Toward Real-Time Path Planning in Changing Environments

Peter Leven, *University of Illinois, Urbana, IL, USA*
Seth Hutchinson, *University of Illinois, Urbana, IL, USA*

We present a new method for generating collision-free paths for robots operating in changing environments. Our approach is closely related to recent probabilistic roadmap approaches. These planners use preprocessing and query stages, and are aimed at planning many times in the same environment. In contrast, our preprocessing stage creates a representation of the configuration space that can be easily modified in real time to account for changes in the environment. As with previous approaches, we begin by constructing a graph that represents a roadmap in the configuration space, but we do not construct this graph for a specific workspace. Instead, we construct the graph for an obstacle-free workspace, and encode the mapping from workspace cells to nodes and arcs in the graph. When the environment changes, this mapping is used to make the appropriate modifications to the graph, and plans can be generated by searching the modified graph.

After presenting the approach, we address a number of performance issues via extensive simulation results for robots with as many as twenty degrees of freedom. We evaluate memory requirements, preprocessing time, and the time to dynamically modify the graph and replan, all as a function of the number of degrees of freedom of the robot.

1 Introduction

In this paper, we present a new method for generating collision-free paths for robots operating in changing environments. Our work builds on recent methods that use probabilistic roadmap planners [3, 8, 12, 22, 25]. The idea that the cost of planning will be amortized over many planning episodes provides a justification for spending extensive amounts of time during a preprocessing stage, provided the resulting representation can be used to generate plans very quickly during a query stage. Thus, these planners use a two-stage approach. During a preprocessing stage, the planner generates a set of nodes that correspond to random configurations in the configuration space (hereafter, C-space), connects these nodes using a (simple, local) path planner to form a roadmap, and, if necessary, uses a subsequent sampling stage to enhance the roadmap. During the second, on-line stage, planning is reduced to query processing, in which the initial and final configurations are connected to the roadmap, and the augmented roadmap is searched for a feasible path.

Our new approach is a descendant of the probabilistic roadmap methods. Our goal, like theirs, is a real-time planner that uses approximate representations such as those provided by computer vision or range sensors. However, unlike probabilistic roadmap methods, our method is intended for robots that will operate in changing environments, and therefore we cannot exploit the premise that planning will occur many times in the same environment.

Our method begins, as do the probabilistic roadmap planners, by constructing a roadmap that represents the C-space. Nodes are generated by a random sampling scheme, and connections between nodes are generated using a simple, straight-line planner. Unlike the probabilistic roadmap planners, we generate a roadmap that corresponds to an obstacle-free environment. Then, in a second phase of the preprocessing stage, we generate a representation that encodes the mapping from cells in the discretized workspace to nodes and arcs in the roadmap. These two phases are specific to the robot, but are independent of the environment in which the robot will operate. The fact that the preprocessing is completely independent of the robot's target environment removes constraints on preprocessing time. Indeed, with our approach, it is feasible that when a new robot is designed, an extended period of preprocessing could be performed, at the end of which, the robot would essentially be preprogrammed to construct path plans in any environment that it might encounter.

In the on-line planning phase, the planner first identifies the cells in the discretized workspace that correspond to obstacles (e.g., by using a range scanner or stereo vision system), and then uses the encoded mapping to delete the corresponding nodes and arcs from the roadmap. Planning is then reduced to connecting the initial and final configurations to the roadmap (again, as is also the case with the probabilistic roadmap planners), and then searching the roadmap for a path between these newly added nodes. Of course it is possible to add obstacles to the environment in such a way that the roadmap becomes disconnected. This is true for any of the probabilistic roadmap planners (once one knows how samples are selected, and how these samples are connected by local planners, it is fairly straightforward to construct environments that will thwart them), but, as we will describe in subsequent sections, there are a number of steps that can be taken to cope with this problem, both at runtime and during the preprocessing stages.

In the on-line planning stage, our method runs in real time; plans are generated in less than one second. Thus, it is feasible to use the planner even in the case when obstacles are moving in the environment with unknown trajectories, provided a sensing system can identify in real-time those regions of the workspace that are occupied by the obstacles.

2 Related research

There has been much work on probabilistic roadmap planners, including planners for articulated robots [12, 22, 8], mobile robots in two dimensional environments [22], free-flying rigid objects in three dimensions [2], and flexible surfaces [7]. There are also versions of these planners that are geared toward single-shot motion planning [9, 15, 24]. Some modifications to the sampling methods have also been introduced: obstacle boundary [22, 3] and medial-axis [25].

The sample configurations generated during the construction of a probabilistic roadmap can be thought of as landmarks in the C-space. A family of approaches based on the Ariadne's Clew algorithm generate landmarks and search for paths between landmarks in a deterministic fashion by solving optimization problems. This method has been used to plan for manipulators [4, 18]. and for manipulation planning (i.e., constructing a sequence of transfer and transit paths) [1].

There have been several previous approaches to path planning in changing environments. In some cases, off-line planners have execution times that make it feasible to directly use them in some kinds of changing environments with no modifications. This is the case, e.g., for the Ariadne's Clew algorithm reported in [4, 18]. The Ariadne's Clew algorithm operates by generating landmarks (during an exploration phase) and then connecting them to the existing network (the search phase). Variations of this algorithm can be obtained by varying the search phase, and by using different optimization criteria to select candidate landmarks [1, 19]. The idea of incrementally expanding a network for single query planning has also been used in [9] and [15]. In both of these, networks are grown from both the initial and goal configurations until they can be connected. In [9] the notion of expansive C-spaces is used, while in the [15], random trees are used. In [24] an adaptable approach that uses multiple local planners is described. At runtime, characteristics of the problem are used to determine which (combination of) local planners will be most effective.

We also note here that in two of these previous approaches ([8] and [19]), the idea of somehow representing the mapping from the workspace to the C-space was incorporated. In [8], during the off-line planning stage, the planner is aware of a set of obstacles that might be present in the environment (in their experiments, a single obstacle was used). The locations of these obstacles are specified a priori, and at runtime the robot sensor system determines which, if any, of the obstacles are present in the environment. During the off-line planning stage, the trajectories in the paths tree that cause collision with each of these obstacles is determined. When objects are detected at runtime, the corresponding arcs are deleted from the graph. In [19], the paths tree (created by the modified Ariadne's Clew algorithm) is augmented to generate a graph. Then, for each path in the graph, the corresponding workspace cells are identified. Thus, when a new obstacle is added to the environment, the set of paths that intersect that obstacle can be deleted from the graph. Because these authors are primarily interested in domains for which extensive preprocessing is not viable, they construct relatively small graphs (fewer than 100 landmarks, with fewer than 400 corresponding paths). Therefore, it is fairly easy to add obstacles that would disconnect the graph, even though these obstacles might not cause the free C-space to become disconnected. In their experimental evaluation of their planner, a single obstacle was added, and the addition of this obstacle resulted in disconnecting the graph into five components. In such cases, their algorithm resorts to the Ariadne's algorithm to reconnect the graph, which can take time long enough to prohibit the planner from being used in environments with moving obstacles.

We believe that our planner is the first planner that constructs a representation that can be used to replan in real time in changing environments. In the remainder of the paper we describe the specific details of the algorithm, give extensive evaluation of our methods via simulation results, and discuss the current trajectory of our research efforts.

3 Constructing the Roadmap

Our construction of a roadmap of the C-space is very similar to methods used in previous probabilistic roadmap planers [12]. Nodes are generated by generating sample configurations, and these nodes are then connected to form a graph. We will denote this graph by $\mathcal{G} = \langle \mathcal{G}_n, \mathcal{G}_a \rangle$, in which $\mathcal{G}_n$ is the set of nodes in the graph, and by $\mathcal{G}_a$ the set of arcs in the graph.

Since the graph is constructed without the presence of obstacles in the workspace, local planning to connect the nodes is trivial. Generating samples is potentially more complex than for the traditional probabilistic roadmap planners, since we cannot exploit the geometry of the C-space obstacle region to guide the sampling. We now discuss sampling the C-space, and generating arcs to connect the resulting nodes.

3.1 Generating Sample Configurations

Since there are no obstacles to consider, it is fairly easy to generate samples in the C-space. The only hard constraint is that self-collision (i.e., collision between distinct links of the robot) is prohibited. At present, we generate configuration nodes by sampling from a uniform distribution on the C-space. This approach reflects a complete absence of prior knowledge about the environment in which the robot will ultimately operate. If prior knowledge, either about the environment or the set of tasks that the robot will perform, were available, an appropriate importance sampling scheme, or even a deterministic scheme (if the existence of certain obstacles were known in advance) could be used. More effective schemes for generating sample configurations are discussed in Section 6.

3.2 Connecting the nodes

In order to create a graph, pairs of nodes are connected by arcs that correspond to trajectories of the arm. Like several of the previously mentioned approaches, we connect each node to its k-nearest neighbors. In our current implementation, the paths corresponding to arcs are not tested to ensure that no self-collisions take place along the path.

The distance metric used for determining the neighbors of a node plays a very important role in the probabilistic roadmap methods [2]. For these planners, good distance metrics provide a set of neighbors to which the local planner has a good likelihood of generating a collision free path. In addition, the distance metric must be fast to compute, as it will be called many times to evaluate pairs of nodes (this is particularly true for the single query planners).

For our application, the swept volume of a path in the workspace connecting two configurations would be an ideal metric. Unfortunately, as noted by others [2, 13], this metric is very expensive to compute, and therefore, in our current implementation, we have opted for approximations that are faster to compute. In [2], a number of metrics were evaluated for efficiency and effectiveness for the case of a rigid object translating and rotating in a three-dimensional workspace. In particular, as we will discuss below, we have performed extensive experimental evaluation with the following four distance metrics:

the 2-norm in C-space

$$\mathcal{D}_2^{\mathcal{C}}(\mathrm{q},\mathrm{q}') = \|\mathrm{q}' - \mathrm{q}\| = \left[\sum_{i=1}^{n}(q_i' - q_i)^2\right]^{\frac{1}{2}}$$

the ∞-norm in C-space

$$\mathcal{D}_\infty^{\mathcal{C}}(\mathrm{q},\mathrm{q}') = \max_n |q_i' - q_i|$$

the 2-norm in workspace

$$\mathcal{D}_2^{\mathcal{W}}(\mathrm{q},\mathrm{q}') = \left[\sum_{\mathrm{p}\in\mathcal{A}} \left\|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\right\|^2\right]^{\frac{1}{2}}$$

the ∞-norm in workspace

$$\mathcal{D}_\infty^{\mathcal{W}}(\mathrm{q},\mathrm{q}') = \max_{\mathrm{p}\in\mathcal{A}} \left\|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\right\|.$$

In each of these equations, the robot has n joints, q and q$'$ are the two configurations corresponding to different nodes in the graph, q_i refers to the configuration of the i-th joint, and p(q) refers to the workspace reference point p of the set of reference points of the robot $\mathcal{A}$ at configuration q. Versions of $\mathcal{D}_\infty^{\mathcal{W}}$ and $\mathcal{D}_2^{\mathcal{W}}$ were also used in [13].

Once a node's k-nearest neighbors have been identified, a local planner is used to connect the corresponding configurations. As a motion planner, this local planner in the preprocessing phase has very modest requirements. It should be reasonably fast, as that reduces the time needed to construct the data structures, but this is not a primary concern. It must be deterministic, i.e., it must always return the same path when given the same two nodes as input. This is required since our approach will use the volume swept by the robot when it traverses this path. If the path changed each time, we would be unable to guarantee that the path was not blocked during on-line planning. In addition, the local planner should consider self-collisions of the robot when determining whether two nodes can be connected (although we have not yet implemented this feature).

We also note here that, during the on-line planning phase, a second local planner will be used to connect the initial and final configurations to the roadmap. In contrast to the planner used to connect nodes in the graph, this planner does have to consider the obstacles around the robot. Since this problem is faced by all of the probabilistic roadmap planners, we plan to adapt existing methods for this problem (e.g., the technique presented in [24] may be used, which selects the local planner based on an evaluation of which planner is most likely to succeed).

4 Workspace to C-space mapping

The ability to plan in real time in changing environments comes from our encoding of the relationship between the workspace and the C-space. To represent the workspace, we use a uniform, rectangular decomposition, which we denote by $\mathcal{W}$. We denote the C-space by $\mathcal{C}$, and define the mapping $\phi : \mathcal{W} \to \mathcal{C}$ as

$$\phi(w) = \{\mathrm{q} \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset\},$$

in which w is cell of the workspace and $\mathcal{A}(\mathrm{q})$ denotes that subset of $\mathcal{W}$ occupied by the robot at configuration q. We note that $\phi(w)$ is exactly the C-space obstacle region (often denoted by $\mathcal{CB}$) if w is considered as a polyhedral obstacle in the workspace. In our approach, we do not explicitly represent the C-space, but instead use the roadmap $\mathcal{G}$. Therefore, we define two additional mappings, one from the workspace to the nodes in the graph, and one from the workspace to the arcs in the graph:

$$\phi_n(w) = \{\mathrm{q} \in \mathcal{G}_n \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset\},$$
$$\phi_a(w) = \{\gamma \in \mathcal{G}_a \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset \text{ for some } \mathrm{q} \in \gamma\}.$$

4.1 Computation of ϕ_a and ϕ_n

In terms of the implementation, it is much easier to compute the inverse maps ϕ_n^{-1} and ϕ_a^{-1}. Therefore, we use the inverse maps for the construction of our representation of the mapping.

The construction of the representation for ϕ_n^{-1} is straightforward. For each $\mathrm{q} \in \mathcal{G}_n$ we note the mapping from each $w \in \phi_n^{-1}(\mathrm{q})$ to the corresponding q. The set of cells in $\phi_n^{-1}(\mathrm{q})$ is computed by expanding a "seed" cell in a set of shells surrounding the seed. The seed for fixed-base articulated robots is the origin of the robot, which is not stored as part of ϕ_n. The shell expands in each direction in the workspace until the collision tester determines that a cell is outside the robot. We use the collision-checking package V-Clip [20] to test for cell intersection with the robot.

The computation of ϕ_a^{-1} is more complex and time-consuming. It involves computing the swept-volume of the robot as it traverses a path computed by the local planner between two configurations. In our implementation, cells that are occupied by the robot at the endpoints of the arc γ (i.e., the configurations that correspond to the two nodes connected by the arc) are not included in $\phi_a^{-1}(\gamma)$. Computing the swept volume for a robot trajectory is not a trivial problem. A method for swept volume computation for three dimensional objects that can only translate is presented in [26]. An extension to this method was presented that could also accommodate rotation about a single axis, but no results were given. This approach could be used to simplify the computation of ϕ_a^{-1}.

We have developed a method for computing $\phi_a^{-1}(\gamma)$ as follows. First, ϕ_n^{-1} for the two endpoints of γ is computed. Then, the path corresponding to γ is sampled using a recursive bisection method, which proceeds as follows. First, the configuration corresponding to the midpoint of the segment connecting the two nodes is computed, and ϕ_n^{-1} for that configuration is computed. If this set contains any cells not already in ϕ_n^{-1} of the endpoints, these new cells are added to the swept volume, and the path is subdivided again on both sides of the midpoint. This process is repeated until no new cells of $\mathcal{W}$ are added.

4.2 Efficient representations of ϕ

Even though computer memories are rapidly growing, it is beneficial to compress the representations of ϕ_n and ϕ_a, provided that this can be done without drastically increasing the computation required compute plans online. Reducing the size of the representation will also enable us to consider larger graphs $\mathcal{G}$, which will increase the efficacy of the online planning.

From an information theoretic viewpoint, compression of a data set involves the reduction of redundancy in that data set. The amount of compression that can be performed is limited by the information content of the data set, which, in turn, is related to the degree of unexpectedness, or randomness, in the data set [5]. There are three sources of redundancy in the representation of ϕ_n and ϕ_a that can be exploited: 1) the spatial coherence of the set $\phi(w)$ in $\mathcal{C}$, 2) spatial coherence of $\phi(w)$ for neighboring w's in $\mathcal{W}$, and 3) the representation of the labels of the nodes and arcs in the graph. The third source provides only limited efficiency increases, as much as a factor of two for our experiments, depending on the size of the graph.

The spatial coherence of ϕ_a and ϕ_n is based on the following ideas. Since the robot is connected, and since w is connected and compact, then $\phi(w)$ is connected and compact. This follows from the application of well-known facts about the C-space obstacles [16], and that $\phi(w)$ is exactly the C-space obstacle region if w is considered as a polyhedral obstacle in the workspace.

The spatial coherence of $\mathcal{CB}$ has been exploited in previous collision checking approaches (e.g., [17, 20]). In our case, spatial coherence derives from the continuity of ϕ, namely, that small changes to w will cause only small changes to $\phi(w)$. Because of this, for some cell, say $w^* \in \mathcal{W}$, we expect that $\phi(w)$ will be very similar to $\phi(w^*)$ for $w \in \eta(w^*)$, with $\eta(w^*)$ some appropriate neighborhood of w^*. This spatial coherence presents a situation that is somewhat analogous to the situation confronted in video compression: in a stream of images, there will be only small variations between most adjacent images in the sequence. This is one of the premises for many modern video compression methods (e.g., MPEG [21]). Unfortunately, we cannot directly apply video compression techniques, since these techniques generally employ lossy compression (i.e., the original image sequence cannot be exactly reconstructed), and in our case this could lead to collisions.

The discussion above suggests the following approach: partition $\mathcal{W}$ into a set of neighborhoods, and, for each neighborhood (a) choose a representative w^*, (b) derive a compact representation of $\phi(w^*)$ and (c) for all $w \in \eta(w^*)$, express $\phi(w)$ in terms of $\phi(w^*)$. We postpone the discussion on step (b) to Section 6.3. In some cases, we may be able to improve upon this by selecting some other reference set in step (c), and we discuss this below.

Given the above, we can formulate the corresponding optimization problem. For specific choice of neighborhood system we have the cost functional

$$\mathcal{L}(\mathcal{W}^*, \eta) = \sum_{w^* \in \mathcal{W}^*} \left\{ \text{cost}[\phi(w^*)] + \sum_{w \in \eta(w^*)} \text{cost}[\phi(w)] \right\}, \quad (1)$$

in which $\mathcal{W}^*$ is a set of representative cells in $\mathcal{W}$, $\eta(w^*)$ is the set of neighbor cells for w^*, and cost$[\phi(w)]$ denotes the cost of encoding the representation. This leads to the the optimization problem

$$\left\{ \begin{array}{ll} \text{minimize} & \mathcal{L}(\mathcal{W}^*, \eta) \\ \text{subject to:} & \bigcup_{w^* \in \mathcal{W}^*} \eta(w^*) = \mathcal{W} \text{ and} \\ & \eta(w_i^*) \cap \eta(w_j^*) = \emptyset, i \neq j. \end{array} \right. \quad (2)$$

This particular formulation of the cost suggests an algorithm that first selects representatives in $\mathcal{W}$ and then builds the appropriate neighborhoods. The encoding of the neighborhoods is based on the idea discussed above that there will be only minor variations in ϕ over local neighborhoods of $\mathcal{W}$. Assuming for the moment that the neighborhoods have been determined, one way to encode ϕ over a neighborhood $\eta(w^*)$ is to first determine a representative $\phi^*(w^*)$ and to then specify $\phi(w)$ relative to $\phi^*(w^*)$ for each $w \in \eta(w^*)$. This is essentially a differential encoding, and can be specified as $\phi'(w) = \phi(w) \oplus \phi^*(w^*)$, where $\phi'(w)$ is the encoded representation of $\phi(w)$ and $\oplus$ is the set symmetric difference operator.

There are two methods for determining $\phi^*(w^*)$. The first of these is to use $\phi(w^*)$ itself, i.e., $\phi^*(w^*) = \phi(w^*)$. The second is to compute $\phi^*(w^*)$ as the set that minimizes

$$\sum_{w \in \eta(w^*)} |\phi'(w)|,$$

where $|\cdot|$ denotes set cardinality. The first method may lead to a more efficient representation of $\phi^*(w^*)$ itself, while the second minimizes $|\phi'(w)|$ for each $w \in \eta(w^*)$. This is a tradeoff that will have to be evaluated when selecting which set to use for $\phi^*(w^*)$.

There is another possible differential encoding scheme that can also be used to take advantage of the spatial coherence of $\phi(w)$ over $\mathcal{W}$. This approach is similar to above, except that instead of choosing one $\phi^*(w^*)$ over $\eta(w^*)$, the ϕ^* is chosen individually for each $w \in \eta(w^*)$ from the set of cells adjacent to w that are closer to w^*. In this approach, $\phi(w^*)$ is encoded by itself, the cells adjacent to w^* are encoded using $\phi(w^*)$ as their reference, and for each of the other $w \in \eta(w^*)$, the neighbor w' to w that results in the smallest $|\phi'(w)|$ is chosen, with $\phi'(w) = \phi(w) \oplus \phi(w')$.

These two approaches may be combined in a hybrid approach, such that the neighborhood $\eta(w^*)$ is composed of two parts $\eta_1(w^*)$ encoded using the first scheme above and $\eta_2(w^*)$ encoded using the second scheme. The cell w^* is encoded as part of $\eta_1(w^*)$, and each $w \in \eta_2(w^*)$ is encoded with respect to the adjacent cell that is closest to a member of $\eta_1(w^*)$.

Still to be addressed is the problem of selecting the best set of representative cells in the workspace and the problem of choosing the best neighborhoods $\eta_1(w^*)$ and $\eta_2(w^*)$ for each $w^* \in \mathcal{W}^*$ that satisfies (2). This is a difficult combinatoric optimization problem, for which we have applied the following greedy expansion algorithm. The elements of $\mathcal{W}$ are placed into a priority queue, $\mathcal{Q}$ in decreasing order of the size of the representation of $\phi(w)$ (i.e., $|\phi_n(w)| + |\phi_a(w)|$). Let w^* be the first element in $\mathcal{Q}$. Initialize $\eta(w^*) = \{w^*\}$. Then, until the cost of adding an additional neighbor to $\eta(w^*)$ exceeds the cost of directly encoding $\phi(w)$ or a size threshold is reached, expand the neighborhood, deleting the added w from $\mathcal{Q}$. In the hybrid approach, each cell w is placed in the neighborhood η_1 or η_2, depending on which minimizes the cost of its representation. This is repeated for the new head of $\mathcal{Q}$ until the neighborhoods form a partition of the workspace.

Note that this operation is performed separately for both ϕ_n and ϕ_a.

5 Empirical evaluation

For a real-time path planner, we are interested in three parameters: graph update time, planning time, and data structure size. We are also interested in the preprocessing time, though this is less important. In this section, we evaluate a preliminary version of the planner, studying the case of serial-link manipulators with revolute joints operating in a two-dimensional workspace. Note that our implementation is not limited to revolute joints; prismatic joints can be used as well. We limit the following discussion to revolute joints to simplify the analysis.

5.1 Experimental set-up

The following experimental set-up was used to evaluate the approach. The robots tested are all serial-link manipulators with two to 20 revolute joints in a two-dimensional workspace. The workspace cells and the robot are given three-dimensional polyhedral descriptions to allow the use of standard collision checking packages for collision testing. Each cell of the workspace is modeled as a unit cube, and each link of the robot is modeled as a rectanguloid with a width of 2.1 units and a height of 1 unit. The total length of the robot was fixed at 70 units, with each link length being scaled appropriately. Each joint of the robot was given the full revolute range of motion, and the joints were allowed to wrap around. In addition, the robot was not tested for self-collision, except when the samples of C-space were generated. To evaluate how the data structures and computation times grow with the

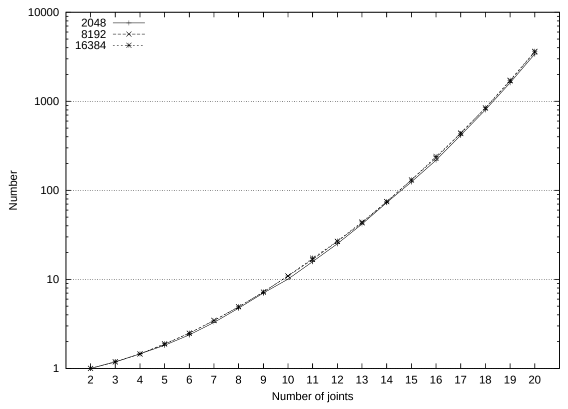


Figure 1: *Number of samples of C-space generated per node accepted.*

graphs, we tested graphs with 2048, 8192, and 16384 nodes.

5.2 Results for serial-link manipulators in a 2D workspace

In this section we present some results from using the above setup. We start with an analysis of the data structures that are computed to implement our planning approach. Each of the four data structures—graph nodes, graph arcs, ϕ_n, and ϕ_a—was computed and stored separately to simplify analysis. We follow the analysis with some compression results for ϕ_n and ϕ_a, and we conclude with some planning examples.

5.2.1 C-space sampling

The C-space for each of the robots was sampled uniformly at random, except for the first 129 samples. The first 129 samples were chosen such that the robot covers as much of the workspace as possible. For the robots tested, this means that 129 uniformly-spaced samples were taken of the range of the first joint, and the remaining joints were held fixed at a position that maximized the length of the robot. Of the remaining random samples, samples in which the robot collides with itself were rejected.

It is interesting to note that, when sampling the C-space of the robot uniformly, it became exponentially more difficult as the number of joints increased to find configurations in which the robot did not collide with itself. This is a consequence of a robot consisting of revolute joints operating in the plane: for each set of three links, there is a region of the C-space in which the third link is in collision with the first. For each additional joint beyond three, there is a combinatoric effect with combinations of links in collision, as well as an interaction among these links that tends to make the collision regions larger. For a non-planar robot operating in 3D, this effect is expected to be drastically reduced.

The exponential sampling effect can be seen in Figure 1, which shows the number of samples of C-space generated per node accepted. Notice that the y-axis scale of this graph is logarithmic. As expected, the number of samples generated per node accepted does not change as the desired number of nodes increases.

The size of the data structure for the configurations associated with the nodes in the graph is $8mn + 12$ bytes, where m is the number of nodes and n is the number of joints.

5.2.2 Graph construction

As described above, arcs in the graph are constructed for the k-nearest neighbors of each node. For our evaluation, we tested the case of $k = 5$ nearest neighbors. We computed the nearest neighbors using the simple $O(n^2)$ method of calculating the distance between all pairs of nodes and keeping the closest five for each node. The paths connecting these neighbors are not checked for feasibility, i.e., whether the robot avoids self-collision while following the paths.

The computation times for computing the graph are shown in Figure 2. The increase in computation time with the number of joints reflects the increase in the cost of computing the metric. The sharp rise in the computation time after 16 joints seen in the graph for the $\mathcal{D}^{\mathbb{C}}_{\infty}$ and $\mathcal{D}^{\mathbb{C}}_{2}$ distance metrics is unexpected; this may be due to a memory effect in the implementation, where the amount memory used for the calculation crosses some system threshold.

What also can be seen in Figure 2 is that there is not much difference in computation time between the 2-norm and the ∞-norm, except that the computations involving the ∞-norm tend to take somewhat more more time to perform (this is likely an artifact of the processor architecture, in which the cost of a branch is greater than the cost of a floating-point multiplication). A greater difference can be seen between the metric in C-space and the metric in the workspace, where the

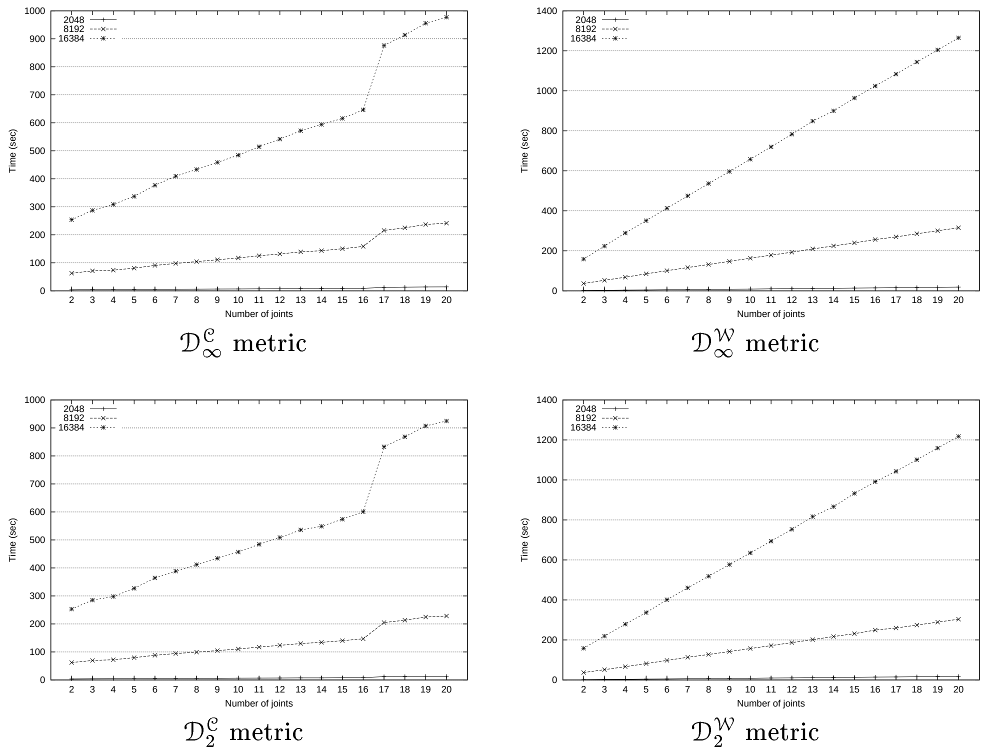


Figure 2: *Time in seconds to compute the k-nearest neighbors for the different distance metrics.*

latter appears to have a more linear increase in computation time as the number of joints increases. Overall, for robots with more than five joints, the workspace metric takes more time to compute.

The size of the graph varies linearly with the number of nodes, with an average size of 163kB for 2048 nodes, 654kB for 8192 nodes, and 1309kB for 16384 nodes. The variance in size from the average was less than 15%.

5.2.3 Computing ϕ_n

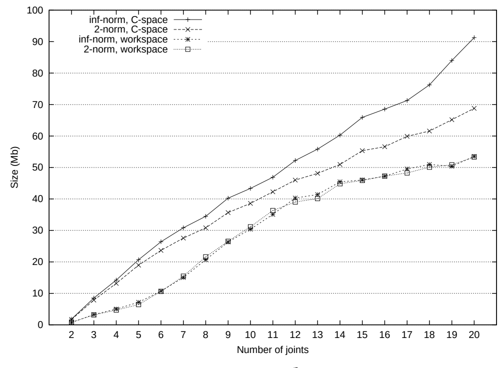
The overall size of ϕ_n is the product of the number of nodes and the average number of cells the robot covers in the workspace. For a robot with a fixed maximum length, this means that the size of ϕ_n is largely independent of the number of joints. For our experiments, the size of ϕ_n per node in $\mathcal{G}_n$ averages around 1000 bytes.

There is a sub-linear increase in the size of ϕ_n as the number of nodes increases that is a result of the overhead for maintaining the data structure being amortized over more nodes. The overhead for the data structure depends on the number of cells in the workspace that intersect with the robot at any node in the graph. This overhead does not depend on the number of nodes in the graph, as long as the graph contains nodes that span the reach of the robot in the workspace (i.e., the graph contains nodes that touch all the cells in the workspace that the robot can reach).

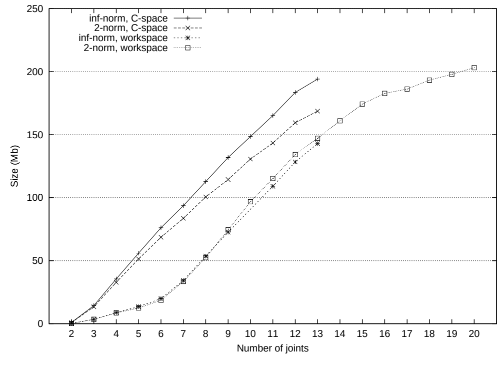
The time required per node to compute the node obstacle data structure increases linearly as the number of joints increases. This is expected because the size of the geometric description of the robot used for collision testing increases with the number of joints.

5.2.4 Computing ϕ_a

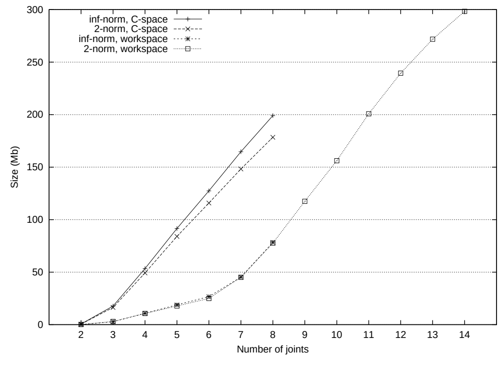
The size of ϕ_a for different numbers of nodes is shown in Figure 3. As can be seen in the figure, ϕ_a can become quite large. This is the reason that the graphs stop at 13 joints for the 8192 node case, and 8 joints for the 16384 case. The limit in this case was the process



2048 nodes



8192 nodes



16384 nodes

Figure 3: *Size of ϕ_a.*

size of the program computing ϕ_a; in order to keep the computation times comparable, the memory size of the program computing ϕ_a was limited to the physical memory available in the computer. More examples were computed for the $\mathcal{D}_2^{\mathcal{W}}$ metric for the 8192 and 16384 node cases in order to evaluate the compressibility of the resulting data files (see Section 5.2.5).

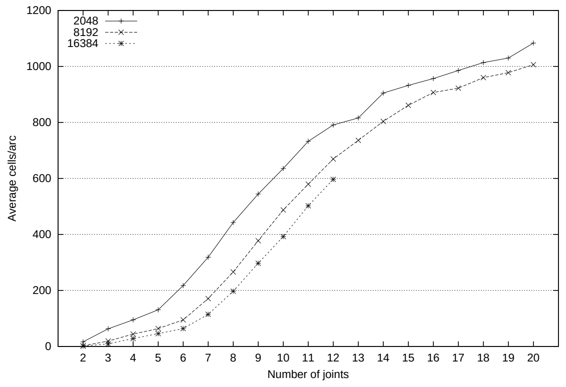
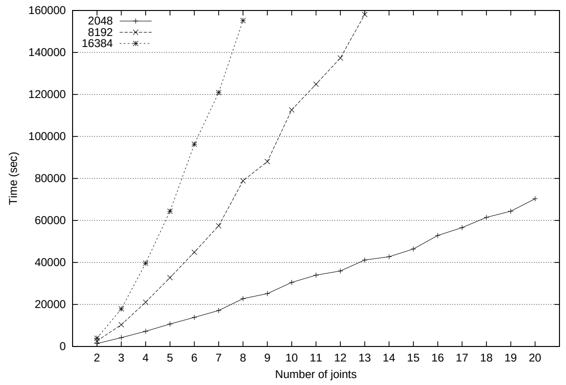


Figure 4: *Average of $|\phi_a^{-1}(\gamma)|$ per γ for $\mathcal{G}_a$ generated from the $\mathcal{D}_2^{\mathcal{W}}$ metric.*

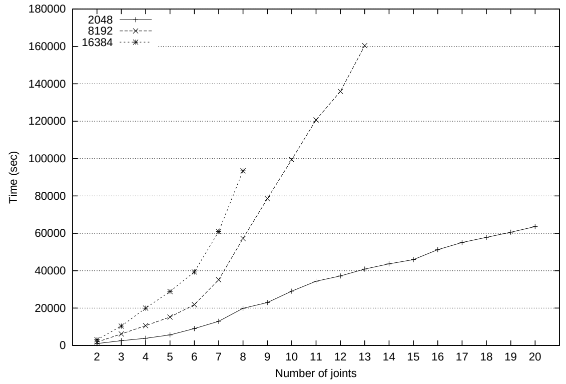
It is also apparent that the $\mathcal{D}_\infty^{\mathcal{C}}$ performs the worst in terms of the size of ϕ_a. The $\mathcal{D}_2^{\mathcal{C}}$ is the next worst, and the $\mathcal{D}_\infty^{\mathcal{W}}$ and $\mathcal{D}_2^{\mathcal{W}}$ produce the best results. It is expected that the two metrics defined on the workspace would produce better results than the C-space metrics, since the workspace metrics penalize the motion in the workspace more than do the C-space metrics. There is no apparent advantage to $\mathcal{D}_2^{\mathcal{W}}$ over $\mathcal{D}_\infty^{\mathcal{W}}$.

Another interesting point in Figure 3 is the apparent inflection point at six joints in the graphs for the size of ϕ_a when computed using the $\mathcal{D}_\infty^{\mathcal{W}}$ and $\mathcal{D}_2^{\mathcal{W}}$ metrics. This inflection point is more obvious in the 8192 and 16384 node graphs. An explanation for this is that, given the resolution of the workspace that we are using, the random sampling is better able to “fill” the C-space with samples for the robot with fewer joints. This results in the nodes being closer together, and, therefore, the paths between them do not cover as much of the workspace. This effect can also be seen in Figure 4, which shows the average size of $\phi_a^{-1}(\gamma)$ per $\gamma \in \mathcal{G}_a$ for the $\mathcal{D}_2^{\mathcal{W}}$ metric ($\mathcal{D}_\infty^{\mathcal{W}}$ produces similar results).

Shown in Figure 5 is the time it takes to compute ϕ_a for the $\mathcal{D}_2^{\mathcal{C}}$ and $\mathcal{D}_2^{\mathcal{W}}$ metrics (the $\mathcal{D}_\infty^{\mathcal{C}}$ and the $\mathcal{D}_\infty^{\mathcal{W}}$ produce similar results to the $\mathcal{D}_2^{\mathcal{C}}$ and $\mathcal{D}_2^{\mathcal{W}}$ metrics, respectively). These times correlate well with the increase in the size of ϕ_a: the larger ϕ_a is, the longer it takes to compute. Notice that the graph for the computation times for the workspace metric shows an inflection point like the graphs for the size of ϕ_a.

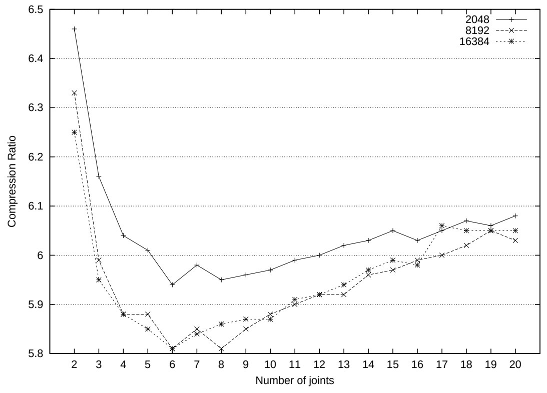


$\mathcal{D}_2^{\mathcal{C}}$ metric

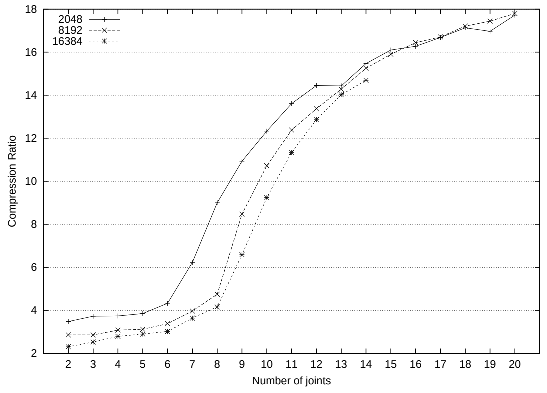


$\mathcal{D}_2^{\mathcal{W}}$ metric

Figure 5: *Computation time for ϕ_a for two metrics.*



(b) ϕ_n



(b) ϕ_a

Figure 6: *Compression ratios.*

5.2.5 Some compression results

Shown in Figure 6 are the compression ratios for ϕ_n and ϕ_a that were achieved by differential encoding of the cells of the workspace and by using a more efficient encoding of the labels used to represent the nodes and arcs in the graph (roughly a factor of two of the compression ratio comes from this change in representation). The compression ratios are relatively flat for ϕ_n as the number of joints changes.

To gain some insight into the behavior of the compression ratios achieved for ϕ_n, a set of images showing the distribution of $|\phi_n(w)|$ over the workspace were generated. Figure 7 shows an example image, showing the two-link robot case with a graph of 2048 nodes. In the figure, the grid lines demark 11x11 collections of workspace cells, and the darkness of a cell is proportional to the logarithm of $|\phi_n(w)|$. Cells that are white have $|\phi_n(w)| = 0$. In the figure, the reach of the first link of the robot is readily apparent as a darker disk inside a lighter disk. Some variation in darkness can be seen in the lighter regions.

As the number of joints in the robot increases, the ring corresponding to the first link becomes smaller, and the links themselves have a greater tendency to cluster near the origin due to the uniform random sampling. This clustering also means that clearly-visible rings corresponding to links of the robot other than the first do not appear. The clustering also has an effect on the compressibility of the ϕ_n. At two joints, the distribution of $|\phi_n(w)|$ is relatively uniform, compared to robots with more joints. A greater uniformity of coverage of the workspace implies greater spatial coherence between neighboring cells and that allows for

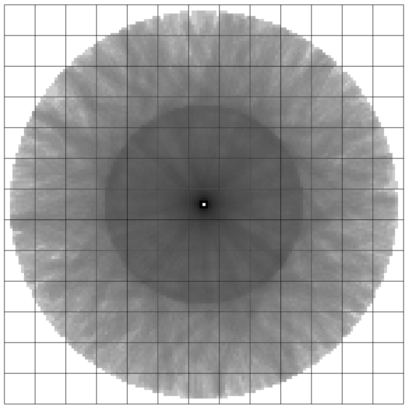


Figure 7: *$|\phi_n(w)|$ per cell for a two-link robot and a graph with 2048 nodes.*

greater compression. At the 20 joints, the distribution tends to be highly clustered near the origin, allowing for higher compression as the cells outside the cluster near the origin are grouped together in larger groups. In between the two extremes, the compression is lower.

On the other hand, the compression ratios for ϕ_a increase with the number of joints. This is expected because the average length of the paths corresponding to arcs in the graph tends to increase with the number of joints, and that in turn leads to an increase in the size of the volume of the workspace swept by the robot as it traverses the paths. As more cells are affected by the longer paths, the spatial coherence of the cells in the workspace increases. Notice that the inflection point seen around eight joints in this graph is similar to that seen in Figure 3. The same effect that minimizes the size of ϕ_a also reduces spatial coherence, and, therefore, the achievable compression ratio.

Shown in Figure 8 is the distribution of $|\phi_a(w)|$ per cell for the two link robot and the 2048-node and 16384-node graphs. These graphs have the same resolution as Figure 7. The sparseness of the distribution of the arcs in this case is the reason for the reduction in the compression ratio available using the differential encoding. Also shown in the figure is the effect of increasing the size of the graph from 2048 nodes to 16384 nodes. As noted earlier, the average distance between nodes decreases, decreasing the average swept volume of the paths corresponding to the arcs in the graph, and, in this case, dramatically increasing the sparseness of ϕ_a.

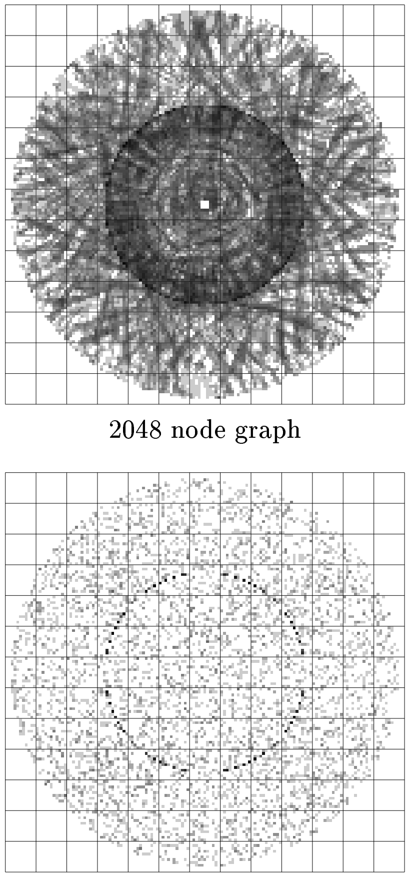


Figure 8: *$|\phi_a(w)|$ per cell for a two-link robot.*

5.2.6 Some planning examples

The current implementation of our path planner tests the connectivity of the graph between two selected nodes in the modified graph; it does not connect arbitrary configurations to the graph. In all of the planning runs tested using the uncompressed representations of ϕ_n and ϕ_a, it takes less than one second to update the node and arc obstacle data structures and to find a path in the modified graph. Extensive tests using the compressed representations have not yet been performed.

Shown in Figure 9 is a 19-joint robot negotiating between two obstacles (black objects in the figure). The steps shown in the figure correspond to the nodes in the graph along the planned path. This plan was generated from the graph that used the $\mathcal{D}_2^{\mathcal{W}}$ distance metric with

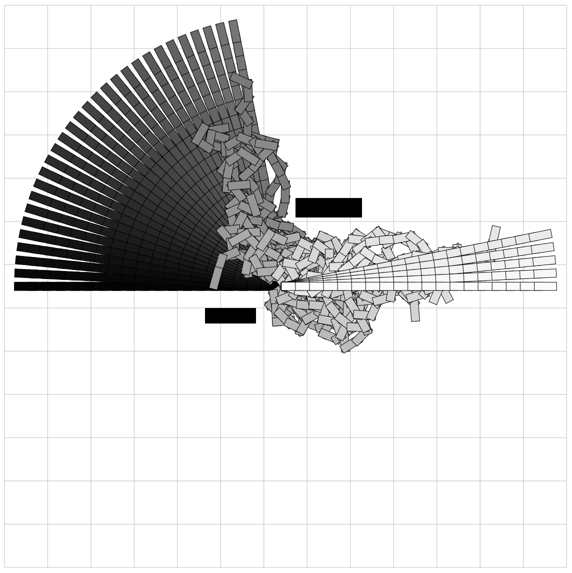


Figure 9: *A plan for a 19-joint robot passing through a (relatively) narrow corridor. The dark blocks are the obstacles.*

2048 nodes. The fan-like parts of the plan come from the set of nodes created to span the workspace of the robot, as described in Section 5.2.1.

6 Future directions

Although we have an implemented version of our planner, there remain a number of issues that we are now beginning to investigate.

6.1 Generating Sample Configuration

The uniform sampling scheme described in Section 3 reflects complete ignorance about the environment. If prior knowledge, either about the environment or the set of tasks that the robot will perform, were available, an appropriate importance sampling scheme, or even a deterministic scheme (if the existence of certain obstacles were known in advance) could be used.

Even without prior knowledge about the environment, there are ways to improve the quality of the set of sample configurations. We could, for example, attempt to maximize the portion of the workspace that is covered by the sampled configurations (e.g., by generating configurations for a uniform distribution of end-effector positions in the workspace). In a similar spirit, we could attempt to maximize the minimum distance between samples as in [4]. The quality of the samples can be further improved by exploiting on the manipulability measure associated with the manipulator Jacobian matrix [27]. The basic idea is the following. In regions of the C-space where manipulability is high, the robot has great dexterity, and therefore relatively fewer samples should be required in these areas. Regions of the C-space where manipulability is low tend to be near (or to include) singular configurations of the arm. Near singularities, the range of possible motions is reduced, and therefore such regions should be sampled more densely.

These criteria can be posed as optimization problems with deterministic solutions; however, in each case, the complexity of the resulting problem would prohibit its direct solution. Therefore, we are currently investigating importance sampling approaches that incorporate the criteria described above for sampling C-space.

6.2 Graph Enhancement

In previous probabilistic roadmap planners, if the initial roadmap was not singly connected, an enhancement stage was used to connect its various components. In our work, since there will be no obstacles in the environment, the roadmap will always have a single connected component, and therefore the traditional idea of enhancement does not apply. However, unlike previous approaches, we are concerned with maintaining the connectivity even if arcs or nodes are deleted from the roadmap. Therefore, we are investigating two methods to improve the robustness of roadmap connectivity to the addition of obstacles to the environment.

In the first method, the enhancement stage will attempt to increase the cardinality of the minimum cut set of $\mathcal{G}$. In particular, after an initial $\mathcal{G}$ is computed, minimal cut sets of both edges and vertices will be found, and then the C-space will be sampled in an attempt to add paths that would reconnect the graph if these cut edges or vertices were removed. This sampling would be driven by constructing the workspace volumes corresponding to the elements of the cut set (these are given by ϕ_n^{-1} and ϕ_a^{-1}). In order to disconnect the graph, an obstacle would need to touch each of these volumes. An algorithm such as the one proposed in [10] could be used to rapidly deduce a minimal set of cells in the workspace that satisfy this condition, and the mapping from workspace to C-space obstacles could then be used to seed an importance sampling algorithm that would add nodes to $\mathcal{G}$.

The second method relies on a quantitative evaluation of the quality of $\mathcal{G}$. We will investigate a property that we call ϵ-robustness (inspired by the notion of ϵ-goodness described in [14]). We say that $\mathcal{G}$ is ϵ-robust if no spherical obstacle in the workspace of radius ϵ (or less) can cause $\mathcal{G}$ to become disconnected. Using this concept, it will be possible to ensure after the enhancement stage that $\mathcal{G}$ can tolerate certain types of obstacles. In particular, for a specified value of ϵ, $\mathcal{G}$ can be tested for ϵ-goodness. This test can be performed in such a way that the specific workspace spheres that violate the ϵ-goodness criterion can be enumerated, and these can then be used, as above, to seed an importance sampling algorithm that will add nodes to $\mathcal{G}$.

6.3 Workspace to C-space mapping

There are a number of possible improvements that can be made to reduce the computation time for the mapping and for more efficient representation of the mapping.

Improvement in computation time Using collision testing algorithms to compute ϕ_n and ϕ_a, while allowing for completely general geometric descriptions of the robot, is not the most efficient means available for computing $\phi_n^{-1}(q)$. An alternative is to consider 3D voxel scan-conversion techniques[11]. In our case, we do not need the full power of these algorithms as used to generate realistic images of three-dimensional scenes. We only need the components that compute the occupancy bitmaps for geometric data. This operation could also take advantage of any hardware acceleration available for this operation, as was done in [6] to accelerate the computation of Voronoi diagrams using polygon rasterizing hardware.

Efficient representations As described above, $\phi(w)$ is connected and compact. Exploiting the connectedness of regions for the purposes of compression has been a popular idea in the image processing and computer vision communities for many years [23]. Well known examples of this include using quadtrees and run length coding to compress images. In each case, the compression exploits an efficient representation of neighborhood structure, and then encodes homogeneous neighborhoods. For quadtrees, neighborhoods are defined by recursive partitions of the image, and in run length coding neighborhoods are defined in terms of individual horizontal rows in the image.

To compress the representations for $\phi_n(w^*)$, we will make use of the observation that often $\phi(w^*)$ is often fairly symmetric with respect to its center of mass. We plan to investigate using a prespecified graph traversal algorithm, and recording the depths at which the graph traversal algorithm exits or reenters $\phi(w^*)$. For most problems, breadth first traversal is likely to be the most effective. For example, if $\phi(w^*)$ were a hyper-sphere in an n-dimensional C-space, then this representation would require only the storage of a root node and a single integer to encode both $\phi_n(w^*)$ and $\phi_a(w^*)$. Breadth first traversal could then reconstruct the exact set of nodes and arcs in $\mathcal{G}$ that correspond to the cell w^*.

This approach is analogous to run length coding. In run length coding, the lengths of strings of one's are stored (for binary images). This approach essentially works by imposing an ordering on the pixels in the image (raster scan ordering), and then encoding when a region of one's is exited or reentered. In our proposed approach, we will use a breadth first tree traversal to impose an ordering on nodes and arcs in $\mathcal{G}$, and use the analogous idea of encoding tree depths at which $\phi(w^*)$ is exited or reentered. We note here that approaches analogous to 2^n-trees are not appropriate in our case, because these methods are very sensitive to small perturbations.

7 Conclusions

We have presented what we believe to be the first planner that is fully able to plan in real time in changing environments. We have presented extensive preliminary analysis of the planner via planning simulations. Although these preliminary results are quite promising, there are a number of open issues that remain, and we have outlined these along with our planned approaches.

References

[1] J. M. Ahuactzin, K. Gupta, and E. Mazer. Manipulation planning for redundant robots: A practical approach. *International Journal of Robotics Research*, 17(7):731–747, July 1998.

[2] 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 Conf. Robotics and Automation*, pages 630–637, 1998.

[3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, pages 155–168, 1998.

[4] P. Bessière, J.-M. Ahuactzin, E.-G. Talbi, and E. Mazer. The "Ariadne's Clew" algorithm: Global planning with local methods. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, pages 39–47, 1994.

[5] D. Hankerson, G. A. Harris, and J. Peter D. Johnson. *Introduction to Information Theory and Data Compression.* Discrete Mathematics and its Applications. CRC Press, New York, 1998.

[6] K. E. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of SIGGRAPH 1999 Conference on Computer Graphics*, pages 277–286, 1999.

[7] C. Holleman, L. E. Kavraki, and J. Warren. Planning paths for a flexible surface patch. In *Proc. IEEE Conf. Robotics and Automation*, pages 21–26, 1998.

[8] T. Horsch, F. Schwarz, and H. Tolle. Motion planning with many degrees of freedom — random reflections at c-space obstacles. In *Proc. IEEE Conf. Robotics and Automation*, pages 3318–3323, 1994.

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

[10] Y.-B. Jia and M. Erdmann. Geometric sensing of known planar shapes. Technical Report CMU-RI-94-24, CMU, July 1994.

[11] A. Kaufman and E. Shimony. 3D scan-conversion algorithms for voxel-based graphics. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, pages 45–75, New York, 1986. ACM.

[12] L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Conf. Robotics and Automation*, volume 3, pages 2138–2145, 1994.

[13] L. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, Aug. 1996.

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

[15] J. J. Kuffner, Jr. and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Conf. Robotics and Automation*, 2000.

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

[17] M. Lin and D. Manocha. Efficient contact determination in dynamic environments. *International Journal of Computational Geometry and Applications*, 7(1):123–151, 1997.

[18] E. Mazer, J. M. Ahuactzin, and P. Bessière. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, 9:295–316, 1998.

[19] A. McLean and I. Mazon. Incremental roadmaps and global path planning in evolving industrial environments. In *Proc. IEEE Conf. Robotics and Automation*, pages 101–107, 1996.

[20] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139, June 1997.

[21] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, editors. *MPEG Video Compression Standard.* Chapman and Hall, New York, 1997.

[22] M. H. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.

[23] A. Rosenfeld and A. Kak. *Digital Picture Processing.* Academic Press, New York, 1982.

[24] D. Vallejo, C. Jones, and N. M. Amato. An adaptive framework for 'single shot' motion planning. Technical Report 99-024, Department of Computer Science, Texas A&M University, College Station, TX, Oct. 1999.

[25] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. Motion planning for a rigid body using random networks on the medial axis of the free space. In *Proceedings of ACM Symposium on Computational Geometry*, pages 173–180, 1999.

[26] P. G. Xavier. Fast swept-volume distance for robust collision detection. In *Proc. IEEE Conf. Robotics and Automation*, pages 1162–1169, 1997.

[27] T. Yoshikawa. Manipulability of robotic mechanisms. *International Journal of Robotics Research*, 4(2):3–9, 1985.