

We also propose a multi-stage connection strategy that is shown to significantly improve the connectivity of the resulting roadmaps, while still controlling the time needed to construct them. The first stage attempts to make the 'easy' connections using many invocations of the fastest local planning method(s), while the later stages make a decreasing number of connection attempts and use increasingly more powerful local planning methods. The last stage may add new roadmap nodes as it attempts to join different connected components of the previous roadmap.

We remark that our goal is to provide empirical evidence that certain node generation and connection strategies work well for OBPRM for certain types of problems. We seek empirical evidence due to the randomized nature of PRMs, which makes them difficult to analyze. Recently, a number of attempts have been made to theoretically explain the success of PRMs (see, e.g., [10, 14, 16]). However, these studies generally make simplifying assumptions regarding the nature of the C-space and/or the PRM components (e.g., local planner), and therefore unfortunately cannot be applied to OBPRM.

2 Preliminaries

The moving objects (robots) considered in this paper are rigid objects in three-space. We represent configurations using six-tuples $(x, y, z, \alpha, \beta, \gamma)$, where the first three coordinates define the position and the last three define the orientation. The orientation coordinates are represented in radians, normalized to $[0 - 1)$.

In addition to collision detection, all PRMs make heavy use of so-called *local planners* and *distance computations.* Local planners are simple, fast, deterministic methods used to make connections between roadmap nodes when building the roadmap, and to connect the start and goal to the roadmap during queries. Distance metrics are used to determine which pairs of nodes one should try to connect.

The local planners and distance metrics used in OBPRM are based on recommendations from [2]. The distance metric we use is scaled Euclidean distance in C-space (the scale places more or less weight on the position coordinates).

The local planners currently implemented in OBPRM are the common *straight-line* in C-space, three versions of a (parameterized) planner proposed in [2] called *rotate–at–s* $(0 \leq s \leq 1)$, and some A^*-like methods (see, e.g., [7, 8, 12, 13, 22]). The suggested order to apply these planners was: first straight-line and rotate-at-$\frac{1}{2}$ (the most successful planners), next rotate-at-0 and rotate-at-1, and finally, the more expensive A^*-like planners, which try to fill in gaps left by the faster planners.

rotate-at-s. Briefly, when moving from c_1 to c_2, the rotate-at-s planner first translates from c_1 to an intermediate configuration c', rotates to a second intermediate configuration c'', and finally translates to c_2. The parameter s represents the fractional part of the translational distance between c_1 and c_2 that the robot travels from c_1 to c'. The straight-line planner is used to plan between each pair of configurations, that is, between (c_1, c'), between (c', c''), and between (c'', c_2).

The rotate–at–$\frac{1}{2}$ planner made more connections than the straight-line planner in nearly every situation studied, perhaps explained by the fact that it has a smaller swept volume. It is also fairly fast, taking about twice as long as the straight-line planner (significantly less than the A^* methods). The other two versions of rotate–at–s used here are $s = 0$ and $s = 1$, which although not as successful do make some connections the straight-line and rotate–at–$\frac{1}{2}$ do not.

A^*-like planners. The A^*-like planners used in OBPRM vary according to: (1) the number of neighboring configurations explored in each iteration, (2) the evaluation function used to select among the neighbors, and (3) the number of iterations they are allowed to run. Currently, OBPRM supports three different neighbor functions yielding three, nine, and fifteen neighbors. The first three neighbors, which are common to all functions, are the configurations in which (i) only the position coordinates, (ii) only the orientation coordinates, and (iii) all coordinates are incremented towards the goal. The additional six (or twelve) neighbors are the configurations in which exactly one of the coordinates is incremented towards (or away from) the goal. The two evaluation functions used are (i) minimum *distance* from the goal, and (ii) maximum *clearance* from

a workspace obstacle; both these functions are approximated using the center of mass of the relevant objects. The maximum number of iterations the planner is allowed to run is a multiple of the number of steps the straight-line planner would take; common values for this are 3, 6, 9, or 15. Note these planners are not truly A^* methods, but rather employ A^*-like strategies.

We denote the various versions of these planners by A^*−eval(nbrs,steps), e.g., A^*-clearance(9nbrs,6steps). Although significantly more expensive than the other planners, the A^*-like methods can sometimes make connections the others can not by 'feeling' their way along in tight places.

3 Node Generation in OBPRM

The prototype version of OBPRM [3, 23] for a many-dof articulated robot in a 2-dimensional Workspace used a simple strategy to generate nodes on contact surfaces. Briefly, for each obstacle X:

PROTOTYPE NODEGENERATION
1. c_{in} := colliding robot cfg with C-obstacle X
2. D := m random directions emanating out from c_{in}
3. for each $d \in D$
4. c_{out} := free cfg in direction d (if exists)
5. find contact cfg on (c_{in}, c_{out}) by binary search
6. endfor

This simple strategy was sufficient to establish the potential of the method. However, it is clear that more sophisticated node generation strategies are needed for more complex objects to produce a 'good' distribution of nodes in all the 'different' regions of C-free. Outlined below are some of the methods we've implemented and tested. Keeping in the spirit of OBPRM, all these methods employ information regarding the environment to guide node generation. Briefly, the methods are designed to generate three types of nodes: (i) *contact* configurations, (ii) *free* configurations (near contact surfaces), and (iii) sets of configurations (*shells*) surrounding C-obstacles.

Generating contact configurations. The node generation strategy used in the prototype version of OBPRM is attractive due to its simplicity and its efficiency (node generation typically accounted for 1-2% of preprocessing time). However, the distribution of the generated nodes is clearly very sensitive to both the shape of the C-obstacle and to the *seed* (origin c_{in} for the binary search). That is, no single seed will yield a good distribution of configurations on the surface of the C-obstacle if its shape is not roughly spherical, and even if the C-obstacle is spherically shaped, a seed removed from the C-obstacle's center will yield more configurations on the region of its surface closest to the seed.

Ideally, we would like a method that is as simple as the binary search technique, but which is less sensitive to the shape of the C-obstacle (which we wish to avoid computing explicitly). One way to achieve this is to use multiple seeds – so long as they are chosen in an intelligent manner. Indeed, the current version of OBPRM uses a different seed for (nearly) every node generated:

GENERATECONTACTCONFIGURATION
1. p_{rob} := point associated with robot
2. p_{obs} := point associated with obstacle of interest
3. c_{in} := translate robot so p_{rob} and p_{obs} coincide and rotate robot randomly until collision
4. d := random direction emanating out from c_{in}
5. c_{out} := free cfg in direction d (if exists)
6. find contact cfg on (c_{in}, c_{out}) by binary search

The types of configurations found in this manner depend upon the shapes of the robot and obstacle, and also on the selected points p_{rob} and p_{obj} (generally these points will be on the objects, e.g., a vertex). We have implemented and tested several methods for selecting points in an object (robot or obstacle) in OBPRM (see Table 1). There are, of course, many other possible variants (e.g., edges instead of triangles), but the ones we've implemented can be thought of as representative strategies.

Generating free configurations. Sometimes roadmap connectivity might be improved by generating some points in freespace – but near constraint surfaces. For example, such configurations may give local planners room to 'maneuver' around obstacle corners. It turns out the above described seed generation strategies can easily be adapted to generate such free configurations. In particular, after selecting the two points

Point Selection Strategies (Node Generation)	
Strategy	*Explanation and Bias*
cM center of mass	The center of mass of the object vertices. Good for (roughly) spherical objects, and also useful to ensure that some points are generated in the vicinity of the center of mass of that object.
rV random vertex	A random object vertex. Biased towards those portions of the object that have more vertices – which are generally the more complex regions.
eV extreme vertex	Select one of 6 extreme vertices with maximal and minimal x, y, and z coordinates at random. Finds configurations distributed around the object, even in regions which do not have high descriptive complexity (which might be captured by the random vertex case).
rT random triangle	Select a triangle at random, and then randomly select a point in that triangle. Biased towards those portions of the object that have more triangles – which are often the regions where the object's shape (e.g., curvature) changes.
wT weighted triangle	Select a triangle with probability proportional to its area, and then randomly select a point in that triangle. Biased towards triangles with large area (largest parts of object).

Table 1: *Point selection strategies for node generation.*

p_{rob} and p_{obj} in the robot and obstacle (according to one of the methods described above), we translate the robot until these points coincide and then randomly rotate the robot to obtain a *free* configuration (rather than a configuration in collision).

Building Shells. The general idea behind OBPRM is that the roadmap should be densest around the C-obstacles since that is where planning is hard. However, since planning is very difficult near contact surfaces, we would like to include paths in the roadmap that leave some clearance between the robot and the obstacles. Roadmap nodes that might be contained in such paths can be found at low cost during node generation. In particular, we can retain some number s of the configurations found during the search (e.g., the s closest, or s uniformly spaced among those found). We refer to s as the number of *shells* generated for the object (as they will hopefully encase the object).

4 Roadmap Connection in OBPRM

The first PRMs attempt connections between each node and the k (e.g., 10) closest nodes to it (as determined by some distance metric) [17]. This is a natural approach to take since initially there is little to differentiate one configuration from another. In OBPRM, however, this strategy would result in most connection attempts being between nodes associated with the same C-obstacle which may be more difficult to connect and also might not be as useful, in terms of roadmap connectivity, as connections between different C-obstacles.

There are three general stages in OBPRM's roadmap connection strategy. The first stage (Simple Connection) attempts to make the 'easy' connections using many invocations (typically thousands) of the fastest local planning method(s). The next two stages (Connecting Components and Growing Components) make a decreasing number of connection attempts and use increasingly more powerful local planning methods. The third stage may add new roadmap nodes as it attempts to join different connected components of the previous roadmap. The reason for the multi-stage strategy is that the faster local planners are not capable of making the crucial connections needed for a well connected roadmap, while the more powerful local planners are computationally too expensive to be used for the easier connections (which generally account for the majority of the roadmap edges).

Overviews of the strategies applied in each stage are described below. Within each stage, there are still several choices to be made, such as, for example, which local planner(s) and distance metric(s) to use. Based on the findings in [2], the current implementation of

OBPRM uses a scaled Euclidean distance metric and employs multiple local planners (see Section 2). We note, that it is likely that the best methods will vary according to the particular problem, and thus ideally the planner should adaptively select appropriate distance metrics and local planning methods.

Stage 1: Simple Connection. The first connection stage in OBPRM is similar to those used in the original PRMs. The main difference is that it takes advantage of the extra C-obstacle information associated with the nodes:

<u>STAGE 1: SIMPLECONNECTION</u>

```
let V_i be the set of nodes associated with C-obstacle i
let m be the number of C-obstacles
for each v ∈ V
   for i := 1,m
      Compute C_{v,i}, the closest K1 nodes in V_i to v.
      Try to connect v to each node in C_{v,i}.
   endfor
endfor
ANALYZEROADMAP /*compute connected components*/
```

The input parameter `K1`, the number of connections attempted for each node, is typically quite small (e.g., 10). Currently, the only local planner OBPRM uses in this stage is the *straight-line* in C-space, which is our fastest planner. After the `K1` connections have been attempted, the connectivity of the resulting roadmap is analyzed, e.g., its connected components, and their sizes, are computed. If, as is usually the case, the roadmap does not consist of a single connected component, we proceed to the next connection stage.

Stage 2: Connecting Components. The goal of this stage is to make connections between different connected components of the stage one roadmap. In particular, the strategy is similar to stage one, except that now connections are attempted between nodes that belong to different connected components rather than different C-obstacles:

<u>STAGE 2: CONNECTINGCOMPONENTS</u>

```
V_i = {v|v ∈ connected component i}, 1 ≤ i ≤ m
assume |V_1| ≤ |V_2| ≤ ... ≤ |V_m|
for i := 1,m /*connected components*/
   for j := i+1,m /*bigger connected components*/
      if |V_i| ≤ MAX2
        then ATTEMPTALL(V_i, V_j, K2.1)
        else ATTEMPTK(V_i, V_j, K2.2)
   endfor
endfor
ANALYZEROADMAP /*compute connected components*/
```

Each iteration of the inner for loop attempts to make a connection between two connected components V_i and V_j. If the size of the smaller component, V_i, is less than some constant `MAX2` (typically 10-30), then connection attempts are made between every node in V_i to the `K2.1` closest nodes in V_j. If, however, $|V_i| \geq$ `MAX2`, then connections are attempted for the `K2.2` closest pairs of nodes, one in V_i and one in V_j. In either case, the procedure returns as soon as one connection is made.

The `stage 2` strategy implemented in OBPRM is actually a bit more sophisticated in that it consists of several substages, each one patterned after the procedure outlined above. They differ from each other in the values of the parameters `MAX2`, `K2.1`, and `K2.2`, and in the set of local planners used to make the connections. In particular, there are two significant substages:

<u>`stage 2.1`</u>: `K2.1` $= 10$, `K2.2` $= 10$, `MAX2` $= 10$. Local Planners: straight-line in C-space, rotate-at-$\frac{1}{2}$.

<u>`stage 2.2`</u>: `K2.1` $= 10$, `K2.2` $= 8$, `MAX2` $= 10$. Local Planners: straight-line in C-space, straight-line in C-space, rotate-at-$\frac{1}{2}$, A^*-clearance(15nbrs,9steps).

The connectivity of the roadmap resulting from each substage is updated before the next substage is entered.

Stage 3: Growing Components. The first two connection stages try to make as many connections as possible between nodes created in the initial node generation stage. As in `stage 2`, the main goal of `stage 3` is to merge different connected components in the current roadmap. However, whereas in `stage 2` we simply tried to make connections between existing nodes in the two connected components of interest, in `stage 3` we also increase individual connected components by adding new nodes to them. The motivation is to 'grow' the connected components so that it will be easier to make connections between them. This stage is similar to the 'enhancement' methods used in PRMs [17, 10].

Currently, two different `stage 3` strategies are implemented in OBPRM. They may be used either independently, or in series.

Stage 3-fp: Expanding failed paths. The motivation behind this first method is that even when connection attempts between connected components fail, they may make some progress which might be useful. The general idea is to save the last valid node c_3 visited on a failed connection attempt between configurations c_1 and c_2, and to add this node to our roadmap:

STAGE 3-FP: EXPANDINGFAILEDPATHS

1. $V_i = \{v | v \in$ connected component $i\}$, $1 \leq i \leq m$
2. $cm_i :=$ the center-of-mass of V_i
3. **for** $i := 1, m$ /*connected components*/
4. $\quad V_j :=$ component with minimum $dist(cm_i, cm_j)$
5. $\quad P_{i,j} :=$ CLOSESTPAIRS(V_i, V_j,`K3.1`)
6. $\quad$ **for each** $(c_1, c_2) \in P_{i,j}$
7. $\quad\quad$ connect(c_1, c_2, c_3) /*c_3 is terminus of failed path*/
8. $\quad\quad$ **if** connect fails
9. $\quad\quad\quad$ **then** ATTEMPTALL(c_3, V_j,`K3.2`)
10. $\quad\quad$ **if** can't connect c_3 to V_j
11. $\quad\quad\quad$ **then** ATTEMPTALL($Nbrs(c_3), V_j$,`K3.3`)
12. $\quad\quad$ **if** can't connect any $v \in Nbrs(c_3)$ to V_j
13. $\quad\quad\quad$ **then** add c_3 to V_i
14. $\quad$ **endfor**
15. $\quad$ ANALYZEROADMAP /*compute connected components*/
16. **endfor**

This method uses the *center-of-mass* to select which connected components to try to connect (the center-of-mass of component V_i is the average of all the configurations in V_i). Next, for each pair (V_i, V_j) of connected components selected, it further selects the `K3.1` closest configuration pairs (c_1, c_2) in $V_i \times V_j$. If we cannot connect c_1 and c_2, we save the last valid node c_3 on this failed path. Then, we attempt to connect c_3 to the `K3.2` closest nodes in V_j. If that fails also, we attempt to connect each of the neighboring configurations $Nbrs(c_3)$ of c_3 to the `K3.2` closest configurations in V_j. If this too fails, then c_3 is added to the roadmap and the process repeats with the next closet pair (c'_1, c'_2) between V_i and V_j.

This whole process can be repeated until no further progress is obtained. Generally, `K3.1` and `K3.2` should be small constants; we have had success using `K3.1`$=$ 15 and `K3.2` $= 10$. Our current implementation uses the A^*-clearance(15nbrs,9steps) local planner for the first two connection attempts (lines 7 and 9) and the rotate-at-s planners for the third attempt (line 11).

Stage 3-sc: Expanding small components. The motivation behind this method is that the configurations that are in 'difficult' regions of C-space are likely to be contained in the smaller connected components, and moreover, these small components may prove instrumental as 'bridges' between the larger connected components. The general idea is to expand the small components by generating more configurations near configurations already in that component.

STAGE 3-SC: EXPANDINGSMALLCOMPONENTS

1. $V_i = \{v | v \in$ connected component $i\}$, $1 \leq i \leq m$
2. **for** $i := 1, m$ /*connected components*/
3. $\quad$ **if** $|V_i| <$ `MAX3`
4. $\quad\quad c :=$ random configuration in V_i
5. $\quad\quad$ generate $Nbrs(c)$
6. $\quad\quad$ **for each** $c' \in Nbrs(c)$
7. $\quad\quad\quad$ ATTEMPTALL(c', V_i, `K3.3`)
8. **endfor**
9. run CONNECTINGCOMPONENTS /* `stage 2` */

As with the other methods, there are several parameters that must be selected appropriately: `MAX3` (the cut-off for small components) and `K3.3` (the number of configurations to try to connect with each neighbor c'). In addition, one must select the number and type of neighbors one would like to generate, and the local planners to use to attempt those connections. The current implementation uses `MAX3` $= 10$, `K3.3` $= 10$, and considers the fifteen neighbors checked by the A^* method. The straight-line planner is used to try to connect each $c' \in Nbrs(c)$ to c's connected component V_i (line 7), and A^*-clearance(15nbrs,9steps) is the planner used in the call to `ConnectingComponents` (the `stage 2` connection strategy).

5 Evaluating OBPRM

The experiments described below were designed to evaluate the node generation and connection strategies in cluttered Workspaces containing narrow C-space corridors. All programs were written in C, and run on SGI machines.

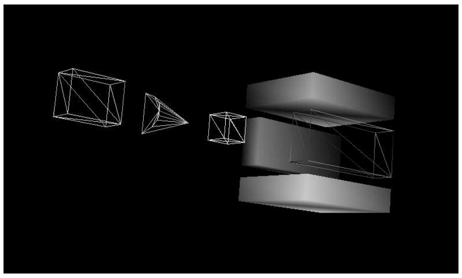

Figure 1: *Corridor 1.*

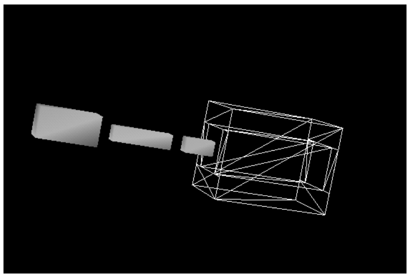

Figure 2: *Corridor 2.*

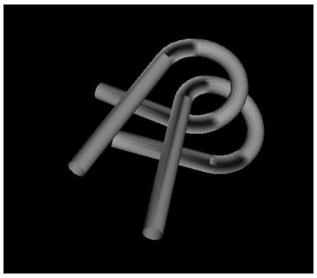

Figure 3: *Alpha puzzle.*

Environments. We considered two Workspace environments containing easily identifiable C-space corridors, and an interesting puzzle. (See Figures 1–3.)

Workspace *corridor 1* had dimension $2 \times 3 \times 5$, and had three different moving objects (robots): a $1 \times 1 \times 1$ cube (easy, rotates easily in corridor), a $1 \times 1.5 \times 2$ block (hard, cannot rotate freely), and a 'half-cone' that has roughly the same dimensions as the block (moderate).

Workspace *corridor 2* consisted of a tunnel of dimension $3 \times 8 \times 3$ contained in a bounding box of dimension $5 \times 28 \times 6$. Again, we considered three moving objects (robots): a small block of dimension $1 \times 2 \times 1$ (easy), a medium block of dimension $1 \times 4 \times 1$ block (moderate), and a large block of dimension $2 \times 4 \times 2$ block (hard).

We also considered an *alpha puzzle* environment containing two tubes twisted into an α shape (1008 triangles per tube). The objective is to separate the intertwined tubes by a sequence of rotations and translations. The puzzle can be made easier by scaling the obstacle tube in one dimension, which has the effect of shrinking or widening the gap between the two prongs of the α and simultaneously transforming the tube's cross-section from a circle to an increasingly thinner and longer ellipsoid.

5.1 Node Generation

The two most important criteria for evaluating a node generation scheme are, first, its ability to generate nodes in difficult regions of C-space, and second, whether the generated nodes can be connected into a roadmap. We used the Corridor 1 environment to evaluate these criteria.

Generating nodes in difficult regions. The nature of the Corridor 1 environments made it easy to determine whether a generated configuration was in the corridor. In particular, using the seed generation strategy of interest, and a chosen number of shells (1, 3, or 5), we first attempted to generate the requested number of nodes (100 per obstacle, or 400 total for shell 1, and 1200 or 2000 total for shells 3 and 5, respectively). Then, each resulting node was classified as being `inside` (all vertices inside the corridor), `border` (some vertices inside and some outside), or `outside`. The values reported are averages of 10 runs.

As expected, it became increasingly difficult to generate `inside` configurations for the cube, the half-cone, and the block. However, the same did not hold for the `border` configurations – which are also very important since they represent the entrance to the corridor. In fact, for all shells, more border configurations were generated for the half-cone than for the cube or block. The reason for this is that the half-cone is longer than the cube and thinner than the block (and thus there are more border configurations possible for it).

Usually the best strategy for at least one of the objects was a combined strategy (i.e., using multiple strategies). This was true even though some of our strategies are more or less indistinguishable for the objects we studied, e.g., random versus weighted triangles for the cube, or extreme versus random vertices for all objects. We suspect this will only be magnified when more complex objects are studied. However, we did see that certain strategies are preferred for certain types of objects — and these adhere to our expectations as listed in Table 1. In particular, **cM** is good for symmetrical objects like the cube, **eV** and **wT** cover important, and perhaps different, object features, and **rV** and **rT** help assure good coverage.

Our findings seemed to indicate that multiple shells are *not* useful for generating `inside` or `border` configurations. For the cubes the percentage of `inside/border` configurations was approximately 10% in the first shell and between 7% and 8% in the second and third shells; a similar relation existed for the half-cone and block. However, our experience showed that three shells did improve roadmap quality, and thus they should probably be retained since they come for free during node generation.

Generating connectable nodes. The connectivity of the resulting roadmap is the other crucial criterion for judging a node generation scheme. Since we were concerned with our ability to find paths *through* C-space corridors, we pruned the `outside` configurations and built `stage 1` roadmaps using only the `inside` and `border` configurations.[1]

Figure 4 shows, for the cube and half-cone, respectively, four `stage 1` roadmaps built using different seed generation strategies; all roadmaps were constructed using all local planners. Each connected component of a roadmap is represented by a bar showing the number of nodes in that component. Thus, in general, the fewer and the taller the bars of a roadmap, the better it is. Of course, it is also desirable for the large connected components to contain `inside` nodes (shown in black).

In general, the best roadmaps were obtained using the combined strategy **rV+wT**. It can be seen from the figures that using **all** the seed strategies yields roadmaps whose connectivity is approximately an average of the others. Thus, if one knows enough about the shapes of the objects, then some advantage can be gained by selecting node generation strategies suited to those objects. However, if this knowledge is not available, then roadmaps with reasonable connectivity can be obtained by using all the strategies simultaneously. In this case, it might be advisable to first build small roadmaps using the various strategies, and then to select appropriate strategies based on these 'test' roadmaps.

[1] Notice that the pruning may reduce connectivity since we may remove `outside` nodes that 'bridge' different connected components that pass through different C-space corridors corresponding to the one Workspace corridor.

5.2 Connection

The success of the roadmap connection phase depends on both the chosen local planning methods and on the the strategies used to select candidate nodes for connection.

Choosing Local Planners. To test the connectivity of the roadmaps resulting from different (combinations of) local planners, we used the Corridor 1 environments, generated sets of nodes (once), and analyzed the connected components in the `stage 1` roadmaps built using different (combinations of) planners. (See Figure 5.)

Each figure shows the connected components of the roadmap constructed using the different combinations of local planners. It is clear the connectivity of the resulting roadmap is improved by using multiple local planners, and that none of them is totally redundant. That is, there are some cases in which each local planner makes the crucial connections between different connected components. For example, the half-cone roadmaps in Figure 5 show that the straight-line and rotate-at-$\frac{1}{2}$ planners make different connections (Rdmp 3 vs Rdmps 1 and 2), and that the A^*-like planners (Rdmps 5 and 6) were needed to obtain one connected component.

While it is true that connectivity can be greatly enhanced by using multiple local planners, it should be remembered that this implies a larger cost for each connection. Some sample node generation and connection times for the `stage 1` (hybrid) roadmaps environment are shown in Table 2; note that the connection times for the Corridor 1 environments are only for the reduced set of `inside` and `border` nodes. As can be seen, the A^*-like planner has a large impact on the running time.

Multi-stage Connection. To evaluate our multi-stage connection strategy, we used the Corridor 2 environments. In each case, we first generated nodes,

Roadmap Construction, seed strategy 8 (all), no. shells = 3															
Envt	Generation			Roadmap Connection											
		#nodes	#nodes	Rdmp1		Rdmp2		Rdmp3		Rdmp4		Rdmp5		Rdmp6	
	sec	all	in+bd	sec	#cc	sec	#cc	sec	#cc	sec	#cc	sec	#cc	sec	#cc
cube	10	872	100	25	21	46	19	49	15	75	7	175	7	199	6
half-cone	22	834	86	31	12	58	18	61	9	105	6	232	6	269	5
block	15	736	73	13	23	23	31	28	23	67	10	162	9	188	9
alpha	2	15	n/a	3	11	4	14	6	10	18	9	30	7	41	7

Table 2: *Preprocessing computation times and statistics for* `stage 1` *(hybrid) roadmaps.*

Multi-Stage Roadmap Connection (node generation: seed strategy 77, no. shells = 3)												
Envt	Generation		Roadmap Connection									
			Stage 1		Stage 2.1		Stage 2.2		Stage 3-fp		Stage 3-sc	
	sec	#nodes	sec	#cc	sec	#cc	sec	#cc	sec	#cc	sec	#cc
sm blk	17	327	63	12	29	6	257	2	1870	2	76	2
sm blk	47	839	159	20	120	7	645	2	4	1	0	1
md blk	7	21	3	7	14	6	487	2	27	1	96	1
md blk	25	128	32	14	34	12	797	3	330	1	331	1
md blk	36	183	42	19	101	11	709	2	5	1	70	1
lg blk	56	72	13	4	1	4	276	2	7020	2	249	2
lg blk	85	141	23	13	13	11	909	2	2273	1	169	2
lg blk	115	198	29	14	27	13	1409	2	3	1	32	1
lg blk	116	210	23	14	41	13	1114	2	14	1	93	2
alpha	15	591	156	27	31	21	3206	10	25832	7		
alpha	129	4949	1346	47	202	22	3440	17	29637	14		

Table 3: *Preprocessing computation times and statistics for multi-stage roadmap connection.*

and then performed the various connection stages; the `stage 3` connection methods were each applied to the roadmap produced in `stage 2.2`. Some resulting roadmaps are shown in Figure 6. The roadmap labels correspond to the connection stage that produced it. In almost every case, there were clear connectivity benefits obtained from one stage to the next. For example, even though `stage 2.1` considers the same set of nodes and uses only the straight-line and rotate-at-$\frac{1}{2}$ local planners, it still usually produced a better connected roadmap. The first time the A^* planners are used is in `stage 2.2`. In every case studied, there was a significant improvement in connectivity between the `stage 2.1` and `stage 2.2` roadmaps. This was particularly true as the environments increased in difficulty.

By design, the Corridor 2 environment has a tendency to have two connected components, one on either side of the corridor. For the medium and large blocks, the more sophisticated `stage 3` strategies proved crucial for joining these components. It is interesting to note that the `stage 3` strategies sometimes discovered crucial **inside** nodes that enable them to connect the two components (e.g., for the large block).

Node generation and connection times for the various connection stages are shown in Table 3. As expected, `stage 1` and `stage 2.1` are generally the fastest since they employ only the faster local planners (straight-line and rotate-at-$\frac{1}{2}$). It can also be seen that the time spent in `stage 1` depends on the number of nodes, and the time spent in `stage 2.1` depends on the number of connected components. The times spent in `stage 2.2` and `stage 3` are generally significantly greater due to the A^*-like planners. Thus, while it is clear that these methods can be highly effective, they must be used with care in order for the resulting methods to be computationally feasible.

Alpha-puzzle. The hardest problem studied in this paper is the alpha-puzzle. This problem illustrates that different problems require different types of node generation strategies. A representative roadmap for this problem must include both configurations in which the two tubes are intertwined and separated. Through experimentation, we found that such configurations are most easily generated using a strategy of **rV+wT** for one tube and **cM** for the other for the *free nodes*; using this strategy, the generated nodes are split roughly 80/20 between intertwined and separated configurations. In contrast to the corridor environments, the contact configurations did not seem to be as useful here.

At present, OBPRM has succeeded in solving a version of the problem in which the obstacle tube is scaled by 1.2 in the z-direction (perpendicular to the tube's cross-section). This version was solved using a `stage 1` roadmap. Although, it is a rather close fit, the gap between the prongs of the α in this version is large enough to enable the other tube to escape by 'sliding' out (rather than the complicated sequence of rotations and translations needed to solve the original version). Currently, we are working on this problem using our multi-stage connection strategy. Based on our preliminary results we believe we will be able to solve (at least) the version in which the obstacle tube is scaled by 1.1.

6 Conclusion

In this paper, we describe and evaluate several strategies for node generation and propose a multi-stage connection strategy for OBPRM in cluttered 3-dimensional Workspaces.

Our evaluation of various node generation strategies in terms of their ability to produce nodes in difficult regions of C-space, and in terms of the connectivity of the resulting roadmaps, yields recommendations for selecting appropriate combinations of node generation strategies for different types of objects (robot or obstacle). When the objects cannot be classified easily, we recommend a default strategy which is essentially a combination of all the others. While it will likely not be the best strategy in any given situation, it seems to give reasonable results for the problems we've studied. Finally, when the relative importance of the contact configurations and the free configurations cannot be determined *a priori*, we recommend generating roughly half of each.

We also show that roadmap connectivity can be greatly improved by using a multi-level connection strategy and multiple local planners. However, this improvement in connectivity does require significant computation. Although our study was aimed at OBPRM, we believe our techniques and conclusions will be useful for PRMs in general, and for other motion planning approaches as well.

Currently, we are working to make OBPRM adapt automatically to the environment (e.g., automatically selecting seed generation strategies), and are further refining the connection strategies described here. We are also planning to augment our implementation for planning contact tasks.

Acknowledgement

This research was supported in part by NSF CAREER Award CCR-9624315 (with REU Supplement), NSF Grant IRI-9619850, and by the Texas Higher Education Coordinating Board under Grant ARP-036327-017. O. Burchan Bayazit is supported in part by the Turkish Ministry of Education. Lucia K. Dale is supported in part by a GE Foundation Graduate Fellowship. Daniel Vallejo is on leave from Universidad de las Americas-Puebla, Mexico and is supported in part by a Fulbright-CONACYT scholarship.

We would like to thank the robotics group at Texas A&M, especially Jeff Trinkle and Li Han, for their suggestions regarding this work. We are also grateful to John Canny, Shane Chang, Lydia Kavraki, and Jean-Claude Latombe for useful discussions and comments. The alpha puzzle was designed by Boris Yamrom of the Computer Graphics & Systems Group at GE's Corporate Research & Development Center. GE also provided us with *Product Vision* (their CAD animation package) which was used to produce the environment

snapshots shown in this paper, and our collision detection routine is based on code developed at GE.

References

[1] J. M. Ahuactzin and K. Gupta. A motion planning based approach for inverse kinematics of redundant robots: The kinematic roadmap. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 3609–3614, 1997.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 1998. To appear.

[3] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 113–120, Minneapolis, MN, April 1996.

[4] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Internat. J. Robot. Res.*, 10(6):628–649, 1991.

[5] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 2, pages 46–51, 1993.

[6] H. Chang and T. Y. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1012–1019, 1995.

[7] P. C. Chen and Y. K. Hwang. SANDROS: A motion planner with performance proportional to task difficulty. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 2346–2353, 1992.

[8] B. Glavina. Solving findpath by combination of directed and randomized search. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1718–1723, 1990.

[9] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom – random reflections at c-space obstacles. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 3318–3323, 1994.

[10] D. Hsu, J-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 2719–2726, 1997.

[11] Y. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[12] Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Trans. Robot. Automat.*, 8(1):23–32, 1992.

[13] Y. K. Hwang and P. C. Chen. A heuristic and complete planner for the classical mover's problem. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 729–736, 1995.

[14] L. Kavraki, M. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 4, pages 3020–3025, 1996.

[15] L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 2138–2145, 1994.

[16] L. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query preprocessing in robot path planning. In *Proc. ACM Symp. Theory of Computing*, pages 353–362, 1995.

[17] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[18] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[19] M. Overmars. A random approach to path planning. Technical Report RUU-CS-92-32, Computer Science, Utrecht University, The Netherlands, 1992.

[20] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.

[21] J. Reif. Complexity of the piano mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 421–427, 1979.

[22] P. Watterberg, P. Xavier, and Y. Hwang. Path planning for everyday robotics with sandros. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1170–1175, 1997.

[23] Y. Wu. An obstacle-based probabilistic roadmap method for path planning. Master's thesis, Department of Computer Science, Texas A&M University, 1996.

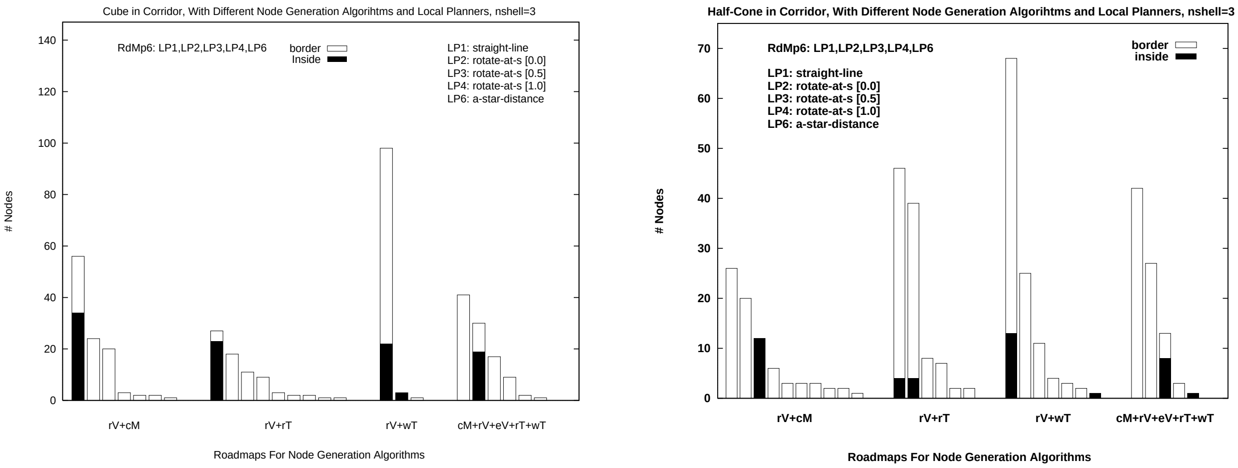

Figure 4: *Node Generation:* `stage 1` *roadmaps (all local planners), corridor 1, cube and half-cone, respectively.*

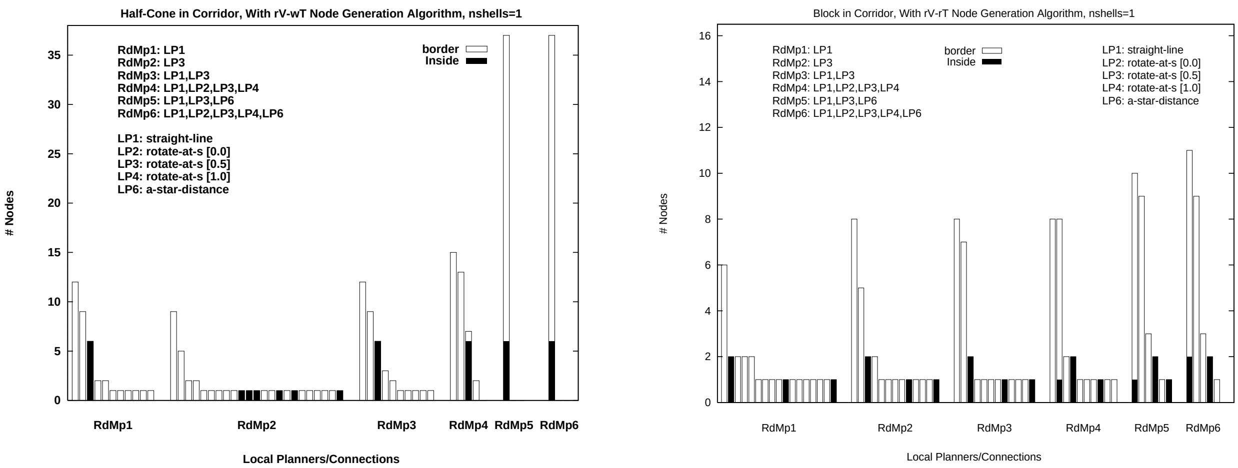

Figure 5: *Local Planners:* `stage 1` *roadmaps, corridor 1, half-cone and block, respectively.*

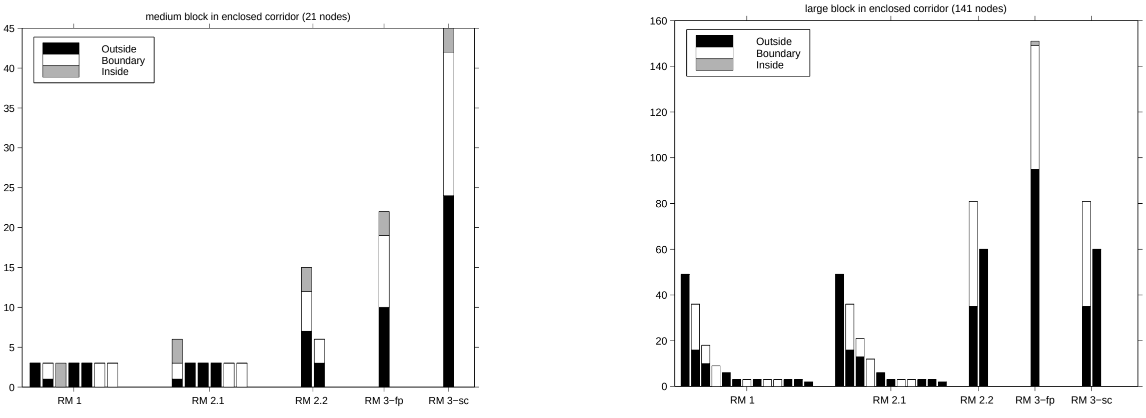

Figure 6: *Connection Stages:* `stage` *1, 2.1, 2.2, 3-fp, 3-sc roadmaps, corridor 2, medium and large block, respectively.*