Path Planning Using Lazy PRM

Robert Bohlin
Department of Mathematics
Chalmers University of Technology
SE–412 96 Göteborg, Sweden

Lydia E. Kavraki
Department of Computer Science
Rice University
Houston, TX 77005, USA

Abstract

This paper describes a new approach to probabilistic roadmap planners (PRMs). The overall theme of the algorithm, called Lazy PRM, is to minimize the number of collision checks performed during planning and hence minimize the running time of the planner. Our algorithm builds a roadmap in the configuration space, whose nodes are the user-defined initial and goal configurations and a number of randomly generated nodes. Neighboring nodes are connected by edges representing paths between the nodes. In contrast with PRMs, our planner initially assumes that all nodes and edges in the roadmap are collision-free, and searches the roadmap at hand for a shortest path between the initial and the goal node. The nodes and edges along the path are then checked for collision. If a collision with the obstacles occurs, the corresponding nodes and edges are removed from the roadmap. Our planner either finds a new shortest path, or first updates the roadmap with new nodes and edges, and then searches for a shortest path. The above process is repeated until a collision-free path is returned.

Lazy PRM is tailored to efficiently answer single planning queries, but can also be used for multiple queries. Experimental results presented in this paper show that our lazy method is very efficient in practice.

1 Introduction and Motivation

The basic path planning problem is to find collision-free paths for a moving object – a robot – among stationary, completely known obstacles. Denoting the configuration space of the robot by $\mathcal{C}$ and the open subset of collision-free configurations by $\mathcal{F}$, the problem can be stated as follows: given an initial configuration q_{init} and a goal configuration q_{goal} in $\mathcal{F}$, find a continuous curve in $\mathcal{F}$ connecting these points, or determine that none exists [20]. Path planning is becoming increasingly important for automated manufacturing and for mobile robots, but it has also found applications in computer animations, medical surgery, and molecular biology (see [21]).

Requirements for Single Query Path Planning Of particular interest are planners that with little preprocessing can answer single queries very quickly. Such planners can be used to re-plan paths in applications where the configuration space obstacles could change. This occurs, for instance, when the robot changes tools, grasps an object, or a new obstacle enters the workspace.

Ideally, the time required for planning should relate to the difficulty of the planning task, i.e., a simple path in an uncluttered environment should be found quickly, while a more complicated path may require more time. In a similar way, the planning time should relate to the desired quality of the returned path. The quality of a path is difficult to quantify (see also Section 3.2), but in general we prefer short paths in $\mathcal{C}$, with respect to some metric. The planner should allow the user to intuitively tune short planning time versus high quality.

We would also like the planner to learn to some extent, i.e., to use information from previous queries in order to speed up subsequent queries. For example, if the planner finds a path through a narrow passage in $\mathcal{F}$, it should be able to use that information when searching for a path through the same passage in subsequent queries.

Contribution of This Paper In this paper we present a lazy approach to probabilistic roadmap planners (PRMs). Our main objective is to provide a very fast planner that adheres as closely as possible to the requirements discussed above. We address standard industrial applications characterized by complex geometry and high-dimensional, relatively uncluttered configuration spaces. To handle the complex geometry, which generally implies time-consuming collision-checking, the main theme of the algorithm is to minimize the number of collision-checks performed during planning. By avoiding local planning and instead keeping the global view, only the part of the configuration space that is essential in answering a query is explored. Experiments in a typical industrial environment show that a very large percentage, on the average 26%, of the total number of collision checks are actually performed on the returned collision-free paths, and are therefore inevitable.

Solutions for the cases of cluttered configuration spaces and fast collision-checking are also proposed, but neither these cases nor the narrow passage problem (see [2, 6, 12]) are our main objectives. Our algorithm, called Lazy PRM, is described in detail in Section 3, and experimentally evaluated in Section 4 using a real industrial environment.

2 Probabilistic Techniques

The path planning problem has been extensively studied in the last two decades, and a number of different approaches are proposed; see [10, 14, 20] for overviews. An algorithm is called *complete* if it always

will find a solution or determine that none exists. However, due to the complexity of the path planning problem [8], complete planners are far too slow to be useful in practice. Trading completeness for speed, probabilistic planners have been successfully applied to many problems in high-dimensional configuration spaces.

The Randomized Path Planner (RPP) in [4] has successfully solved problems for robots with more than 60 degrees of freedom [19]. The planner uses a potential field as a guidance towards the goal, and random walks to escape local minima.

Another interesting approach is presented in [24] – the Ariadne's clew algorithm. Considering the initial configuration as a landmark, the planner incrementally builds a tree of feasible (i.e., collision-free) paths as follows. Genetic optimization is used to search for a collision-free path from one of the landmarks to a point as far as possible from all previous landmarks. The planner places a new landmark at this point, and searches for a path to the goal configuration. New landmarks are placed until the goal configuration can be connected to the tree.

2.1 Probabilistic Roadmap Method

The brief survey in this section is restricted to probabilistic techniques. In particular, the Probabilistic Roadmap Method [18] is described in detail, since the method forms the base of our solution. The idea behind the Probabilistic Roadmap Method (PRM), described in [17, 18, 27], is to represent and capture the connectivity of $\mathcal{F}$ by a random network, a *roadmap*, whose nodes and edges correspond to randomly selected configurations and path segments respectively. In a preprocessing step, or a *learning phase*, a large number of points are distributed uniformly at random in $\mathcal{C}$, and those found to be in $\mathcal{F}$ are retained as nodes in the roadmap. A local planner is then used to find paths between each pair of nodes that are sufficiently close together. If the planner succeeds in finding a path between two nodes, they are connected by an edge in the roadmap. In the *query phase*, the user-specified start and goal configurations are connected to the roadmap by the local planner. Then the roadmap is searched for a shortest path between the given points.

Even though a powerful local planner will require few nodes to obtain a well connected roadmap, most implemented PRMs show that it is computationally more efficient to distribute nodes densely and use a relatively weak, but fast, local planner [18, 27]. The local planner may for instance only check the straight line between two nodes. Other local planners are discussed and evaluated in [1].

Often the learning phase of PRM has a *node enhancement* step in order to increase the connectivity of the roadmap by adding more nodes in difficult regions of $\mathcal{F}$. Different techniques are used to identify these regions; one way is to distribute new points close to a number of *seeds* randomly selected among the existing nodes. In [17], the probability that a node is selected is proportional to $\frac{1}{1+b}$, where b is the number of edges connected to the node. An alternative selection can be based on a node's ratio of failed attempts by the local planner to find paths to other nodes [18]. Other techniques to increase the connectivity of the roadmap are described in [2] and [11].

PRM has been shown to work well in practice in high-dimensional configuration spaces [17]. In particular, it is useful for multiple queries, since once an adequate roadmap has been created, queries can be answered very quickly.

2.2 Variations of PRM

PRM has a weakness in finding paths through narrow passages in $\mathcal{F}$. The node enhancement step in [17] was developed in an effort to address this issue. Other efforts have been made; one is to distribute nodes close to the boundary of $\mathcal{F}$, and has led to several new sampling strategies. The planner in [12] initially allows the robot to penetrate the obstacles to a certain extent. Small neighborhoods around the configurations just in collision are then re-sampled in order to place nodes close to the boundary of $\mathcal{F}$. The Obstacle Based PRM (OBPRM) in [2, 3] determines configurations in collision to be origins of a number of rays. Binary search is then used along each ray to find points on the boundary of $\mathcal{F}$, where roadmap nodes are placed. In [6], the planner identifies the boundary of $\mathcal{F}$ by distributing points in pairs. Each pair is generated by first picking one point uniformly at random in $\mathcal{C}$, and then picking another point close to the first one. One of the points is added to the roadmap only if it is in $\mathcal{F}$ and its pair is not. Another technique to increase the number of nodes in narrow passages of $\mathcal{F}$ is presented in [29]; points are picked uniformly at random in $\mathcal{C}$ and then retracted onto the medial axis of $\mathcal{F}$.

A few methods using probabilistic roadmaps, do not divide the planning process into a learning phase and a query phase. Given an initial and a goal configuration, the planner in [26] inserts randomly distributed nodes in $\mathcal{F}$, one at a time, and connects them to the different components of the roadmap by a local planner. New nodes are inserted until the initial and goal configurations can be found in the same connected component of the roadmap. See also [9] and [15] for related algorithms. The latter paper gives an adaptive scheme for adjusting the power of the local planner.

Other methods, described in [13] and [22], build two trees rooted at the initial and goal configurations respectively. As soon as the two trees intersect, a feasible path can be extracted. In [13], the trees are expanded by generating new nodes randomly in the vicinity of the two trees, and connecting them to the trees by a local planner. The planner in [22] iteratively generates a configuration, an attractor, uniformly at random in $\mathcal{C}$. Then, for both trees, the node closest to the attractor is selected and a local planner searches for a path of a certain maximum length towards the attractor. A new node is placed at the end of both paths. The process stops when the two trees intersect.

The general theme for roadmap algorithms is to construct a network of paths verified to be collision-free by a local planner. Unfortunately, it is difficult to find a global strategy that can use local planners efficiently in order to avoid regions from where the algorithm can-

not proceed to the goal. This often means that too much time is spent on planning local paths that will not appear in the final path. Moreover, most probabilistic algorithms either heavily rely on fast collision checking or require long preprocessing. Collision checking is in many real applications with complex workcells very time consuming, making probabilistic planners yet too slow for single queries in industrial environments in which small changes occur.

Our solution is to avoid using local planners as much as possible, and instead keep a more global view throughout the entire planning process. In the next section, we present Lazy PRM – a path planning algorithm tailored for single queries, but which is also useful for multiple queries. To make the planner fast, the main theme is to minimize the number of collision checks.

3 Lazy PRM

This section describes a new algorithm for single and multiple query path planning. The algorithm is similar to the original PRM in [17] in the sense that the aim is to find the shortest path in a roadmap generated by randomly distributed configurations. In contrast with existing PRMs, we do not build a roadmap of feasible paths, but rather a roadmap of paths *assumed* to be feasible. The idea is to lazily evaluate the feasibility of the roadmap as planning queries are processed. Similar ideas about lazy evaluation have been developed concurrently but independently in [25] in a planner called Fuzzy PRM.

In other words, let q_{init}, q_{goal}, and a number of uniformly distributed points form nodes in a roadmap, and connect by edges each pair of nodes being sufficiently close together. Given a procedure that estimates the length of a path, we find a shortest feasible path in the roadmap by repeatedly searching for a shortest path, and then checking whether it is collision-free or not. Each time a collision occurs, we remove the corresponding node or edge from the roadmap, and search for a new shortest path.

This procedure can terminate in either of two ways. If there exist feasible paths in the roadmap between q_{init} and q_{goal}, we will find a shortest one among them. Otherwise, if there is no feasible path, we will eventually find q_{init} and q_{goal} in two disjoint components of the roadmap. In the latter case we either report failure or, if we still have time, add more nodes to the roadmap by a procedure we call node enhancement, and start searching again. A high-level description of the algorithm is given in Figure 1, and the rest of this section explains the different steps of the algorithm in more detail.

3.1 Building the Initial Roadmap

The first step of the algorithm is to build a roadmap $\mathcal{G}$ in $\mathcal{C}$ consisting of $\mathrm{q}_{init} \in \mathcal{F}$, $\mathrm{q}_{goal} \in \mathcal{F}$, and N_{init} nodes distributed uniformly at random. N_{init} is a parameter determined by the user. We can, of course, use heuristics to increase the density of nodes in regions we in advance believe are of particular interest, but this is in general difficult.

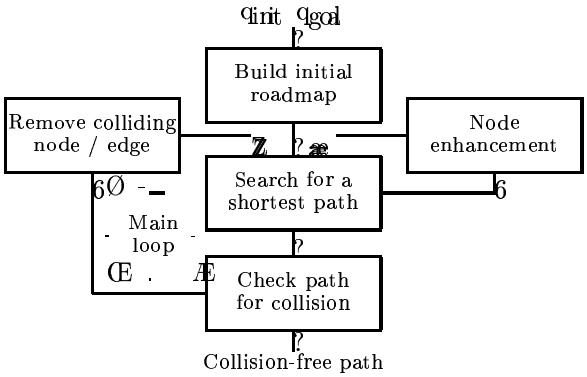

Figure 1: High-level description of Lazy PRM.

Selecting Neighbors We connect each node in $\mathcal{G}$ by edges to a set of neighbor nodes. An edge represents the straight line path in $\mathcal{C}$ between two nodes. (In principle, any other local technique for connecting nodes could be substituted here.) Since it would take far too much memory to connect all pairs of nodes, and it is less likely that the straight line path between two nodes far apart is feasible, it is natural to only consider nodes which are sufficiently close together.

In order to select reasonable neighbors, we need a metric $\rho_{coll} : \mathcal{C} \times \mathcal{C} \to [0, \infty)$ such that the distance between two configurations under this metric reflects the difficulty of connecting them by a collision-free straight line path. Then we connect each pair of nodes $(\mathrm{q}, \mathrm{q}')$ such that $\rho_{coll}(\mathrm{q}, \mathrm{q}') \leq R_{neighb}$. For any fixed radius R_{neighb}, the number of neighbors of a node is a random variable, so depending on the initial number of nodes, N_{init}, we choose R_{neighb} such that the expected number of neighbors equals the parameter M_{neighb} determined by the user.

In many cases it is harder to make feasible connections in certain directions than in others. Consider for instance an articulated robot arm; then it is more likely that a collision occurs when the base joint is moving one unit, than if a joint close to the end-effector is moving one unit. With this in mind, we let ρ_{coll} be a weighted Euclidean metric,

$$\begin{aligned} \rho_{coll}(\mathrm{x}, \mathrm{y}) &= \Big(\sum_{i=1}^{d} w_i^2 (x_i - y_i)^2 \Big)^{1/2} \\ &= \big((\mathrm{x} - \mathrm{y})^T W (\mathrm{x} - \mathrm{y}) \big)^{1/2}, \end{aligned} \qquad (1)$$

where d is the dimension of $\mathcal{C}$, $\{w_i\}_{i=1}^{d}$ are positive weights, $W = diag(w_1^2, \ldots, w_d^2)$, and x^T is the transpose of x. The weights are chosen in proportion to the maximum possible distance (Euclidean distance in the workspace) traveled by any point on the robot, when moving one unit in $\mathcal{C}$ along the corresponding axis. This metric is easy to use and has been shown to work well in our experiments presented in Section 4.

3.2 Searching for a Shortest Path

The second step in the algorithm is to find a shortest path in $\mathcal{G}$ between q_{init} and q_{goal}. We use the A^* algorithm [20], and a metric $\rho_{path} : \mathcal{C} \times \mathcal{C} \to [0, \infty)$ to measure the length of a path and the remaining distance

to q_{goal}. If the search procedure finds a path, we need to check it for collision. Otherwise, if no path exists in the roadmap, we either report failure, or go to the node enhancement step to add more nodes to the roadmap and start searching again. The choice is determined by the overall time allowed to solve the problem.

By defining ρ_{path}, we give preference to certain paths and reject others, i.e., decide which paths are of high quality and which paths are of poor quality. In this paper we focus on articulated robots and use the Euclidean configuration space $I_1 \times \cdots \times I_d$, where I_i is the range of joint i. The metric ρ_{path} is a weighted Euclidean metric, similar to (1), and the weights are equal to $\frac{1}{v_i}$, $i = 1, ..., d$, where v_i is the maximum angular velocity of joint i. This tends to give preference to paths with short execution time, which in many applications is the most interesting response variable.

3.3 Checking Paths for Collision

When the A^* algorithm has found a shortest path in the roadmap between q_{init} and q_{goal}, we need to check the nodes and edges along the path for collision. The edges are discretised and checked with a certain resolution, so our algorithm only requires a collision checker for points in $\mathcal{C}$; see [7, 23, 28].

The overall purpose of the Search, Check, and Remove steps of our algorithm (Figure 1), is roughly to identify and remove colliding nodes and edges of the roadmap until the shortest path between q_{init} and q_{goal} is feasible. Accordingly, when checking a path for collision, we are not primarily interested in verifying whether an individual node or edge is in $\mathcal{F}$ or not, but rather to remove colliding nodes and edges as efficiently as possible. Since a removal of a node implies all its connected edges to be removed, it seems reasonable to first check the feasibility of the nodes along the path, before checking the edges.

Checking Nodes Starting respectively with the first and the last node on the examined path and working toward the center, we alternately check the nodes along the path. As soon as a collision is found, we remove the corresponding node, and search for a new shortest path.

The reason for checking the nodes in this order is that the probability of having the shortest feasible path via a particular node is higher if the node is close to either q_{init} or q_{goal}, so we want to check end-nodes first; see [5] for a discussion.

Checking Edges If all nodes along the path are in $\mathcal{F}$, we start checking the edges in a similar fashion; working from the outside in. However, to minimize the risk of doing unnecessary collision checks, we first check all edges along the path with a coarse resolution, and then do stepwise refinements until the specified resolution is reached. As with the nodes, if a collision is found, we remove the corresponding edge, and search for a new shortest path. If no collision is found along the path, the algorithm terminates and returns the collision-free path.

To make this algorithm efficient, we of course record which nodes have been checked for collision, and to which resolution each edge has been checked, in order to avoid checking any point in $\mathcal{C}$ more than once.

The total number of collision checks depends on the resolution with which the edges along the path are checked. Again, since ρ_{coll} reflects the probability of collision, we determine the resolution with respect to this metric. Since the resolution should depend on the scale of $\mathcal{C}$ and the weights defining the metric, we introduce a parameter M_{coll}, specifying the number of collision checks required to check the longest possible straight line path in $\mathcal{C}$. In other words, assuming that $\mathcal{C}$ is a d-dimensional rectangle and q and q' are two opposite corners, the resolution – quantified by a step-size δ – is related to the length of the diagonal of $\mathcal{C}$ according to

$$\delta = \frac{\rho_{coll}(q, q')}{M_{coll}}.$$

3.4 Node Enhancement

If the search procedure fails, no feasible path between q_{init} and q_{goal} exists in the roadmap and new nodes are necessary in order to find one. In the node enhancement step, we generate N_{enh} new nodes, insert them to $\mathcal{G}$, and select neighbors in the same way as when $\mathcal{G}$ was initially built.

We may not only distribute the new nodes uniformly, but rather use the information available in the roadmap (or what is left of the roadmap), in order to distribute them in difficult regions of $\mathcal{C}$. In a method similar to the node enhancement in [17], we select a number of points in $\mathcal{G}$, called *seeds*, and then randomly distribute a new point close to each of them.

Although the seeds may help us identify difficult regions of $\mathcal{C}$, we still want to maintain a smooth distribution all over $\mathcal{C}$, because the knowledge about $\mathcal{C}$ is limited and we do not want to rely too much on the selection of seeds. In our algorithm, we let half of the enhancement nodes be uniformly distributed, and the rest distributed around seeds. This ensures *probabilistic completeness*, i.e., the probability of finding an existing path approaches 1 as time goes to infinity; see [5] for a proof.

Selecting Seeds The set of edges which have been removed from the roadmap and have at least one endpoint in $\mathcal{F}$, will certainly intersect the boundary of $\mathcal{F}$. Using the mid-points of these edges as seeds, may help us distribute points close to the boundary of $\mathcal{F}$.

However, if the enhancement step is executed several times, this may cause problems with clustering of nodes. Assume that we add a new node q. This node will give rise to a number of edges which in the next enhancement step may increase the probability of adding even more nodes close to q. Thus, the distribution of new enhancement nodes depends on the preceding enhancement steps, and may eventually cause undesired clusters of nodes. To avoid this phenomenon, we only use edges whose end-nodes are generated uniformly at random when selecting seeds.

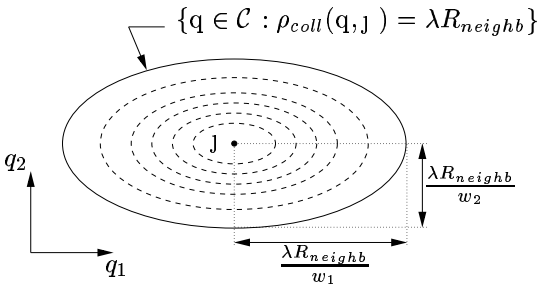

Figure 2: Example of a seed q_J in a 2-dimensional configuration space. If a new point q is distributed according to $N_d(q_J, \Sigma)$, with Σ as in (3), then q is distributed within the confidence ellipse (solid line) with probability $1-\alpha$. The dashed ellipses are contours of the distribution function. w_1 and w_2 are the weights defined in (1).

Distributing New Nodes Around Seeds When distributing a new point q around a seed q_J, we use the multivariate normal distribution. This distribution is smooth, easy to use, and allows us to control the distribution of q in terms of the metric ρ_{coll}. Hence, we can stretch the distribution in directions where the probabilities of making feasible connections are higher.

Introducing two parameters $\alpha \in (0,1)$ and $\lambda > 0$, we will show that we can choose a distribution such that

$$\rho_{coll}(q, q_J) \leq \lambda R_{neighb} \tag{2}$$

is an event with probability $1-\alpha$; see Figure 2. R_{neighb} is the maximum length of an edge defined in Section 3.1.

To achieve this property, we define a covariance matrix Σ as follows:

$$\Sigma = \frac{\lambda^2 R_{neighb}^2}{\chi_d^2(\alpha)} W^{-1}. \tag{3}$$

Here W is the same as in (1) and $\chi_d^2(\alpha)$ is the upper α percentile of a χ^2-distribution with d degrees of freedom (dof). Then we let the new point $q \sim N_d(q_J, \Sigma)$, i.e., q is multivariate normally distributed with d dof, mean q_J, and covariance matrix Σ. Since Σ is diagonal, this simply means that each component $q_i, i = 1, \ldots, d$, of q is normally distributed with mean η_i and variance $\Sigma_{i,i}$.

To show (2), we use that $(q - q_J)^T \Sigma^{-1} (q - q_J)$ is χ^2-distributed with d dof [16]. Thus, the event

$$(q - q_J)^T \Sigma^{-1} (q - q_J) \leq \chi_d^2(\alpha)$$

has probability $1-\alpha$. Using (1) and (3), gives the confidence ellipsoid in (2).

If we choose $\alpha = 0.05$, then the parameter λ controls the size of the 95% confidence ellipsoid relative to R_{neighb} (see Figure 2). In our experiments we found that $\lambda = 1$ is a good choice.

3.5 Multiple Queries

When the planner has found a collision-free path, it terminates and returns the path. Information about which nodes and edges have been checked for collision is stored in the roadmap, and as long as $\mathcal{C}$ remains the same, we use the same roadmap when processing subsequent queries. Thus, we benefit from the information already obtained. The new initial and final configurations are simply added to the roadmap, and the same algorithm, except for the initial generation of nodes, is run again.

As several queries are processed, more and more of the roadmap will be explored, and the planner will eventually find paths via nodes and edges which have already been checked for collision. This makes the planner efficient for multiple queries.

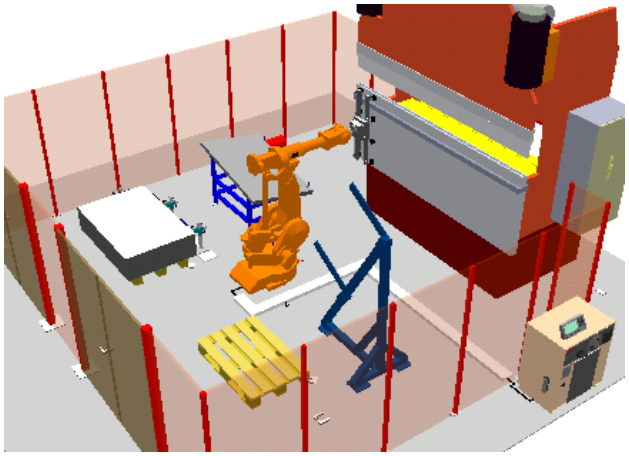

Figure 3: The workcell used in our experiments. The robot is in its home configuration denoted by A.

4 Experimental Results

In this section we present some performance tests of Lazy PRM when applied to a 6 dof robot in a realistic industrial environment. The planner has been implemented in C++ as a plug-in module to RobotStudio[1] – a simulation and off-line programming software running under Windows NT. The collision checks are handled internally in RobotStudio. The experiments have been run on a PC with a 400 MHz Pentium II processor and 512 MB RAM. In the tests we let $N_{init} = 10000$, $M_{neighb} = 60$, $M_{coll} = 200$, and $N_{enh} = 500$.

4.1 Path Planning Tasks

Our test example is a part of a manufacturing process in which an ABB 4400 robot is tending press breaking. In this particular case, plane sheets of metal are picked at a pallet, bent twice by the hydraulic press shown in Figure 3, and then placed at another pallet.

The process is divided into several steps, and our aim is to automatically plan the unconstrained paths of the robot. We let A to J denote ten different configurations shown in Figures 3 and 4. These are used as either initial or goal configurations in eight planning tasks, denoted for example $A \rightarrow B$, where A is the initial configuration and B is the goal configuration.

The scenario is as follows. Starting from the home configuration A, the robot picks a sheet of metal from the pallet at B (task $A \rightarrow B$), adjusts the grip at C (task $B \rightarrow C$), and puts the sheet-metal at the press

[1] RobotStudio is developed by ABB Digital Plant Technologies AB, Göteborg, Sweden.

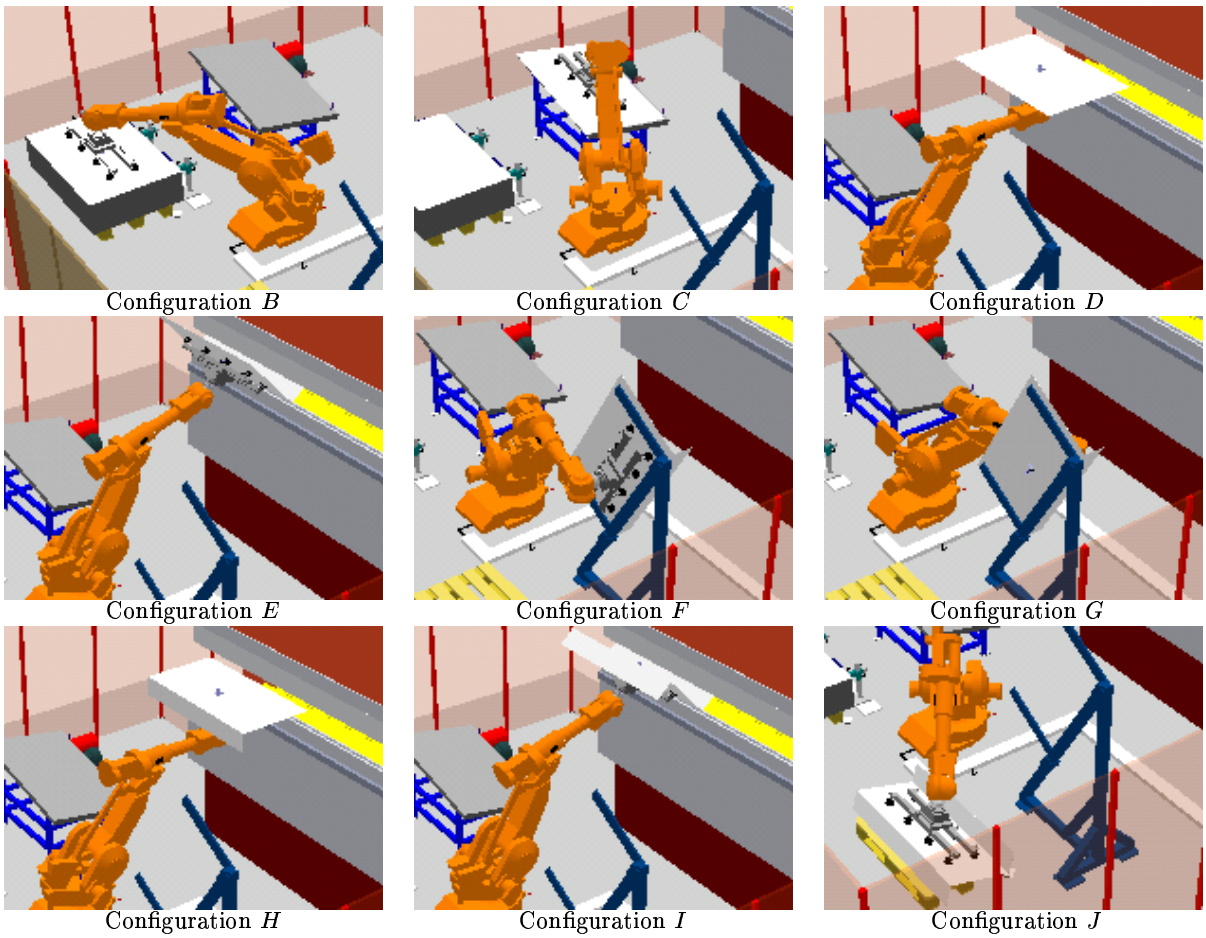

Figure 4: Configurations B to J used in the experiments.

D (task $C \rightarrow D$). After the breaking, the robot grasps the sheet-metal at E, moves to the re-gripper F (task $E \rightarrow F$), places the sheet-metal, moves to the other side G (task $F \rightarrow G$), grasps the sheet-metal, and moves back to the press H (task $G \rightarrow H$). After the second breaking, the sheet-metal is grasped at configuration I and placed at the pallet J (task $I \rightarrow J$). Then the robot returns to the home configuration A (task $J \rightarrow A$).

Thus, we have eight paths to plan. Note that during this series of steps, $\mathcal{C}$ changes several times. As soon as we grasp or place a sheet of metal, the collision-free part, $\mathcal{F}$, is changing. Neglecting the small displacement of the sheet-metal caused by the centering operation at C, the tasks $B \rightarrow C$ and $C \rightarrow D$ can be planned in the same configuration space. Accordingly, we have seven different configuration spaces in which to plan, and we have have to build one roadmap in each of them.

The reported results include the number of collision checks, the number of enhancement steps, and the planning time; the minimum, average, and maximum values, based on 20 consecutive runs for each task, are shown in Tables 1(a) - 1(c). The average number of collision checks performed on nodes and edges respectively are presented, as well as the average number of collision checks performed on the collision-free paths that the planner returned. The planning times in Table 1(c) are divided into graph building (including distance calculations and node and edge adding), graph searching, and collision checking.

In the last column of Table 1, the average values of the recorded data are summed up, thus indicating the average number of collision checks and average planning times for the entire press breaking operation.

In Table 1(d), we have for comparison reasons included some results obtained with a PRM-like algorithm. For each task, we simply checked *all* nodes and edges for collision in one of the roadmaps built by Lazy PRM in the initial step (see Figure 1). This corresponds to the learning phase without node enhancement in the original PRM [17]. Note that even with this long preprocessing, there is no guarantee that the planner will find a feasible path immediately. In Table 1(b), we see that several enhancement steps are needed with Lazy PRM, thus also needed here.

4.2 Interpretation of Results

We clearly see in Table 1(a) that collision checking represents the vast majority of the planning time (80%), but also that the graph building takes a lot of time (18%). Note that for the task $C \rightarrow D$, the same roadmap is used as for the task $B \rightarrow C$, making the graph building time significantly shorter. Interestingly, the time spent on graph searching is negligible, about 2%. Even if we carefully select the points to check

		Task								
		$A \to B$	$B \to C$	$C \to D$	$E \to F$	$F \to G$	$G \to H$	$I \to J$	$J \to A$	Total
Lazy PRM										
Collision checks										
for nodes	ave	9	41	172	263	129	134	320	18	1088(40%)
for edges	ave	83	125	273	235	283	158	361	124	1643(60%)
for returned path	ave	78	60	86	82	121	80	114	82	704(26%)
	min	74	45	143	154	175	135	139	81	
total	ave	92	166	445	499	412	293	682	142	2730
	max	131	463	701	1010	820	442	1290	299	

Table 1(a).

No. of enh. steps										
	min	0	0	0	0	0	0	0	0	
	ave	0	0.3	0.8	1.9	0	0.8	1.6	0	
	max	0	1	2	5	0	2	4	0	

Table 1(b).

Planning time (sec.)										
graph building	ave	6.6	6.7	0.8	8.3	6.5	7.3	8.2	6.6	51.0(18%)
graph searching	ave	0	0.1	0.5	1.3	0.9	0.4	3.0	0	6.3(2%)
coll. checking	ave	6.1	13.3	35.4	42.3	38.3	24.7	59.8	11.6	231.6(80%)
	min	11.2	9.7	10.8	19.7	22.1	16.8	17.8	13.0	
total	ave	12.7	20.2	36.8	52.0	45.7	32.5	71.0	18.2	289.0
	max	16.2	45.7	60.9	97.3	87.3	47.8	129.5	31.2	

Table 1(c).

PRM									
Collision checks									
for nodes	10000	10000	10000	10000	10000	10000	10000	10000	
of which in $\mathcal{F}$	4085	2942	2975	3047	3976	3038	3090	4121	
for edges	763063	409561	423443	451254	728012	447541	447000	787507	
total	773063	419561	433443	461254	738012	457541	457000	797507	
Planning time (sec.)	56625	31428	32299	35840	51774	35097	35200	56234	

Table 1(d).

Table 1: Performance data for Lazy PRM based on 20 consecutive runs for each task. Table 1(d) shows data for PRM based on one run for each task. The initial number of nodes, N_{init}, is 10000 in all tests.

for collision, and frequently search the roadmap for the shortest path, the total time spent on that is still very short.

Comparing the average number of collision checks performed by Lazy PRM (92 - 682) in Table 1(a) to the number of collision checks required to explore the entire initial roadmap (of order 500 000) in Table 1(d), we see that Lazy PRM only explores a small fraction, less than 0.1%, of the roadmap. We also see that a large percentage, 26%, of the total number of collision checks performed are actually done on the returned collision-free paths, and are therefore inevitable. This is the strength of the algorithm; by avoiding local planning and instead keeping a more global view, only the essential part of the roadmap is explored. As a consequence, very few edges – often only the edges along the final path – are checked with the finest resolution. This also makes the algorithm relatively insensitive to the resolution with which the paths are checked.

Table 1(b) reveals another important property of Lazy PRM and this is the ability to efficiently conclude that node enhancement is needed. Although several enhancement steps are needed in many cases, the number of collision checks in Table 1(a) is still very small compared with the total number of nodes in the roadmap. Thus, by only checking a small number of nodes (possibly also some edges), the algorithm can conclude that no feasible path exists in the roadmap, hence proceed to the node enhancement step.

5 Discussion

We have described a new probabilistically complete path planning algorithm which is particularly useful in high dimensional, relatively uncluttered configuration spaces, especially when collision checking is an expensive operation. Single queries are handled very quickly; indeed, no preprocessing is required. As subsequent queries are processed, the algorithm learns more about the configuration space since it automatically retains information obtained during previous queries. Thus, the planner works efficiently also for multiple queries.

The aim of Lazy PRM is essentially to minimize the number of collision checks while searching the shortest feasible path in a roadmap in the context of a PRM planner. This is done on the expense of frequent graph search. For a complex robot working in a complex workspace, like our 6 dof example, collision checking is a expensive operation, and careful selection of the points being checked for collision reduces the planning time considerably.

Lazy PRM has essentially one parameter that is critical for the performance – N_{init}, the initial number of nodes. N_{init} is strongly correlated to the probability of finding a feasible path without using the node enhancement step, and the optimal choice depends on the dimension of $\mathcal{C}$, the workspace, the planning task, and

the desired quality of the collision-free path. Our future work includes an investigation of the dependence between N_{init} and the planning time in different environments.

Probabilistic techniques, like Lazy PRM, often give very fast planning. However, in Table 1(c), we can see that the maximum planning time is approximately twice as long as the average planning time. New improved enhancement techniques are needed in order to make the algorithms more robust in the sense that the worst case performance is improved. This will also be a topic of our future research.

Acknowledgements The authors would like to thank ABB Digital Plant Technologies AB for initiating the project and for providing suitable software. Parts of this work was performed during the visit of Robert Bohlin to the Robotics Group at the Computer Science Department at Rice University. Robert Bohlin was supported by NUTEK, the Swedish National Board for Industrial and Technical Development, project P10499. Work on this paper by Lydia Kavraki has been supported in part by NSF CAREER Award IRI-970228 and NSF CISE SA1728-21122N. The authors would like to thank all members of the robotics group at Rice, Christian Nielsen and Jean-Claude Latombe for their comments.

References

[1] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1998.

[2] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 630–637. AK Peters, 1998.

[3] N.M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 113–120, 1996.

[4] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. of Rob. Research*, 10:628–649, 1991.

[5] R. Bohlin. *Motion Planning for Industrial Robots.* Licentiate thesis, Chalmers University of Technology, 1999.

[6] V. Boor, M.H. Overmars, and F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1018–1023, 1999.

[7] S. Cameron. Enhancing GJK: Computing minimum distance and penetration distanses between convex polyhedra. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 3112–3117, 1997.

[8] J.F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.

[9] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1718–1723, 1990.

[10] K. Gupta and A. P. del Pobil. *Practical Motion Planning in Robotics.* John Wiley, West Sussex, England, 1998.

[11] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom - random reflections at C-space obstacles. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 3318–3323, 1994.

[12] D. Hsu, L.E. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P. Agarwal, L. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 141–154. A K Peters, 1998.

[13] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 2719–2726, 1997.

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

[15] P. Isto. A two-level search algorithm for motion planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 2025–2031, 1997.

[16] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis.* Prentice Hall, New Jersey, 1998.

[17] L.E. Kavraki and J.C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 2138–2145, 1994.

[18] L.E. Kavraki, P. Švestka, J.C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. & Aut.*, 12:566–580, 1996.

[19] Y. Koga, K. Kondo, J. Kuffner, and J.C. Latombe. Planning motions with intentions. *Computer Graphics (SIGGRAPH'94)*, pages 395–408, 1994.

[20] J.C. Latombe. *Robot Motion Planning.* Kluwer, Boston, MA, 1991.

[21] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *Int. J. of Rob. Research*, 18(11):1119–1128, 1999.

[22] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 473–479, 1999.

[23] M.C. Lin and J.F. Canny. A fast algorithm for incremental distance computation. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1008–1014, 1991.

[24] E. Mazer, J.M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. of Art. Intelligence Research*, 9:295–316, 1998.

[25] C.L. Nielsen and L.E. Kavraki. A two level fuzzy PRM for manipulation planning. Technical Report TR2000-365, Rice University, 2000.

[26] M. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Utrecht University, the Netherlands, 1992.

[27] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K.Y. Goldberg, D. Halperin, J.C. Latombe, and R.H. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 19–37. A K Peters, 1995.

[28] F. Thomas and C. Torras. Interference detection between non-convex polyhedra revisited with a practical aim. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1994.

[29] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1024–1031, 1999.