## A Framework for Real-time Path Planning in Changing Environments


Peter Leven and Seth Hutchinson



*The International Journal of Robotics Research* 2002 21: 999
DOI: 10.1177/027836490202101200 1

**Peter Leven**
**Seth Hutchinson**

Electrical and Computer Engineering
The Beckman Institute
University of Illinois
Urbana, IL 61801

# A Framework for Real-time Path Planning in Changing Environments

## Abstract

*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, thus facilitating real-time planning. 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.*

*In this paper, we first discuss the construction of the roadmap, including how random samples of the configuration space are generated using an importance sampling approach and how these samples are connected to form the roadmap. We then discuss the mapping from the workspace to the configuration space roadmap, explaining how the mapping is generated and how it can be encoded efficiently using compression schemes that exploit redundancy in the mapping. We then introduce quantitative robustness measures and show how these can be used to enhance the robustness of the roadmap to changes in the environment. Finally, we evaluate an implementation of our approach for serial-link manipulators with up to 20 joints.*

KEY WORDS—probabilistic roadmaps, motion planning

## 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 (PRM) planners (Amato et al. 1998; Horsch, Schwarz, and Tolle 1994; Kavraki and Latombe 1994; Overmars and Švestka 1994; Wilmarth, Amato, and Stiller 1999). 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 direct descendant of the PRM 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 PRM methods, our method is aimed at changing environments and therefore cannot exploit the premise that planning will occur many times in the same environment. Note that we are not concerned here with the problem of motion planning among moving obstacles whose trajectories are known a priori. We are interested in motion planning for environments in which obstacle locations are unknown a priori, or in which obstacle motion is unpredictable, such as would be the case for a mobile robot equipped with a manipulator operating in a previously unknown environment.

The development of our ideas has been driven by the recent and continuing explosion of available computational power, both increased processor speeds and increased memory sizes. It is not, of course, reasonable to expect algorithms with exponential space or time requirements to become feasible merely by increasing computational power, but it is quite reasonable now to encode fairly fine discretizations of the environment (which is at most three-dimensional (3D)) and to encode a

The International Journal of Robotics Research
Vol. 21, No. 12, December 2002, pp. 999-1030,

representation of the mapping between workspace obstacles and obstacle regions in the C-space, provided the C-space is represented by a combinatorial data structure that does not grow exponentially with the dimension of the C-space (for example, the graph representations used by PRM methods). These newly realized abilities form the basis for our new approach.

Our method begins, as do the PRM planners, by constructing a graph 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 PRM 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 graph. 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 motion 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, and then deletes the corresponding nodes and arcs from the graph, using the encoded mapping. Planning is then reduced to connecting the initial and final configurations to the graph (again, as is also the case with the PRM planners), and then searching the graph 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 graph becomes disconnected. This is true for any of the PRM planners (once we know 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 during the preprocessing stages.

In the on-line planning stage, our method runs in real time; plans are generated in less than 1 s. 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.

In the remainder of the paper, we discuss the features of our algorithm. After a brief review of the related path planning literature in Section 2, we present a discussion in Section 3 of the construction of the roadmap to be used for path planning, followed by a discussion of the construction of the mapping from the workspace to the roadmap in Section 4. We continue with a discussion of methods for enhancing the roadmap in Section 5. We then discuss results using this approach with serial-link robots in two-dimensional (2D) and 3D environments in Section 6, and we finish with concluding remarks in Section 7.

## 2. Related Research

The earliest work in path planning produced exact algorithms (see, for example, Canny (1988) and Schwartz, Sharir, and Hopcroft (1987)) and methods that build an approximate representation of the full volume of C-space (see, for example, Brooks and Lozano-Pérez (1983), Kambhampati and Davis (1986) and Lozano-Pérez (1983)). In the former case, the best-known algorithms have exponential complexity and require exact descriptions of both the robot and its environment, whereas in the latter case, the size of the representation of C-space grows exponentially in the dimension of the C-space.

The fact that real robots rarely have an exact description of the environment, coupled with a desire for real-time planning, led to the development of potential field approaches (Hwang and Ahuja 1988; Khatib 1986; Koditschek 1989). The idea of potential field approaches is to construct a scalar function over the C-space that represents the goal region as the global minimum in the field and the obstacles as local maxima. Path planning is then reduced to following the gradient of the potential function until the goal is reached. The advantage of this approach is that the potential functions are easy to compute, making the planner fast. Unfortunately, it is difficult to create potential functions with a single global minimum at the goal; therefore, these planners are easily trapped by local minima.

The problems of local minima in potential field planners led to the development of randomized planning (Barraquand and Latombe 1991). In this approach, when a local minimum is detected, a random motion is performed to try to escape the local minimum. Planning then can be considered a graph search, where the nodes of the graph are the sequence of local minima encountered when searching for the goal.

The randomized motion planners proved effective for a large range of problems, but required extensive computation time for some robots in certain environments (Hsu, Latombe, and Motwani 1999; Kavraki et al. 1996). This limitation, together with the idea that a robot will operate in the same environment for a long period of time, led to the development of the PRM planners (Amato et al. 1998; Horsch, Schwarz, and Tolle 1994; Kavraki and Latombe 1994; Overmars and Švestka 1994; Wilmarth, Amato, and Stiller 1999). The idea that the cost of planning will be amortized over many planning episodes justifies 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.

The two stages of the PRM planners can be described as follows. In the preprocessing stage, a roadmap is constructed

in the free C-space. The nodes of the roadmap are created by some random sampling scheme, and pairs of nodes are interconnected using a simple local planner. After construction, this roadmap may contain more than one connected component, in which case an enhancement operation may be performed to try to connect the different components together. In the second stage, planning queries are performed. For each planning query, the initial and goal configurations are connected to the network and the network is searched for a path.

The PRM planners that use this two-stage processing mechanism are targeted toward environments in which the obstacles are stationary and their positions are known in advance. For environments for which this assumption does not hold, *single-query* variants were developed that build the roadmap as they search for a path (Hsu, Latombe, and Motwani 1999; Kuffner and LaValle 2000; Vallejo, Jones, and Amato 1999). PRM-type approaches have also been used for sensor-based exploration of unknown environments. For example, Mehrandezh and Gupta (2002) use a robot equipped with a skin sensor to explore the environment using a lazy-PRM approach.

There have been a number of previous approaches to path planning in changing environments. In some cases, 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, for example, for the Ariadne's Clew algorithm reported in Bessière et al. (1994) and Mazer, Ahuactzin, and Bessière (1998). 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 (Ahuactzin, Gupta, and Mazer 1998; McLean and Mazon 1996). The idea of incrementally expanding a network for single-query planning has also been used in Hsu, Latombe, and Motwani (1999) and Kuffner and LaValle (2000). In both of these, networks are grown from both the initial and goal configurations until they can be connected, although the details for expanding the network differ. The incremental expansion method in Hsu, Latombe, and Motwani (1999) has also been used in a dynamic environment (Kindel et al. 2000). In Vallejo, Jones, and Amato (1999) an adaptable approach that uses multiple local planners is described. At run time, 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 (Horsch, Schwarz, and Tolle 1994; McLean and Mazon 1996), the idea of somehow representing the mapping from the workspace to the C-space was incorporated. In Horsch, Schwarz, and Tolle (1994), 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 run time 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 collisions with each of these obstacles are determined. When objects are detected at run time, the corresponding arcs are deleted from the graph. In McLean and Mazon (1996), the paths tree (created by a 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. The key advances in our work over these latter approaches are: (1) the ability to represent very large mappings by using data compression schemes that exploit redundancies in the mapping; and (2) methods for improving the robustness of the underlying roadmap to changes in obstacle placement in the environment.

## 3. Constructing the Roadmap

Our construction of a roadmap of the C-space is very similar to methods used in previous PRM planners (Kavraki and Latombe 1994). Nodes are generated by generating sample configurations, and these nodes are then connected to form a roadmap. We will denote this roadmap by $\mathcal{G} = \langle \mathcal{G}_n, \mathcal{G}_a \rangle$, in which $\mathcal{G}_n$ is the set of nodes in the roadmap and $\mathcal{G}_a$ is the set of arcs in the roadmap.

Generating samples to construct $\mathcal{G}_n$ is potentially more complex than for the traditional PRM planners, since we cannot exploit the geometry of the obstacle region in the C-space $\mathcal{C}$ to guide the sampling of $\mathcal{C}$. In traditional PRM planners, the goal is to find enough collision-free samples of the C-space so that the connectivity of the collision-free portion of $\mathcal{C}$ can be captured by the roadmap. In our approach, all samples are collision-free because there are no obstacles (except for the case of self-collision); therefore, our goal is to generate samples such that our roadmap remains connected despite the introduction of obstacles into the workspace.

Once the nodes have been generated, they must be connected to form the roadmap. There are two issues to be addressed in this phase of the roadmap construction. The first

is to decide which pairs of nodes to connect. For this purpose, a distance function is defined that provides a measure of the expected difficulty of connecting a pair of nodes. The second issue is verification that a pair of nodes can be connected. This is the role of the local planner, which must verify that a collision-free path exists joining the two nodes. Traditionally, this is the most expensive part of the PRM method, as it involves many calls to collision testing libraries to determine the feasibility of paths. With our approach, connecting the nodes to generate $\mathcal{G}_a$ is simpler; since our roadmap is constructed without the presence of obstacles in the workspace, only self-collision need be considered when generating local paths between nodes.

The remainder of this section is organized as follows. We begin with a discussion on the sampling of $\mathcal{C}$, including techniques others have used and those that are appropriate for our planner. We follow this with a discussion of the local planner that is used to connect pairs of configurations. Finally, we discuss the distance functions that are used to determine which pairs of nodes to connect in the roadmap.

### *3.1. Generating Sample Configurations*

For traditional PRM methods, the roadmap is constructed for a workspace that contains obstacles. The goal of sampling is then to generate sample configurations in the free C-space (denoted $\mathcal{C}_{free}$). The simplest approach is to use uniform random sampling, discarding samples that correspond to collisions with obstacles. This technique makes no assumptions about the distributions of the obstacles and is relatively easy to analyze (Kavraki, Kolountzakis, and Latombe 1996). Unfortunately, the number of samples this technique places in any particular region of $\mathcal{C}_{free}$ is proportional to its volume; therefore, uniform sampling is unlikely to place samples in narrow passages, and the resulting roadmap is likely to have a connectivity that differs from that of $\mathcal{C}_{free}$.

A number of sampling approaches have been proposed recently to deal with this problem. In Horsch, Schwarz, and Tolle (1994) and Kavraki and Latombe (1998), a second stage of sampling is used, in which samples are added to regions where the roadmap has few nodes. A number of approaches attempt to place samples near the boundaries of obstacles (Amato et al. 1998; Overmars and Švestka 1994; Amato and Wu 1996; Boor, Overmars, and van der Stappen 1999). An alternative to placing samples near the obstacle boundaries is to sample near the medial axis of $\mathcal{C}_{free}$ or $\mathcal{W}$ (Wilmarth, Amato, and Stiller 1999; Guibas, Holleman, and Kavraki 1999; Holleman and Kavraki 2000). Other sampling approaches aim at growing a search tree in $\mathcal{C}_{free}$ so that the tree optimally explores the space (Hsu, Latombe, and Motwani 1999; Kuffner and LaValle 2000; Ahuactzin, Gupta, and Mazer 1998; Yu and Gupta 1999; Ahuactzin et al. 1992; Ahuactzin, Mazer, and Bessière 1995). For the more difficult problem of kinodynamic planning (which considers constraints on robot velocity as well as configuration) samples can be generated by sampling from the space of control inputs, and applying the sample control at a node in the existing roadmap (Kindel et al. 2000; LaValle and Kuffner 1999, 2000).

For our planner, 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 (that is, collision between distinct links of the robot) is prohibited. The first method that we have investigated is to sample 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 have been used.

In addition to uniform sampling of the C-space of the robot, we have investigated an importance sampling scheme that is based on the manipulability measure associated with the manipulator Jacobian matrix (Yoshikawa 1985). Manipulability is a quantitative measure of the ability of a robot to position and orient its end-effector. The formula for the manipulability is $\omega = \sigma_1\sigma_2 \ldots \sigma_n$, in which $\sigma_i$ are the singular values of the manipulator Jacobian.

The basic idea for using the manipulability as a bias for sampling 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.

We adopt the following strategy for concentrating samples in regions of low manipulability. First, for each robot, we compute a discrete representation of the cumulative distribution function of the manipulability. We create this distribution function using the following approach. First, we sample the C-space of the robot uniformly at random and compute the manipulability for each configuration. We exclude from this computation configurations in which the robot collides with itself. We then create a histogram of the manipulability values that we have computed. Finally, we normalize the number in each bucket of the histogram and create the cumulative distribution function from these normalized values. The bucket size of the histogram and the number of samples to take are parameters.

Some example manipulability probability distributions for six-joint robots are shown in Figure 1. In this figure, the plots labeled "Not filtered" correspond to sampling the manipulability of the robot without filtering out samples in which the robot is in self-collision; the plots labeled "Filtered" do exclude these samples. In both cases and for both robots, 10 million samples were evaluated for manipulability. In addition,

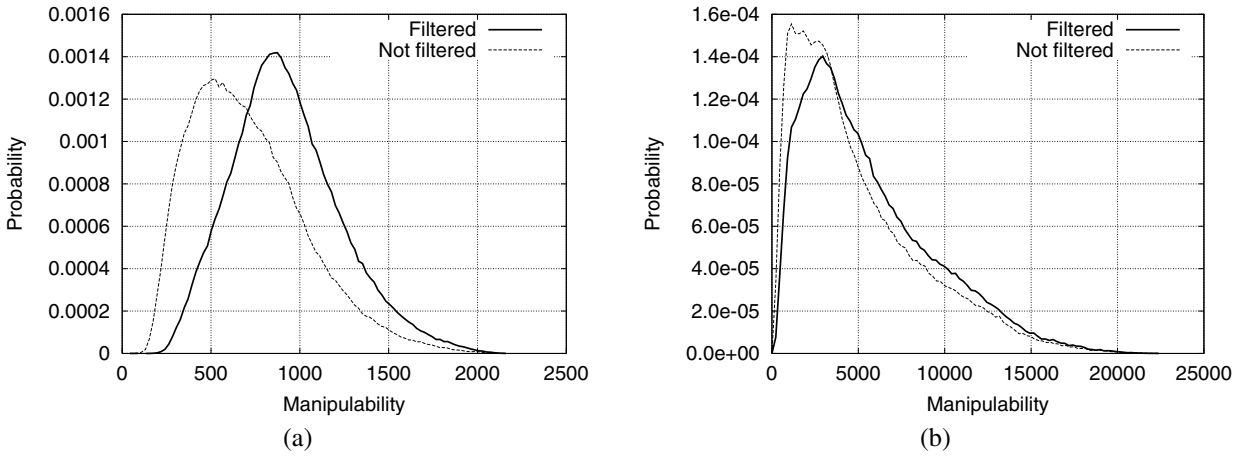



Fig. 1. Probability distribution functions for the manipulability of a robot with six joints: (a) a planar robot and (b) a 3D workspace robot.

the gnuplot "csplines" function was used to smooth the plots. An explanation for the shift that can be seen for both robots in the probability distribution when filtering out self collisions is that configurations in which the robot is in collision with itself tend to be configurations for which the manipulability is low.

We use these manipulability distributions to guide the sampling as follows. First, we compute a self-collision-free sample of the C-space uniformly at random. We then compute the manipulability measure for this configuration and use this value as an index into the manipulability cumulative distribution table of the robot. The value in the table at this index is then used as the probability of rejection for this configuration.

One shortcoming of the manipulability measure for our purposes is that it does not reflect joint limits. When the robot is near a joint limit, its movement is restricted. In an effort to include samples near joint limits we adopt the following convention: at configurations in which some joint is near a limit, the manipulability is defined to be zero. The nearness of a joint to its limit is a parameter of our sampling algorithm.

### *3.2. Local Planner*

The local planner determines how the nodes of the roadmap are connected and whether the connection is feasible, given the presence of obstacles. In traditional PRM approaches, local planning can be a difficult problem due to the presence of obstacles. As a result, many different approaches have been proposed (Kavraki and Latombe 1994; Overmars and Švestka 1994; Hsu, Latombe, and Motwani 1999; Bessière et al. 1994; Ahuactzin, Gupta, and Mazer 1998; McLean and Mazon 1996; Kindel et al. 2000; Kavraki, Kolountzakis, and Latombe 1996; Boor, Overmars, and van der Stappen 1999; Ahuactzin, Mazer, and Bessière 1995; LaValle and Kuffner 1999, 2000; Amato et al. 2000; LaValle, Yakey, and Kavraki 1999; Overmars and Švestka 1995; Holleman, Kavraki, and Warren 1998; Kavraki, Lamiraux, and Holleman 1998, Anshelevich 2000).

In our approach, the roadmap is built without obstacles in the workspace. Therefore, the requirements for the local planner are very modest. It should be reasonably fast, as that reduces the time needed to construct the data structures. It should always return the same path when given the same two configurations as input. In addition, the local planner should consider self-collisions of the robot when determining whether two nodes can be connected.

In our implementation, we use the simple straight-line planner (that is, for each arc, a straight-line trajectory in the C-space is used to connect the two configurations whose connection is implied by that arc). In addition, using the straight-line planner simplifies the calculation of some of the distance functions described below.

### *3.3. Distance Functions*

The distance function provides a measure of the difficulty the local planner would have connecting two configurations, and thus it is useful for selecting pairs of nodes to connect in the roadmap. An ideal distance function would be the swept volume in the workspace of the trajectory connecting two configurations, since intuitively, trajectories with larger swept volumes are more likely to be blocked by obstacles in the environment. Unfortunately, as noted by others (Kavraki et al. 1996; Amato et al. 2000), this distance function is very expensive to compute; therefore, most PRM methods use approximations based solely on the two configurations that are to be connected.

Several distance functions have been defined on the C-space of the robot. These distance functions typically treat the C-space as a Cartesian space and define the distance function accordingly. For example, the Euclidean distance, which is the 2-norm of the differences of each configuration variable, is used in a few planners (Bessière et al. 1994; Amato and Wu 1996; Han and Amato 2000). The 1-norm has also been used (LaValle, Yakey, and Kavraki 1999). A problem with these distance functions is that they weight the configuration parameters equally, when some parameters may have a larger effect than others. A solution to that problem is to add weights to the different configuration parameters, and this has been used in a number of planners (Amato et al. 1998; LaValle and Kuffner 2000; Overmars and Švestka 1995; Bohlin and Kavraki 2000).

Workspace distance functions attempt to measure the motion of the robot in the workspace. One method that has been used for articulated robots is to take the 2-norm of the Euclidean distances between the joint positions in the workspace (Kavraki and Latombe 1994). Two methods for 3D rigid objects that were tested in Amato et al. (2000) are the distance between the center of mass of the object at two configurations and the maximum distance between any vertex of the bounding box at one configuration and its corresponding vertex at the other configuration. For flexible surface objects, a workspace-defined distance function is the sum of the translation distance, scaled rotation, and maximum displacement of a control point (Kavraki, Lamiraux, and Holleman 1998). Another distance function is defined as the sum of the distances between unit vectors of the coordinate frame of the end-effector of the robot. This distance function has been used for manipulation planning (Ahuactzin, Gupta, and Mazer 1998) and for inverse kinematics of redundant robots (Ahuactzin and Gupta 1999). Another variant on a workspace distance function is defined for nonholonomic robots in 2D workspaces (Overmars and Švestka 1995). In this approach, the distance function is defined as the length of the minimum RTR path connecting two configurations.

For our experiments, we selected from the literature the four distance functions listed in Table 1. For the equations in this table, the robot has $n$ joints, q and q′ are the two configurations corresponding to different nodes in the roadmap, $q_i$ 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 $\mathcal{A}$ at configuration q. Versions of $\mathcal{D}_2^{\mathcal{W}}$ and $\mathcal{D}_2^{\mathcal{W}}$ were also used in Kavraki et al. (1996).

Most of the distance functions defined in Table 1 try to capture the cost of a connection by using a measure defined only on the endpoints of the path. While this allows the value of the distance function to be calculated quickly, it does not sufficiently penalize the motion of the robot as it follows the path generated by the local planner. An example of this is shown in Figure 2, where Figure 2(a) shows two configurations that are considered close by the $\mathcal{D}_2^{\mathcal{W}}$ function, and Figure 2(b) shows the region of the workspace swept out by the straight line in the $\mathcal{C}$ planner. For this robot, the first joint has unlimited range of motion, whereas the latter nine have limited range.

To incorporate a measure of the motion of the robot into the distance function, we have defined two new pseudo-distance functions that incorporate the midpoint $\mathrm{q}_m = (\mathrm{q} + \mathrm{q}')/2$ of the path. The first is similar to $\mathcal{D}_2^{\mathcal{W}}$ defined above but with the addition of the midpoint of the path:

$$\mathcal{D}_{m2}^{\mathcal{W}}(\mathrm{q}, \mathrm{q}') = \left[ \sum_{\mathrm{p} \in \mathcal{A}} \left\| \mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q}_m) \right\|^2 + \sum_{\mathrm{p} \in \mathcal{A}} \left\| \mathrm{p}(\mathrm{q}_m) - \mathrm{p}(\mathrm{q}) \right\|^2 \right]^{\frac{1}{2}}.$$

The second is based on the coordinate frame of the end-effector. A form of this function was defined in Ahuactzin and Gupta (1999) for use in a motion-planning-based approach to inverse kinematics. The idea for this function is to connect nodes of the roadmap with similar end-effector positions. Let $\mathcal{F}_a$ and $\mathcal{F}_b$ denote two coordinate frames. We define the distance between these two frames to be the sum of the distances between the unit vectors along the coordinate axes, that is,

$$d(\mathcal{F}_a, \mathcal{F}_b) = d_x + d_y + d_z,$$

where $d_x$, $d_y$, and $d_z$ are the Euclidean distances between the unit vectors along the $x$, $y$, and $z$ axes, respectively (see Figure 3). Using this definition, the distance between two configurations q and q′ is

$$\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}(\mathrm{q}, \mathrm{q}') = d(\mathcal{F}(\mathrm{q}), \mathcal{F}(\mathrm{q}_m)) + d(\mathcal{F}(\mathrm{q}_m), \mathcal{F}(\mathrm{q}')),$$

where $\mathcal{F}(\mathrm{q})$ is the end-effector frame corresponding to the configuration q and $\mathrm{q}_m = (\mathrm{q} + \mathrm{q}')/2$.

It is important to note that, in general, neither of these two functions satisfies the triangle inequality. This restricts the type of nearest-neighbor algorithms that can be used when constructing the roadmap as well as its use for searching for paths in the planning phase. Also, because of the cost of computations associated with using the midpoint, using either of these two functions will increase the time needed to compute the nearest neighbors during roadmap construction, although this may lead to time savings elsewhere.

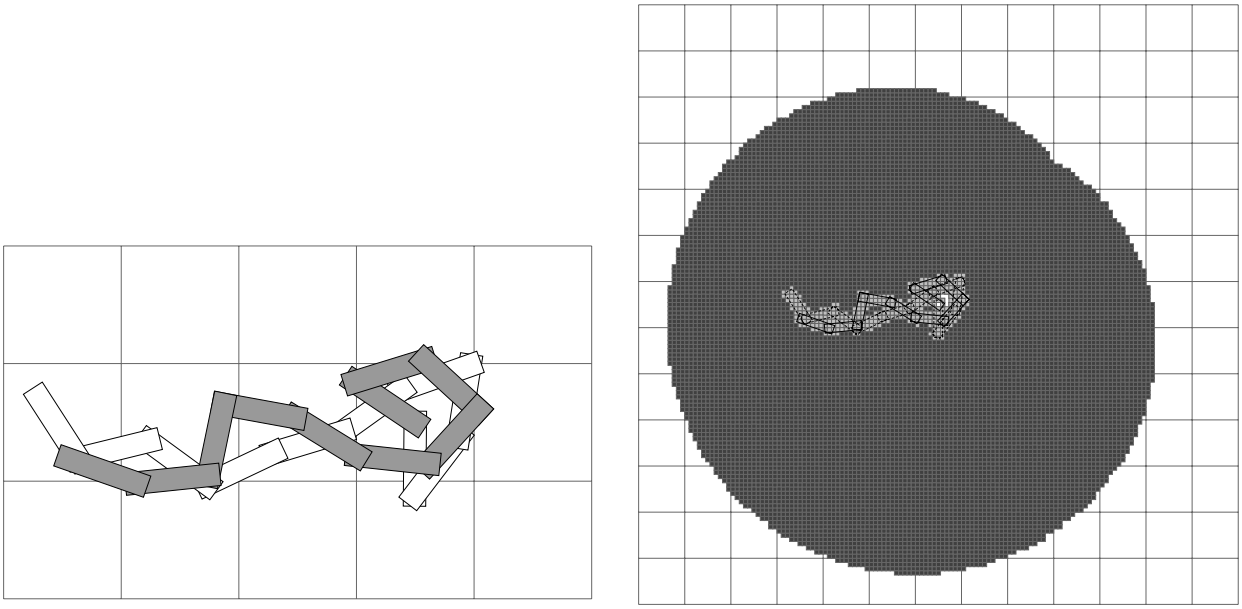
## 4. Workspace to Configuration Space Mapping

A key element of our path-planning approach is the mapping from the workspace to the roadmap in 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 $\phi : \mathcal{W} \rightarrow \mathcal{C}$ as

$$\phi(w) = \{\mathrm{q} \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset\}, \tag{1}$$

**Table 1. Four Distance Functions from the Literature That We Have Investigated**

| | |
|---|---|
| 2-norm in C-space: | $\mathcal{D}_2^{\mathcal{C}}(\mathrm{q}, \mathrm{q}') = \|\mathrm{q}' - \mathrm{q}\| = \left[\sum_{i=1}^{n}(q_i' - q_i)^2\right]^{\frac{1}{2}}$ |
| $\infty$-norm in C-space: | $\mathcal{D}_\infty^{\mathcal{C}}(\mathrm{q}, \mathrm{q}') = \max_n \lvert q_i' - q_i \rvert$ |
| 2-norm in workspace: | $\mathcal{D}_2^{\mathcal{W}}(\mathrm{q}, \mathrm{q}') = \left[\sum_{\mathrm{p}\in\mathcal{A}} \left\|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\right\|^2\right]^{\frac{1}{2}}$ |
| $\infty$-norm in workspace: | $\mathcal{D}_2^{\mathcal{W}}(\mathrm{q}, \mathrm{q}') = \max_{\mathrm{p}\in\mathcal{A}} \left\|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\right\|.$ |



(a) (b)

Fig. 2. (a) Two configurations considered close by the $\mathcal{D}_2^{\mathcal{W}}$ function and (b) the area swept by the robot following a straight line in $\mathcal{C}$ connecting the two configurations.

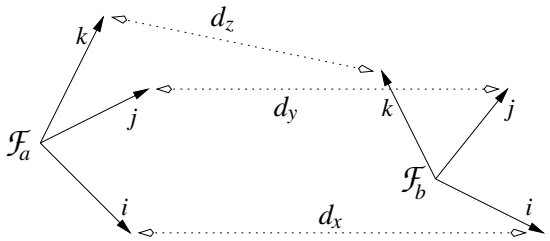



Fig. 3. Illustration of $d_x$, $d_y$, and $d_z$.

in which $w$ is cell of the workspace and $\mathcal{A}(\mathrm{q})$ denotes the 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 an 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 roadmap, and one from the workspace to the arcs in the roadmap:

$$\phi_n(w) = \{\mathrm{q} \in \mathcal{G}_n \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset\}, \quad (2)$$

$$\phi_a(w) = \{\gamma \in \mathcal{G}_a \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset \text{ for some } \mathrm{q} \in \gamma\}. \quad (3)$$

An example of this mapping is shown in Figure 4 for a two-link robot in a 2D workspace. Figure 4(a) shows the robot in its workspace along with a single obstacle ($1 \times 1$ dark square). Figure 4(b) shows the C-space for this robot, including the obstacle region, $\mathcal{CB}$ (shaded), and a sample C-space roadmap that was generated using the $\mathcal{D}_2^{\mathcal{W}}$ distance function. The mapping $\phi_n(w)$ for this obstacle includes the nodes of the roadmap shown as empty circles, and $\phi_a(w)$ includes the arcs shown as dotted lines. Note that we need not include the arcs connected to the nodes in $\phi_n(w)$ in the representation of $\phi_a(w)$, because having one of the endpoints of the arc blocked by an obstacle means that the arc itself is also blocked. For this reason, the arcs connected to nodes in $\phi_n$ are not shown as dotted lines in Figure 4.

### 4.1. Computation of $\phi_n$ and $\phi_a$

In terms of the implementation, it is much easier to compute the inverse maps $\phi_n^{-1}$ and $\phi_a^{-1}$ and, for this reason, we use the inverse maps for the construction of our representation of the mapping. The inverse map $\phi_n^{-1}(q)$ corresponds to the set of cells in $\mathcal{W}$ that intersect with the robot at configuration q, and the map $\phi_a^{-1}(\gamma)$ corresponds to the cells in $\mathcal{W}$ that intersect with the swept volume of the robot as it follows the path $\gamma$ in the roadmap. In our definition of $\phi_a^{-1}(\gamma)$, we exclude the cells in $\phi_n^{-1}$ for both of the endpoints of $\gamma$. An example of $\phi_n^{-1}(q)$ is shown in Figure 5(a) and $\phi_a^{-1}(\gamma)$ is shown in Figure 5(b). The lightly shaded cells correspond to those in $\phi_n^{-1}$ and the darkly shaded cells to those in $\phi_a^{-1}$. When the inverse maps are computed, the forward maps can easily be represented, for example, by using doubly linked pointers.

The construction of the representation for $\phi_n$ 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 $\phi_n^{-1}(q)$ is computed by a voxelization algorithm inspired by the voxelization algorithm for polyhedra described in Kaufman and Shimony (1986). This algorithm essentially limits computation by efficiently building the representation of the inverse map without examining every cell in $\mathcal{W}$.

The computation of $\phi_a^{-1}$ 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. Computing the swept volume for a robot trajectory is not a trivial problem (Abrams, Allen, and Tarabanis 1993; Abrams and Allen 1995; Blackmore and Leu 1990; Boussac and Crosnier 1996; Ma, Jiang, and Chan 2000). There is a simple solution for convex polyhedra that only translate (Xavier 1997), but the general case for motion with rotation is much more complex. Because of this complexity, methods for approximating the swept volume have been proposed, such as a polyhedral approximation (Abrams, Allen, and Tarabanis 1993; Abrams and Allen 1995), or B-splines in Ma, Jiang, and Chan (2000).

We have developed an approximate method for computing $\phi_a^{-1}(\gamma)$ for each $\gamma \in \mathcal{G}_a$ as follows. First, $\phi_n^{-1}$ is computed for the two endpoints $\mathrm{q}_a$ and $\mathrm{q}_b$ of $\gamma$. Then, the path corresponding to $\gamma$ is sampled using a recursive bisection method, which proceeds as follows. First, the configuration $\mathrm{q}_m$ corresponding to the midpoint of the segment connecting the two nodes is computed, and $\phi_n^{-1}(\mathrm{q}_m)$ for that configuration is computed. Cells in $\phi_n^{-1}(\mathrm{q}_m)$ that are not in either $\phi_n^{-1}(\mathrm{q}_a)$ or $\phi_n^{-1}(\mathrm{q}_b)$ are added to $\phi_a^{-1}(\gamma)$. In the current implementation, the subdivision of the path continues until no new cells are added to $\phi_a^{-1}(\gamma)$ by the robot at configuration $\mathrm{q}_m$. An example of this subdivision process is shown in Figure 6.

### 4.2. Efficient Representations

With the recent rapid increase in computer memories, it is feasible to use a naïve encoding to represent $\phi_n$ and $\phi_a$ for a fairly large roadmap and a fairly fine discretization of the workspace. An example showing the size of a naïve encoding of $\phi_a$ for several roadmap sizes and different numbers of joints for planar robots without joint limits is shown in Figure 7. Nevertheless, it is beneficial to find more efficient encodings of $\phi_a$ and $\phi_n$, provided that the encoding does not drastically increase the computation required to compute plans on-line. Reducing the size of the representation will also enable us to consider larger roadmaps $\mathcal{G}$, which will increase the efficacy of the on-line planning.

From an information theoretic point of view, 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 (Hankerson, Harris, and Johnson 1998). There are three main sources of redundancy in the representation of $\phi_n$ and $\phi_a$ that can be exploited: (1) the spatial coherence of the set $\phi(w)$ in $\mathcal{C}$ for a specific $w$ in $\mathcal{W}$; (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 roadmap.

The spatial coherence of $\mathcal{CB}$ has been exploited in previous collision checking approaches (e.g. Lin and Manocha 1997; Mirtich 1997; van den Bergen 1999). In our case, spatial coherence derives from the continuity of $\phi$, namely that small

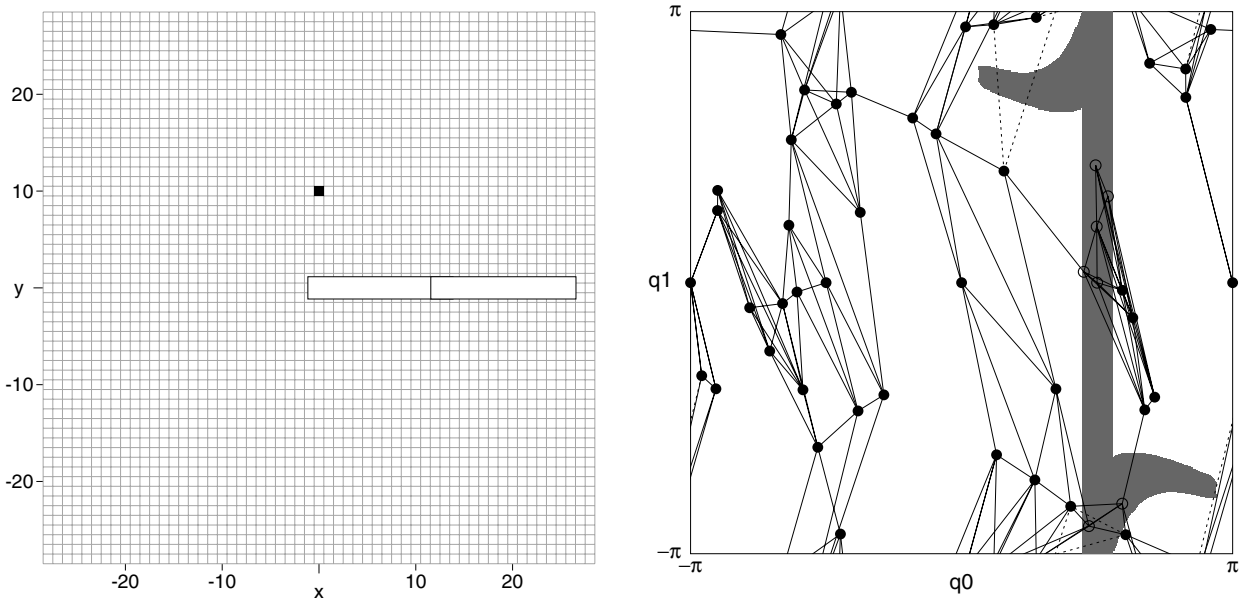



Fig. 4. (a) The workspace of a two-link robot with an obstacle at (0, 10), and (b) the corresponding C-space of the robot.

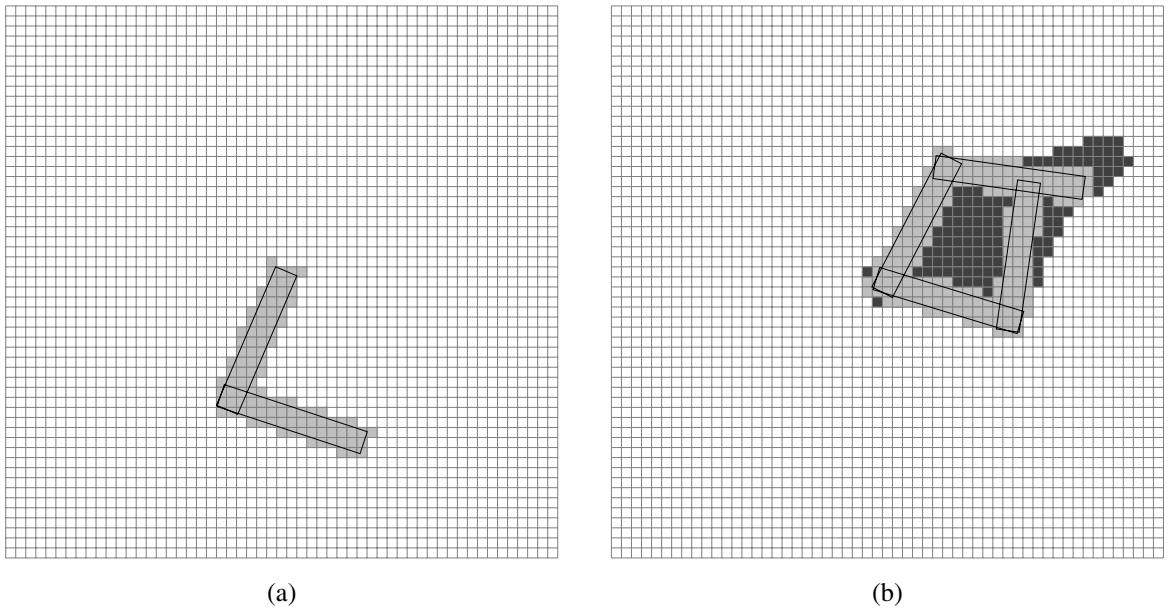



Fig. 5. An example of (a) $\phi_n^{-1}(q)$ and (b) $\phi_a^{-1}(\gamma)$.

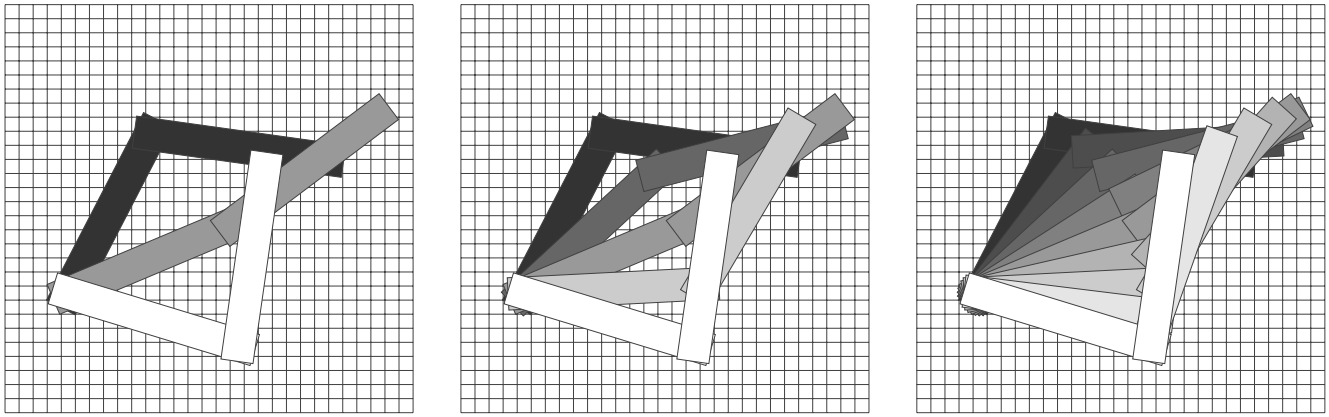


Fig. 6. Illustration of the subdivision method for computing $\phi_a^{-1}(\gamma)$.

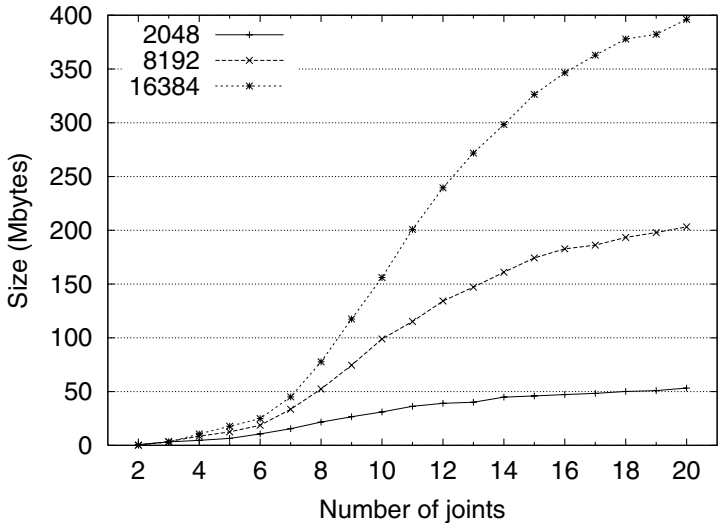



Fig. 7. Size of $\phi_a$ using a naïve encoding.

changes in the location of $w$ 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 $w^*$. We also expect that, because the robots we consider consist of a small set of convex polyhedra, for many $w \in \mathcal{W}$, $\mathcal{CB}(w)$ will contain a small number of connected components, and that these may be exploited to derive a compact representation of $\phi(w)$.

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 $\phi(w^*)$. In some cases, we may be able to improve upon this by selecting some reference set in step (c) other than $\phi(w^*)$, and we discuss this below.

Given the above, we can formulate the corresponding optimization problem. For a specific choice of neighborhood system we can define the cost of the representation by

$$\mathcal{L}(\mathcal{W}^*, \eta) = \sum_{w^* \in \mathcal{W}^*} \left\{ \text{cost}[\phi(w^*)] + \sum_{w \in \eta(w^*)} \text{cost}[\phi(w)] \right\}, \tag{4}$$

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 of the mapping for cell $w$. In this formulation, we are given a set of neighborhoods (this set is defined by the set $\mathcal{W}^*$ of representative cells, and the neighborhood function $\eta$), and the cost of encoding the entire representation is the sum of the costs for encoding the

neighborhoods. The encoding cost for a specific neighborhood is the cost of encoding the representative cell, $w^*$, plus the cost of encoding its neighbors. This leads to the optimization problem

$$\begin{array}{ll} \text{minimize} & \mathcal{L}(\mathcal{W}^*, \eta) \\ \text{subject to:} & \bigcup_{w^* \in \mathcal{W}^*} \eta(w^*) = \mathcal{W} \quad \text{and} \\ & \eta(w_i^*) \cap \eta(w_j^*) = \emptyset, i \neq j. \end{array} \tag{5}$$

This particular formulation of the cost suggests an algorithm that first selects representatives in $\mathcal{W}$, builds the appropriate neighborhoods, and then efficiently encodes the representatives and the neighborhoods. For our compression approach, we borrow techniques from text encoding for the encoding of numbers and sorted sequences of numbers. From image encoding, we borrow ideas for the representation of neighborhoods. We also use the idea of differential encoding of neighbors from MPEG (Mitchell et al. 1997; Furht, Greenberg, and Westwater 1997).

The remainder of this section presents the details of our compression algorithms. While these details are essential to fully understand our approach, they are not necessary for a general understanding of our work.

#### *4.2.1. Efficient Encoding of $\phi(w^*)$*

As described above, certain cells in the workspace are designated as representatives to be used for differential encoding of the neighbors. We have tested the following idea for compressing the representation of $\phi_n(w^*)$; for cells where $|\phi_n(w^*)|$ is relatively large, there is some graph traversal algorithm that efficiently separates the nodes inside $\phi_n(w^*)$ from those outside. In the ideal case, this graph traversal algorithm crosses from the nodes inside $\phi_n(w^*)$ to those outside only once; in this case, only the starting node and the point at which the boundary is crossed need be stored. For example, if $\phi(w^*)$ were a hypersphere in an $n$-dimensional C-space, then using a breadth-first search to trace the graph would require only the storage of the root node and a single integer that denotes the crossing point to encode $\phi_n$. Using the same traversal algorithm starting at the specified start node can then reconstruct the exact set of nodes in $\mathcal{G}$ that corresponds to the cell $w^*$. In our implementation, we have tested the depth-first and breadth-first search algorithms to evaluate the efficacy of this approach with our roadmaps.

This approach is analogous to run-length encoding. In run-length encoding, the lengths of strings of ones and zeros are stored (for binary images) (Witten, Moffat, and Bell 1999). This approach essentially works by imposing an ordering on the pixels in the image (raster scan ordering), and then encoding when a region of ones is exited or entered. In our approach, the graph traversal is used to impose an ordering on nodes and arcs in $\mathcal{G}$. 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 (Hunter and Steiglitz 1979; Rosenfeld and Kak 1982; Samet 1984).

#### *4.2.2. Encoding $\phi$ over Neighborhoods in $\mathcal{W}$*

The encoding of the neighborhoods is based on the idea discussed above that there will be only minor variations in $\phi$ over local neighborhoods of $\mathcal{W}$. Assuming for the moment that the neighborhoods have been determined, one way to encode $\phi$ over a neighborhood $\eta(w^*)$ is to first determine a representative set of nodes and arcs in the roadmap $\phi^*$ and to then specify $\phi(w)$ relative to $\phi^*$ for each $w \in \eta(w^*)$. This is essentially a differential encoding and can be specified as $\phi'(w) = \phi(w) \oplus \phi^*$, where $\phi'(w)$ is the encoded representation of $\phi(w)$ and $\oplus$ is the set symmetric difference operator. This basic idea is at the heart of our schemes for encoding over neighborhoods.

We have investigated two methods for defining the neighborhoods in $\mathcal{W}$. The first method is a region-growing approach, in which seed cells are selected and regions are grown around them. The second method for defining neighborhoods is hierarchical, based on an octree decomposition of the workspace. Both of these will now be discussed in more detail.

In the first method, region growing is used to define neighborhoods. We are confronted with two issues: how to choose seed cells, and when to stop expanding the neighborhood. Choosing the optimal seed cells is a difficult combinatorial optimization problem. Therefore, we have applied a greedy selection approach: at each iteration, the algorithm selects as the seed cell the unencoded workspace cell with the largest representation of $\phi(w)$. In our approach, we simply define the size of the representation of $\phi(w)$ to be the total number of nodes and arcs in $\phi(w)$.

We expand the neighborhood around the seed cell in two stages, constructing two neighborhoods, $\eta_1(w^*)$ and $\eta_2(w^*)$. An example of a resulting neighborhood structure is shown in Figure 8. The neighborhood $\eta_1(w^*)$ is constructed as follows. Let $\phi_{n_1}^*$ be the set

$$\phi_{n_1}^* = \arg\min_{\phi_{n_1}} \left|\phi_{n_1}\right| + \sum_{w \in \eta_1} \left|\phi(w) \oplus \phi_{n_1}\right|,$$

in which $|\cdot|$ denotes set cardinality. That is, $\phi_{n_1}^*$ is such that, if every $w \in \eta_1(w^*)$ is encoded by a symmetric difference with $\phi_{n_1}^*$, then the cost of the representation of $\eta_1(w^*)$ is minimized. Beginning with the seed $w^*$, we add cells to $\eta_1$ until it becomes more efficient to encode the additional cells using the encoding method for $\eta_2$. Cells that are assigned to $\eta_1(w^*)$ are then encoded by their symmetric difference with $\phi_{n_1}^*$.

Each cell in $\eta_2(w^*)$ is encoded by the symmetric difference with the representation of one of its neighbors, $w_N \in \eta_1(w^*)$, that minimizes the resulting representation size. This is illustrated in Figure 8, where the arrows originating in the darkly

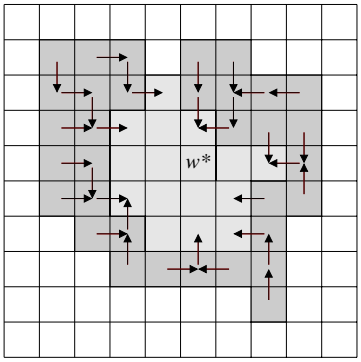



Fig. 8. The two neighborhoods $w^*$: $\eta_1(w^*)$ is the light region containing $w^*$ and $\eta_2(w^*)$ is the darker region.

colored cells point to the neighbor that is used for the differential encoding. Of course, care must be taken to avoid loops in this referencing scheme.

Construction of the neighborhood $\eta_2$ relies on two parameters: a recursion depth and a scale factor. The recursion depth limits the length of the chain of references between adjacent cells. For example, the longest such chain in Figure 8 is 4, with two cells having chains of this length. The scale factor is used to ensure that the encoding of the new cell is sufficiently small; that is, the cell $w$ is added to $\eta_2$ if $|\phi(w) \oplus \phi(w_N)| < s|\phi(w)|$, where $s$ is the scale factor. For cells where there is no neighbor such that $|\phi(w) \oplus \phi(w_N)| < s|\phi(w)|$, we add the cell to $\eta_2$ if $|\phi(w) \oplus \phi(w_N)|$ is the best we can do for this cell, provided $|\phi(w) \oplus \phi(w_N)| < |\phi(w)|$.

The second method that we have investigated for defining the neighborhoods in $\mathcal{W}$ uses an octree-based decomposition of the workspace. The result is a hierarchical set of references for neighborhoods, at the bottom level, composed of eight cells. An advantage of defining the neighborhoods in this manner instead of using region growing is that the reference number for each cell is implicit with the location of the cell, and need not be stored. A disadvantage is that the region-growing method can define neighborhoods that more effectively exploit the redundancy in the mapping.

Once the neighborhoods have been defined by the hierarchy, we are left with the choice of what set to store at the parent of each set of eight children in the hierarchy. To simplify the notation, we label the parent $p$ and the children $c_i$. We have evaluated two such choices, which we label "union" and "best". In the union approach, the set $\phi^*(p)$ stored with the parent is $\cup_{i=1}^{8} \phi(c_i)$. This approach is equivalent to a multiresolution decomposition of the workspace. In the "best" approach, the set $\phi^*(p)$ is chosen to be the set that minimizes

$$\left|\phi^*(p)\right| + \sum_{i=1}^{8} \left|\phi(c_i) \oplus \phi^*(p)\right|.$$

For both choices of the parent set, the set encoded with each child is the set $\phi(c_i) \oplus \phi(p)$.

These two neighborhood models are options to use for the encoding of $\phi$; the particular neighborhood model to use for a given $\phi$ is a choice of the user. We will show in Section 6 the effect of using these models for different robot examples.

### *4.2.3. Encoding the Labels*

Each entity in the representation of $\phi$ (including the nodes in $\phi_n$, the arcs in $\phi_a$, and the cells in $\mathcal{W}$) requires a label. For simplicity, we use non-negative integers for the labels. Since there are many such labels, we can benefit from previous work that has been done in data compression, particularly in the area of efficiently encoding positive integers (Bell, Cleary, and Witten 1990). We can also impose some additional constraints on the labels, such as storing the lists of labels in sorted order. This allows us to use differential encoding of the labels, which can reduce the number of bits required to represent the list (Witten, Moffat, and Bell 1999).

We have devised an encoding scheme, which we call $\Gamma_b$ because of its similarity to the $\gamma$ code (Bell, Cleary, and Witten 1990; Elias 1975). In this code, a number expressed in binary is split into pieces $b$ bits in length. The number of pieces is encoded in unary, with one bit of the length appended to each piece. For example, to encode the number 11 using the $\Gamma_3$ code, first split the binary representation of 11 into two pieces, 001 and 011. To the first piece, prepend a 1 bit, and to the second, prepend a 0 bit. The resulting code is 10010011. Encoding and decoding the $\Gamma_7$ version of this code is particularly efficient on byte-addressable computers.

We have also created an extension to the $\Gamma_b$ code, which we call $\Gamma'_b$. This code treats the number zero as a special case, using just the bit 0 to encode it. Other numbers are encoded by prepending a 1 bit to the $\Gamma_b$ code for the number less 1. This code is particularly useful for sequences of numbers in which the number zero is predominant.

The savings that are achieved by compressing the labels are much less significant than those achieved by the other methods described in this section. For this reason, we omit

here a detailed description of our compression schemes for the labels.

## 5. Roadmap Enhancements

As we have discussed in Section 3, one of the difficulties with the traditional PRM planners is placing samples in narrow passages in $\mathcal{C}_{free}$. This has led to the development of an enhancement phase, in which more sampling is performed to improve the connectivity of the roadmap (Horsch, Schwarz, and Tolle 1994; Kavraki and Latombe 1994; Kavraki et al. 1996; Kavraki and Latombe 1998, Holleman, Kavraki, and Warren 1998; Švestka and Overmars 1995). Since our roadmap is constructed without obstacles, reducing the number of connected components is not the goal for our enhancement phase. Instead, we are interested in preserving the connectivity of the roadmap when obstacles are added to the workspace.

We have examined two methods for evaluating the connectivity of our roadmap in the presence of obstacles, and from these evaluation methods we have derived mechanisms for enhancing roadmap connectivity. The first evaluation method is based on a traditional measure of graph connectivity: the minimum cut set. The second method involves a quantitative measure of the robustness of the roadmap to the addition of obstacles to the workspace. We now discuss both of these evaluation methods and the corresponding enhancement techniques.

### 5.1. Connectivity-based Enhancement

Our first method for enhancement is to examine the edge connectivity of the roadmap and add arcs in regions of low connectivity. The edge connectivity of the roadmap is defined as the minimum of the maximum number of arc-distinct paths (that is, paths with no arc in common) between any distinct pair of nodes. The basic idea is that, in regions of low connectivity, it is relatively easy to disconnect clusters of nodes in the roadmap with the addition of a small set of obstacles in the workspace. The goal is to have the edge connectivity of the roadmap be at least equal to the minimum degree of any node in the roadmap.

Our approach for this form of enhancement is to add arcs to the roadmap until the edge connectivity of the roadmap is equal to the minimum node degree. To do this, we repeatedly apply the unweighted minimum cut set algorithm from Matula (1993) to find the minimum number of arcs that partition the roadmap into two pieces. We then add arcs between these two pieces until the number of such arcs is equal to the minimum node degree. We stop when the number of arcs in the minimum cut set is equal to the minimum node degree.

Once we have found a partition of the roadmap, we use the following approach to add arcs between the two pieces of the partition. First, for each node q in the smaller of the two pieces, we compute the set of cells in $\mathcal{W}$ in $\phi_n^{-1}(q)$, using the inverse of the mapping $\phi_n$ described in Section 4. Next, for each arc $\gamma$ in the cut set, we compute the subset set of cells in $\mathcal{W}$ in $\phi_a^{-1}(\gamma)$ which excludes those cells in $\phi_n^{-1}(q)$ for any node q in the smaller piece. The cells that remain in $\phi_a^{-1}(\gamma)$ for each arc $\gamma$ in the cut set form a set of potential obstacles in the workspace whose presence would disconnect the roadmap into these two pieces.

We then add arcs until enough arcs have been added between the two pieces in the partition. In each phase, we select a set of cells from the subset of $\phi_a^{-1}(\gamma)$ described above for each of the arcs in the cut set. These cells will be considered as being occupied by obstacles as we attempt to add more arcs connecting the two pieces. We then take each node of the smaller piece in random order and attempt to add an arc between it and some node in the larger piece. The reason for selecting the nodes of the smaller piece at random is to try to evenly distribute the new arcs over the nodes of the smaller piece. We evaluate each node of the larger piece in order of increasing distance from the test node in the smaller piece, until we succeed in adding an arc or until we have tested all arcs of the larger piece.

### 5.2. Workspace-based Enhancement

The second method for roadmap enhancement examines quantitatively the relationship between the roadmap and the workspace. In particular, we derive a property that we call *$\rho$-robustness*. We say that $\mathcal{G}$ is $\rho$-robust if no spherical obstacle in the workspace of diameter $\rho$ (or less) can cause $\mathcal{G}$ to become disconnected. Note that $\rho$-robustness is a global property of the roadmap $\mathcal{G}$.

Some motivation for the $\rho$-robustness idea can be seen in Figure 9. Figure 9(a) shows the workspace for a two-link planar arm with no joint limits for joint 1 and joint limits for joint 2. A single obstacle of dimension $1 \times 1$ has been added to the workspace and, as a result, as seen in Figure 9(b), the 50-node roadmap in the C-space is broken into three distinct components. Hence, the roadmap in Figure 9(b) is not $\rho$-robust for any value of $\rho$. An enhanced roadmap is shown in Figure 9(c) (the enhancement caused 26 arcs to be added to the roadmap). This enhanced roadmap is $\rho$-robust for $\rho \leq 4$. In other words, no single obstacle of size less than four units can cause the enhanced roadmap to become disconnected.

Our approach to enhancement based on $\rho$-robustness is to add arcs to the roadmap until the desired level of $\rho$-robustness is achieved (if possible). Our algorithm proceeds as follows. Starting with a cube of size $1 \times 1 \times 1$, sweep cube-shaped cells over the workspace of the robot. For each location of the cube, compute the set of connected connected components. If there is more than one component, add an arc, if possible, to repair the connectivity of the roadmap, using the local planner to verify the feasibility of each arc tested. Repeat the sweep with larger cubes, $2 \times 2 \times 2$, $3 \times 3 \times 3$, etc., up to a cube of side length $\lceil \rho + 1 \rceil$.

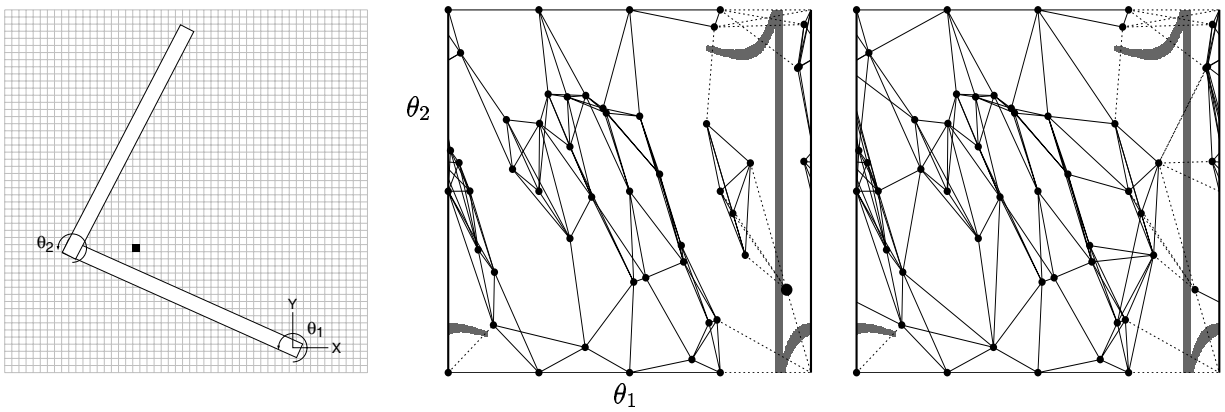



Fig. 9. Effect of an occupied cell in the workspace on the connectivity of the roadmap before and after enhancement.

Unfortunately, it is not always possible to achieve $\rho$-robustness, even for $\rho = 1$. For many robots (particularly those with joint limits), there are specific cells in the workspace such that if an obstacle is placed in one of these cells, $\mathcal{C}_{free}$ itself may become disconnected, or the passage connecting two regions in $\mathcal{C}_{free}$ may become sufficiently narrow that representing this passage in the roadmap becomes difficult. Our approach is to flag these cells in $\mathcal{W}$ and to eliminate them from consideration during the enhancement stage. The resulting roadmap will, of course, not be globally $\rho$-robust, but will be as nearly $\rho$-robust as possible.

## 6. Results

In this section, we evaluate our planner, studying serial-link manipulators operating in 2D and 3D workspaces. We have evaluated our planning approach with several different types of robots: planar arms with between two and twenty joints, robots such as those illustrated in Figure 10 (with fixed base), robots such as those illustrated in Figure 11(a) (with mobile base) and robots such as those illustrated in Figure 11(b) (rigid bodies with six degrees of freedom). For each articulated arm, all but the first joint are subject to joint limits. In earlier work (Leven and Hutchinson 2000), we considered the case of robots without joint limits.

We used the collision-detection package SOLID (van den Bergen 1997, 1999) to test for self-collision of the robot, and V-Clip (Mirtich 1997) to test for collisions between obstacles and the robot; both of these packages use the quickhull algorithm (Barber, Dobkin, and Huhdanpaa 1996) for computing convex hulls. To evaluate how the data structures grow with the size of the roadmap, we tested roadmaps with 2048, 8192, and 16,384 nodes.

The results we show for the planar robots with joint limits are obtained by averaging the results from seven runs. For each of these runs, we construct the roadmap, compute $\phi$, and then compress the representation of $\phi$. This averaging of results does not include the results for the roadmap enhancement techniques.

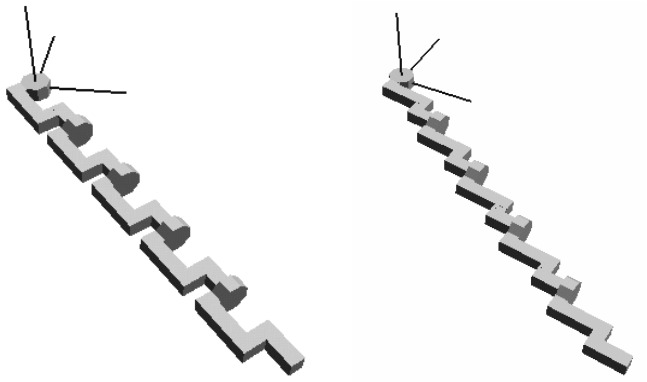
### 6.1. Roadmap Construction

We use two different sampling distributions for sampling the C-space of each robot. The first set of samples comes from a uniform sampling of the first joint for planar robots or the first two joints for robots with a 3D workspace; for both cases, the positions of the other joints are prespecified default values. The purpose of these samples is to touch as much as possible the maximum reach of the robot. The remaining samples come from sampling the C-space of the robot uniformly at random. Of this second set of samples, those in which the robot collides with itself are rejected.

From the uniform random samples, we may also reject samples with probability based on the value of the cumulative distribution function at the manipulability value for the sample. When we impose the manipulability constraint on the random samples, we observe that about twice as many collision-free samples must be generated.

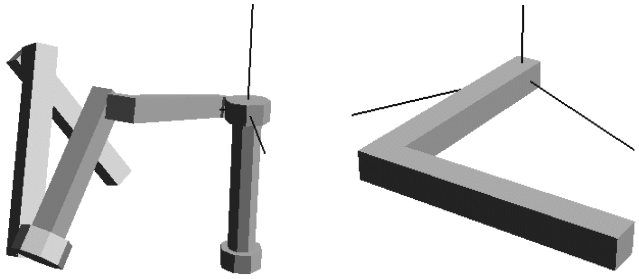
### 6.2. Computing the Representation of $\phi$

In this section, we compare the size of the initial, naïve representation of $\phi_n$ and $\phi_a$ with the results of using a more efficient encoding. This section contains results for a number of test cases. A summary of these results can be found in Section 6.2.5.



(a) (b)

Fig. 10. Two robots with 3D workspace.



(a) (b)

Fig. 11. Two mobile robots with 3D workspace.

*6.2.1. The Sizes of $\phi_n$ and $\phi_a$ Before Compression*

The overall size of $\phi_n$ is the product of the number of nodes and the average number of cells the robot intersects in the workspace. For the planar robots with a fixed maximum length, this means that the size of $\phi_n$ is largely independent of the number of joints. For our experiments, the size of $\phi_n$ per node in $\mathcal{G}_n$ averages around 1000 bytes. The situation is different for the 3D workspace robots, in which the number of cells that intersect with the robot increases with the number of joints. The effect of this on the size of $\phi_n$ can be seen in Figure 12, which shows the size of $\phi_n$ per node in the roadmap. The sublinear increase in the size of $\phi_n$ with the number of nodes is the result of the amortization of the overhead for each cell in $\phi_n$.

The size of $\phi_a$ is shown in Figure 13 for several distance functions for the planar robots with joint limits; an explanation for the labels can be found in Table 2. This figure shows how large the naïve representation of $\phi_a$ can grow for the planar robots. The representation of $\phi_a$ is too large for the 3D workspace robots to compute with the naïve representation; therefore, we defer showing the size of $\phi_a$ for now.

**Table 2. Definition of the Labels Used on Figures for Distance Functions**

| Label | Description |
|---|---|
| w | The $\mathcal{D}_2^{\mathcal{W}}$ distance function |
| W | The $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function |
| F | The $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance function |
| a | When associated with W or F, indicates that an approximation approach is used to find the nearest neighbors |

Figure 13 also shows the effect of the different distance functions and methods for computing nearest neighbors on the size of $\phi_a$. The smallest size of $\phi_a$ for all number of joints is for the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function, both for exact and approximate nearest neighbor computation (the approximate form results

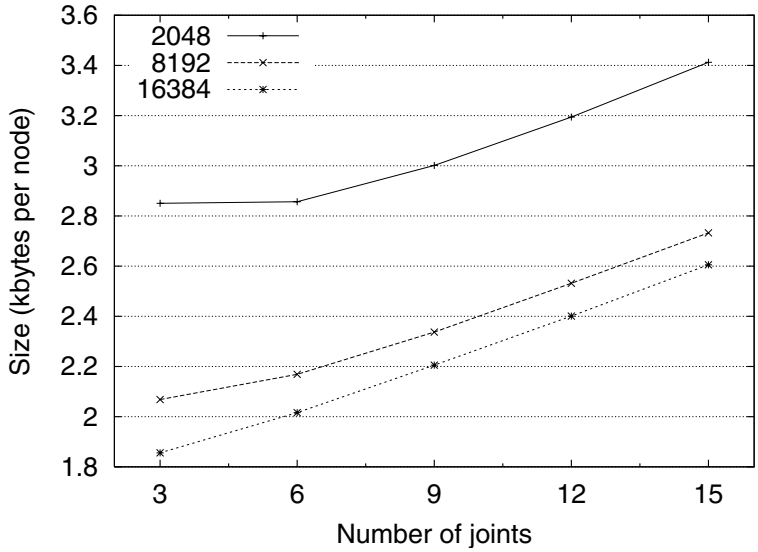



Fig. 12. Size of $\phi_n$ for 3D workspace robots.

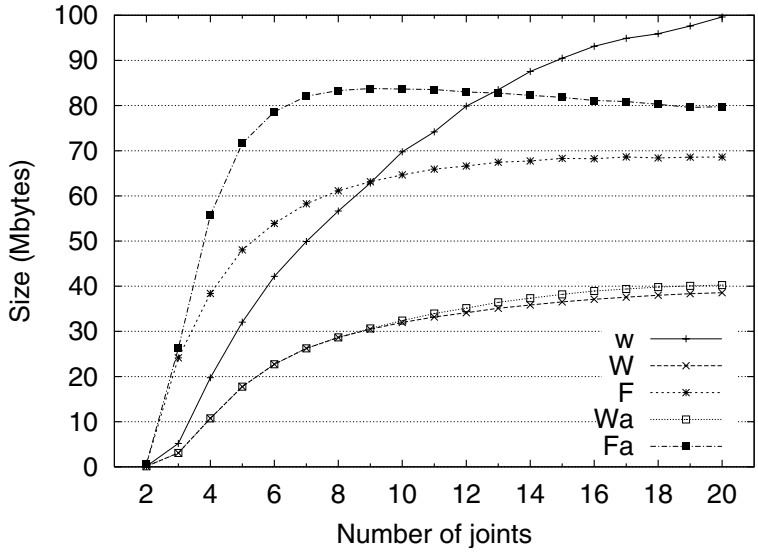
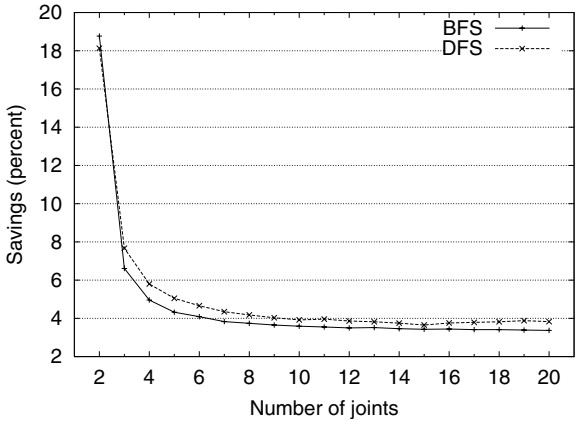



Fig. 13. Size of uncompressed $\phi_a$ for planar robots with joint limits and for roadmaps with 16,384 nodes and five distance functions.

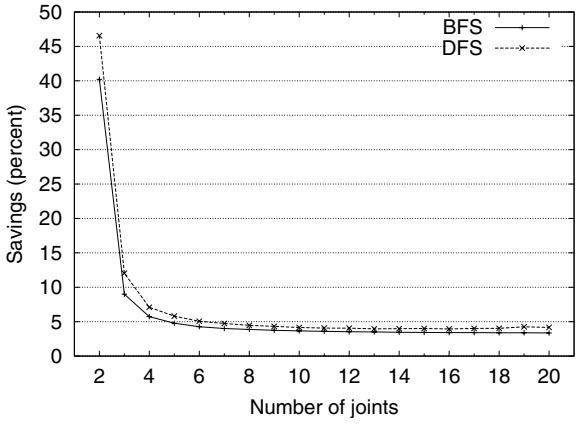
in a slightly larger size). The distance function defined on the end-effector coordinate frame performs worse, which is expected because this distance function uses less information about the positions of the links of the robot.

### *6.2.2. Compression of $\phi(w^*)$*

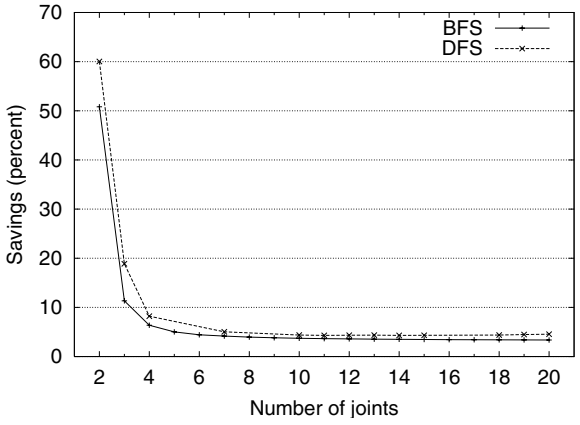
As discussed in Section 4.2.1, we can compress the representation of $\phi(w^*)$ for a particular $w^*$ by imposing an ordering on the nodes in the roadmap and then using a sort of run length coding. We tested this idea on $\phi_n$, trying both depth-first search (DFS) and breadth-first search (BFS) as the method for imposing the ordering. For each cell in $\phi_n$, we tested each node as a candidate starting node, and kept the one that produced the best results. The results are shown in Figure 14. As can be seen in the figure, the results are rather disappointing, showing that it is only really successful for robots with two, or possibly three, joints. The results do not change significantly for




2048 nodes




8192 nodes




16 384 nodes

Fig. 14. Compression results using run-length graph encoding for $\phi_n$. The savings are in terms of the reduction of the number of labels.

different roadmaps constructed using the same set of nodes but different distance functions. What this shows is that the roadmaps we use for planning do not cluster the nodes well enough for the run-length encoding to work well. There may be other roadmaps for which this method will work better.

### 6.2.3. Compression of $\phi$ over neighborhoods in $\mathcal{W}$

In Section 4.2.2 we described how to compress $\phi$ by exploiting redundancy in the mapping over neighborhoods in $\mathcal{W}$. We have evaluated the three neighborhood models proposed there: region-growing, "union" hierarchical, and "best" hierarchical. For the two hierarchical neighborhoods, "union" and "best" define the set of labels that is stored with the parent: with "union", the union of the labels of the children is stored, and with "best", the set of labels that minimizes the label count among the parent and children is stored. For the region-growing neighborhood models, the results shown below use the parameters with a size factor of 2 and a depth limit of 17.

Figure 15 shows the savings that we achieve using the different neighborhood models for $\phi_n$ for both the planar robots and the 3D workspace robots. Using the region-growing method, the amount of savings is relatively flat with different numbers of joints, with a factor of around 3 for the planar robots and 2 for the 3D workspace robots. The savings also tend to be greater for roadmaps with fewer nodes, indicating that the distribution of labels is more uniform for roadmaps with fewer nodes.

Using the hierarchical models yields results that are not as good as those for the region-growing models. Indeed, for the 3D workspace robots, using the "union" model for the neighborhoods results in an expansion of the number of labels (a "savings" of less than one). The hierarchical models tend to perform better as the number of joints of the robot increases. This is due in part to the increase in the average number of cells occupied by the robot at a configuration as the number of joints increases. Another factor is that, with more joints, the links of the robot have a greater tendency to cluster around the origin. With fewer labels in the outer regions of the workspace, the regions of uniformity are larger.

Figure 16 shows the savings that we achieve for $\phi_a$ for planar robots. Figures 16(a), (c), and (e) show the cases for robots without joint limits and roadmaps constructed using the $\mathcal{D}_2^{\mathcal{W}}$ distance function. This is the case for which the region-growing method of compression was designed, and the savings reach a factor of 8.5 at 20 joints using this neighborhood definition. The hierarchical models also perform well in this case, although the "best" model achieves nearly twice the level of savings of the "union" model. For a robot with joint limits and using the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function, the compression results are quite different. This difference can be seen in Figures 16(b), (d), and (f). For these robots, the average swept volume per arc converges to some constant value as the number of joints increases, and this same convergence is reflected in the savings achieved. Notice that for these roadmaps, the "best" hierarchical and the region-growing models produce similar savings. This is also true for the roadmaps for these robots for the other distance functions, although the region-growing neighborhood results are slightly better for the $\mathcal{D}_2^{\mathcal{W}}$ distance function.

Figure 17 shows the savings achieved for robots with 3D workspaces, using the different neighborhood models for $\phi_a$ and roadmaps constructed using the $\mathcal{D}_{m2}^{\mathcal{W}}$ and $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance functions. Results for the second distance function are given to show that the compression ratios do improve for roadmaps constructed with distance functions that result in larger average swept volumes for the arcs. The results for $\phi_a$ for these robots follow the general trend for the planar robots; that is, the region-growing neighborhoods tend to produce the most savings, followed by the "best" hierarchical, and with the "union" hierarchical resulting in an expansion of the number of labels in many cases.

### 6.2.4. Compressing the Labels

As mentioned previously, significant savings were not obtained from compressing the labels. Therefore, we do not present individual results for this case. However, in the summary results presented in Section 6.2.5, label compression is incorporated into the overall compression process.

### 6.2.5. Summary of Compression Results

We now examine the total space required for $\phi$, including the representation of the labels and various sources of overhead, such as the reference number stored in each cell for the region-growing neighborhoods, the count numbers which indicate the number of labels stored with each reference and cell, and the length numbers, which are used to locate a particular cell or reference in the compressed data.

Figure 18 shows the result for $\phi_n$ with a 16,384-node roadmap for both planar and 3D workspace robots; a description of the labels used in this and later figures is given in Table 3.

For the planar robots, compression using the region-growing neighborhoods clearly outperforms the other methods. This is due in large part to the reduction in the number of labels that must be stored using this model. The region-growing neighborhoods also have a slight advantage over the hierarchical neighborhoods in terms of the overhead for storing the neighborhood structure; this is due to the hierarchical neighborhoods being defined for 3D workspaces, and this hierarchy is less efficient for the planar case.

For the 3D workspace robots, on the other hand, the "best" hierarchical neighborhoods tie the performance of the region-growing neighborhoods for robots with more joints, even though the region-growing neighborhoods result in fewer labels. This shows the trade-off between arbitrary

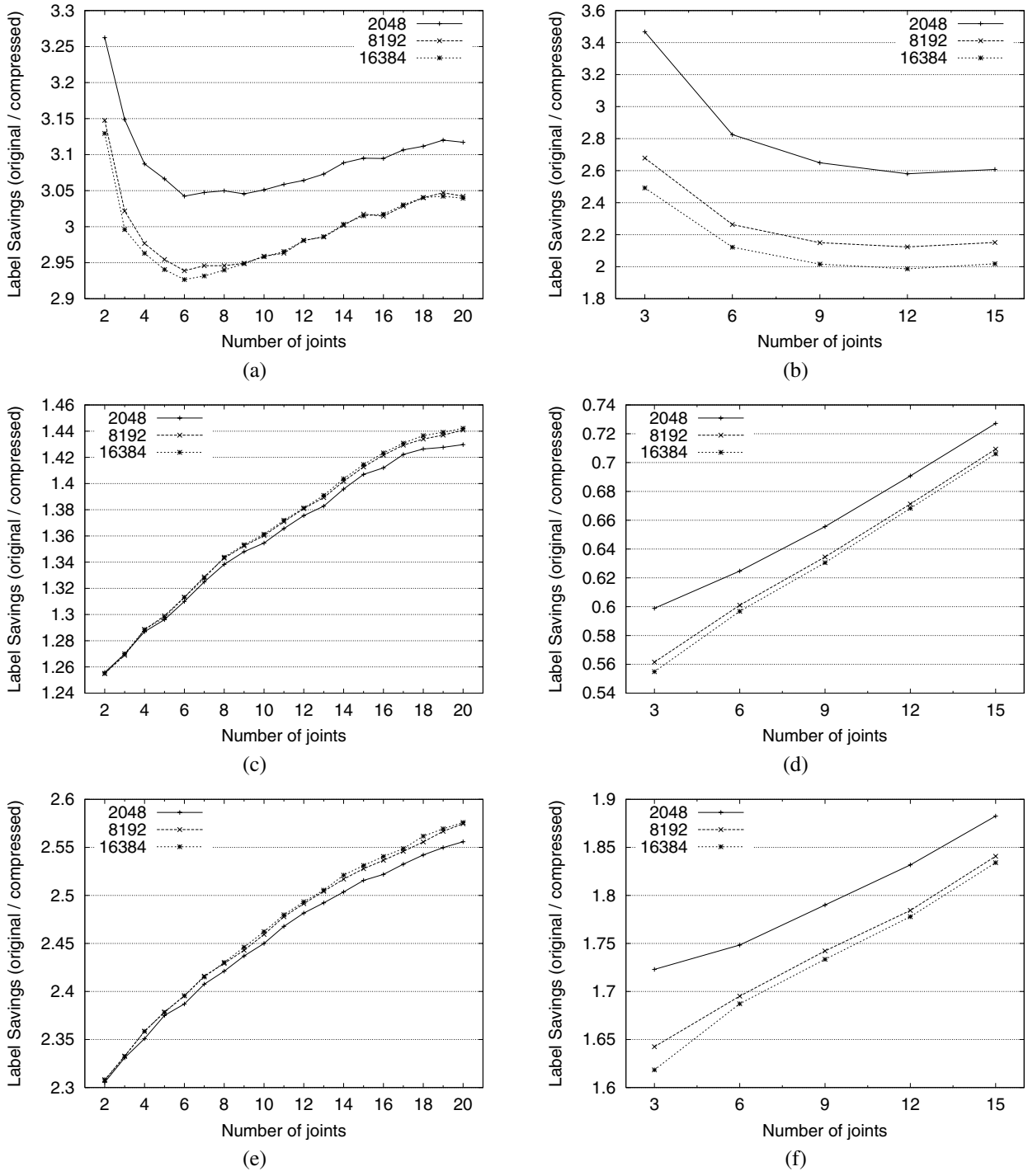



Fig. 15. Compression results for $\phi_n$ for planar robots (a, c, e) and robots with 3D workspace (b, d, f). The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f).

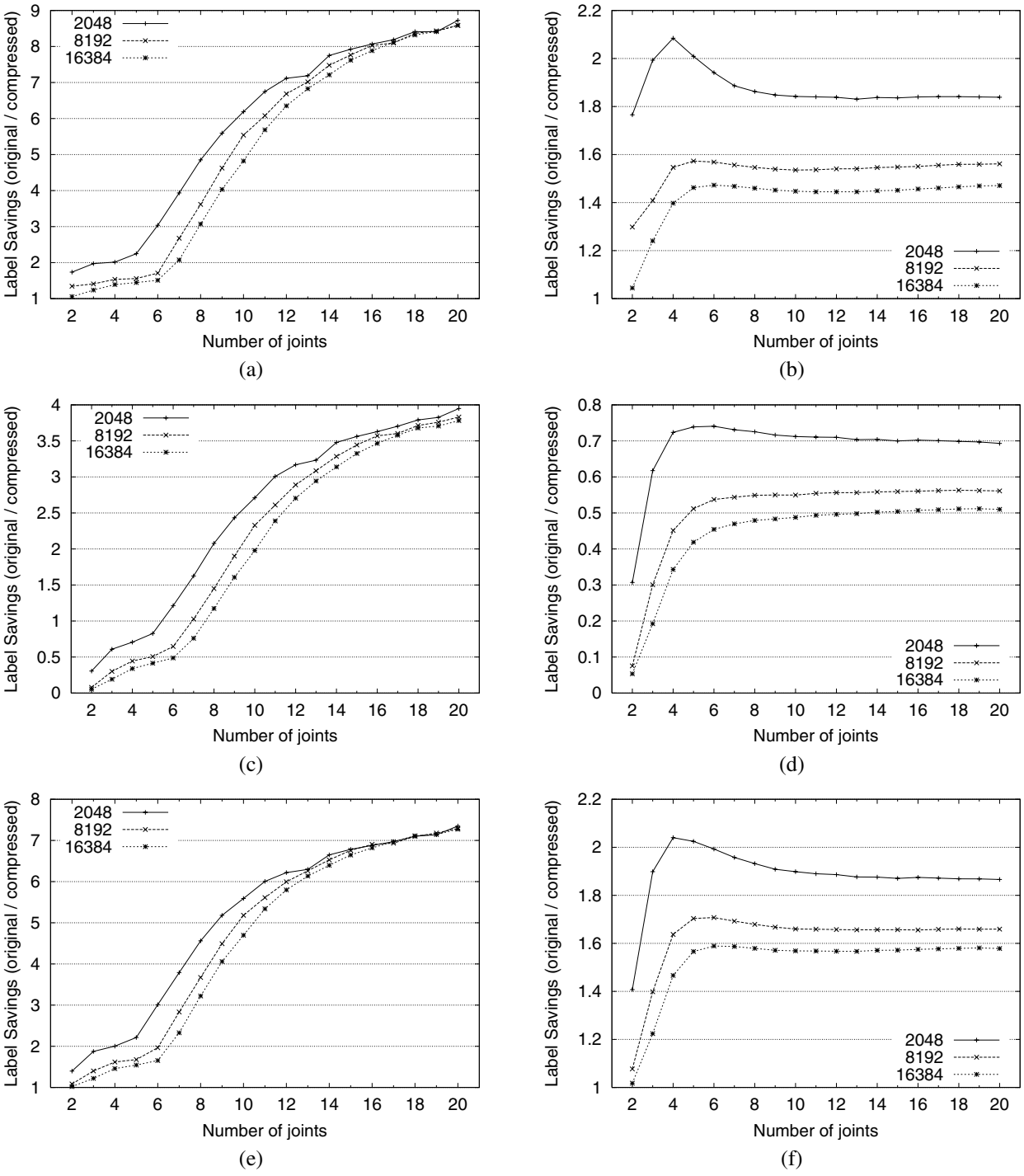



Fig. 16. Compression results for $\phi_a$ for planar robots: $\mathcal{D}_2^{\mathcal{W}}$ distance function for robots without joint limits (a, c, e) and $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function for robots with joint limits (b, d, f). The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f).

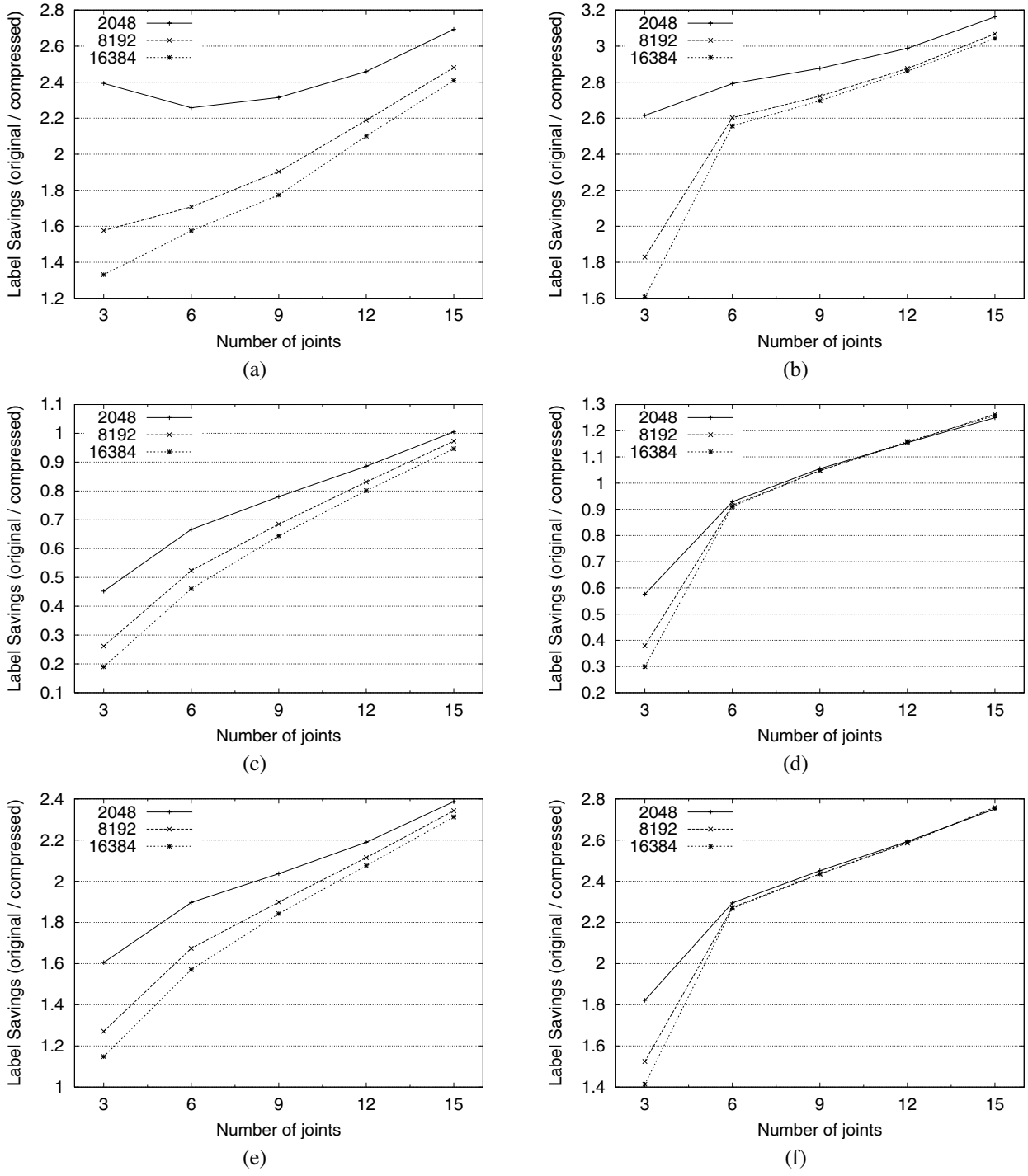



Fig. 17. Compression results for $\phi_a$ for robots with 3D workspace: roadmap constructed with the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function (a, c, e), and roadmap constructed with the $\mathcal{D}^{\mathcal{W}}_{\mathcal{F}}$ distance function. The neighborhoods are region-growing in (a) and (b), "union" hierarchical in (c) and (d), and "best" hierarchical in (e) and (f).

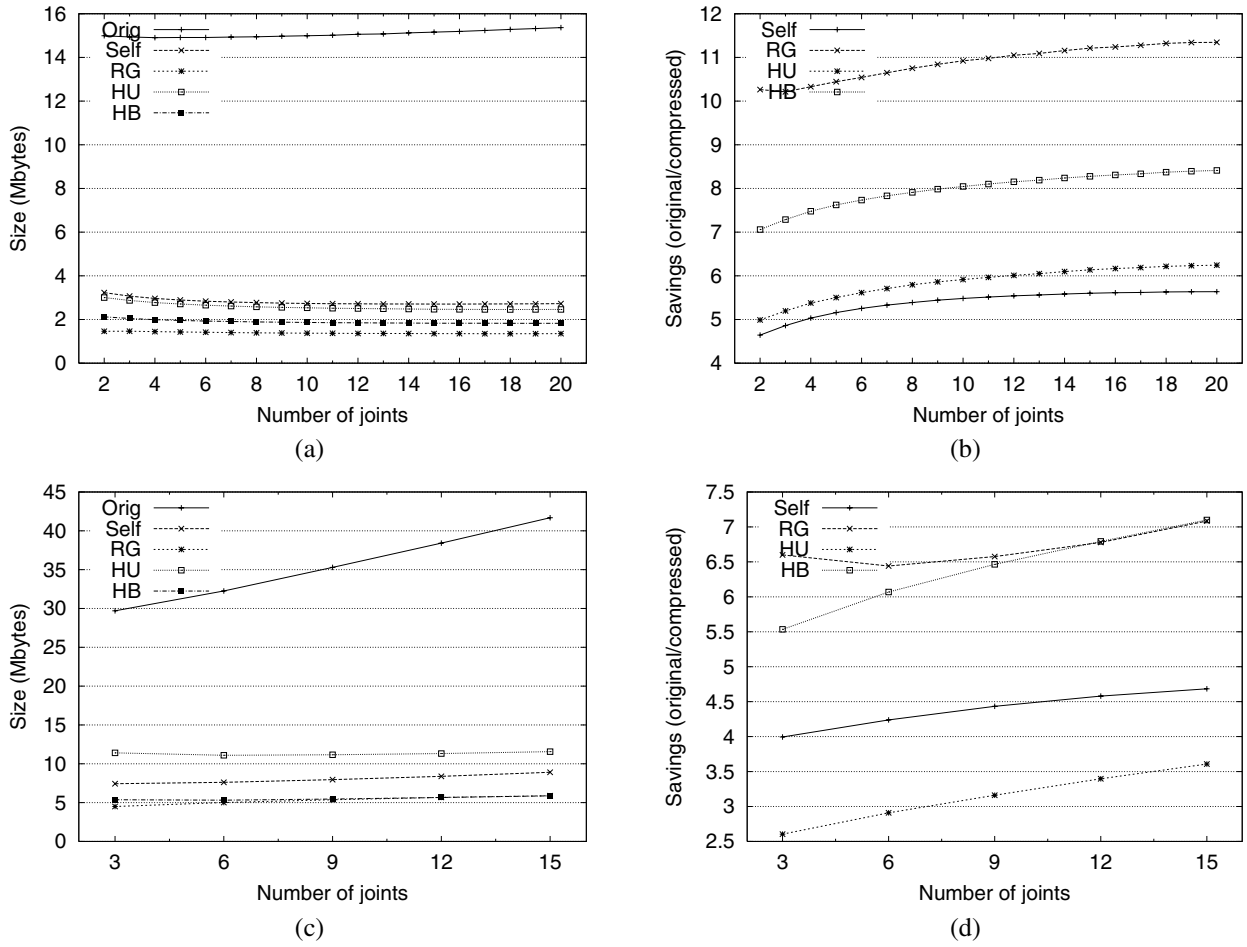



Fig. 18. File sizes and compression ratios for $\phi_n$ and a 16,384-node roadmap with efficient label encodings and different neighborhood models. The results for planar robots are in (a) and (b), and for 3D workspace robots in (c) and (d).

**Table 3. Definition of the Labels Used on Figures for the Different Encoding Models**

| Label | Description |
|---|---|
| Orig | Naïve encoding of the labels |
| Self | Efficient encoding of the labels only |
| RG | Region-growing neighborhood model with efficient label encoding |
| HU | "Union" hierarchical neighborhood model with efficient label encoding |
| HB | "Best" hierarchical neighborhood model with efficient label encoding |

neighborhoods, in which a reference number must be stored with each workspace cell, and the hierarchical neighborhoods, in which the reference for each cell is implicit in the tree structure.

Figure 19 shows the compression results for $\phi_a$ for planar robots without joint limits. The roadmap for this case has 16,384 nodes and was constructed using the $\mathcal{D}_2^{\mathcal{W}}$ distance function. The compression results here follow the results obtained for the savings in the number of labels using the different neighborhood models. Here, again, the neighborhoods defined by region-growing have the best compression performance, *reaching a savings factor of more than 70.*

The situation is slightly different for robots with joint limits. First, even for roadmaps constructed using the same distance function, $\phi_a$ is smaller for robots with joint limits than for those without. This difference can be seen by comparing Figures 20(a) and 19(a), where the uncompressed form of $\phi_a$ for robots with joint limits is about one quarter the size of that for robots without joint limits. The savings achieved using the different compression models are also lower, as shown in Figure 20(b). The region-growing neighborhoods still perform best, *reaching a compression ratio of nearly 25.*

This trend continues in Figures 20(c) and (d). In this case, the roadmap was constructed with 16,384 nodes using the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function. Using this distance function, the average swept volume for each arc is lower, resulting in less space required to store $\phi_a$ using the naïve encoding. The corresponding compression ratios are also smaller, although the region-growing neighborhoods still perform best.

The compression picture is more mixed for the robots with a 3D workspace. An example for this is shown in Figure 21, which shows the final file sizes for 16,384-node roadmaps computed using two different distance functions. For these cases, the size of the file for the naïve encoding is not available, since it is too large. In the case with the roadmap constructed with the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function, the best performance was given by two of the neighborhood models: region-growing and "best" hierarchical. On the other hand, for the roadmap constructed with the $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance function, the region-growing neighborhoods perform best.

### *6.3. Roadmap Enhancement*

In this section we discuss results from using our enhancement techniques on the roadmaps, including examining the minimum cut set of the roadmap represented as an unweighted graph and the $\rho$-robustness property of the roadmaps.

#### *6.3.1. Enhancement Based on Minimum Cut Sets*

The purpose of using a minimum cut set algorithm is to find clusters of nodes in the roadmap that are not well connected to the rest of the roadmap. For our roadmaps, we define the roadmap to be well connected if the number of arcs in the minimum cut set is equal to the minimum number of arcs connected to any node in the roadmap. Using this definition, it turns out that most of the roadmaps are already well connected, particularly when the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance function is used to construct the roadmap. There are two primary exceptions to this. One exception occurs for robots with two or three joints and tends to be more prevalent for roadmaps with more nodes. An explanation for this is that for low-dimensional C-spaces, the samples are closer together and therefore more readily form clusters.

The other exception to well-connected roadmaps is when the roadmap construction process fails to produce a roadmap with a single connected component; for this situation, the minimum cut set procedure is a useful tool for ensuring that the roadmap does consist of a single component. We have observed two cases for which this happens. The first of these is for planar robots and roadmaps constructed using the $\mathcal{D}_2^{\mathcal{W}}$ distance function. This case tends to occur for robots with 14 or more joints, and the cluster tends to consist of the samples drawn uniformly from the range of the first joint, whereas the positions of the other joints are fixed. These samples correspond to a dense sampling of a line in the C-space, and for robots with more joints, these samples tend to be "far" from the other samples taken uniformly at random relative to other samples on the line. The second case in which the roadmap did not consist of a single component occurred for one of the 3D workspace robots and the roadmap constructed using the $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance function. This case was also more prevalent for robots with more joints and for roadmaps with more nodes.

#### *6.3.2. Enhancement Based on $\rho$-Robustness*

As described in Section 5.2, $\rho$-robustness provides a quantitative measure of the robustness of the roadmap to the introduction of obstacles into the workspace. A roadmap can be made $\rho$-robust by adding arcs to the roadmap so that no obstacle of size $\rho$ can disconnect the roadmap.

We begin by evaluating the $\rho$-robustness of typical roadmaps with respect to the distance functions used in their construction. In Section 3, we conjectured that distance functions that accurately reflect the swept volume for a path between two configurations will yield more robust roadmaps. This is demonstrated in Figure 22. In particular, for these results we used the value of $\rho = 1$, and counted the number of single-cell obstacles that could disconnect the roadmap. (This was determined using a non-recursive version of the DFS-based algorithm in Tarjan (1972) to find strongly connected components.) The labels in these graphs for the distance functions follow the notation in Table 2. In Figure 22(b) we excluded the cases for the $\mathcal{D}_{\mathcal{F}}^{\mathcal{W}}$ distance function for which the initial roadmap was not connected. For comparison, note that the number of nonempty cells in the workspace is about 15,000 for planar robots, and 300,000 for 3D robots. The data

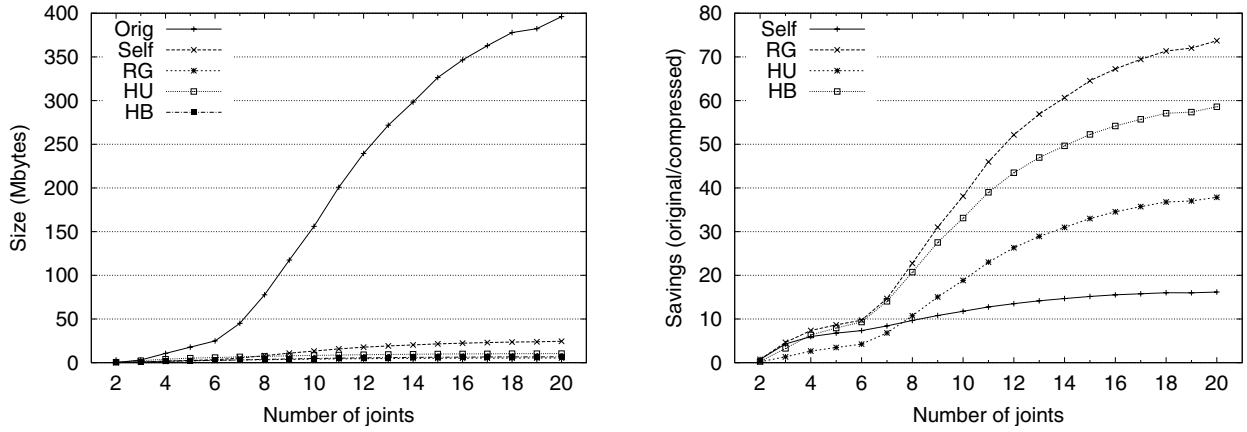



Fig. 19. File sizes and compression ratios with efficient label encodings and different neighborhood models for $\phi_a$ for a 16,384-node roadmap for planar robots without joint limits.

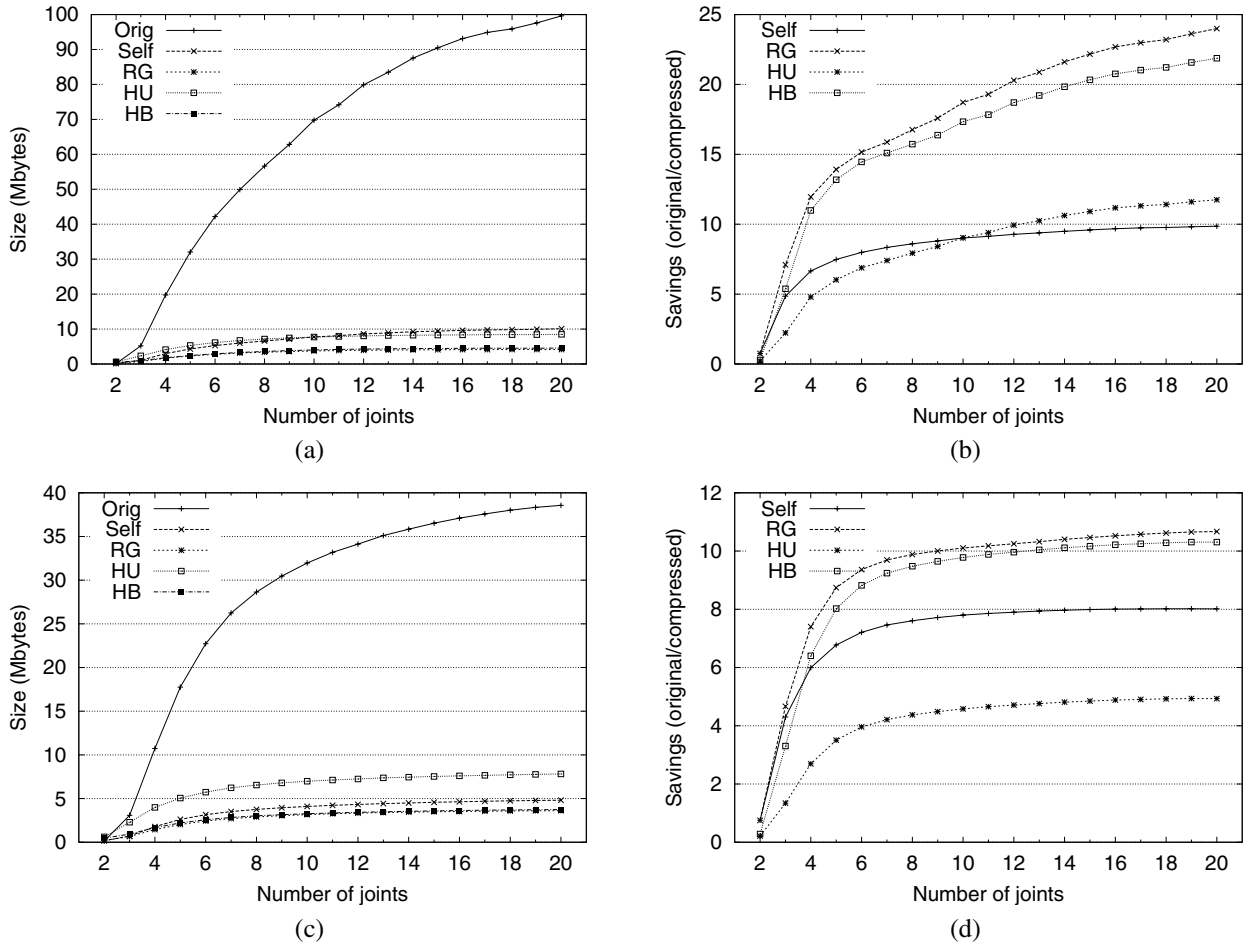



Fig. 20. File sizes and compression ratios with efficient label encodings and different neighborhood models for $\phi_a$ for a 16,384-node roadmap for planar robots with joint limits.

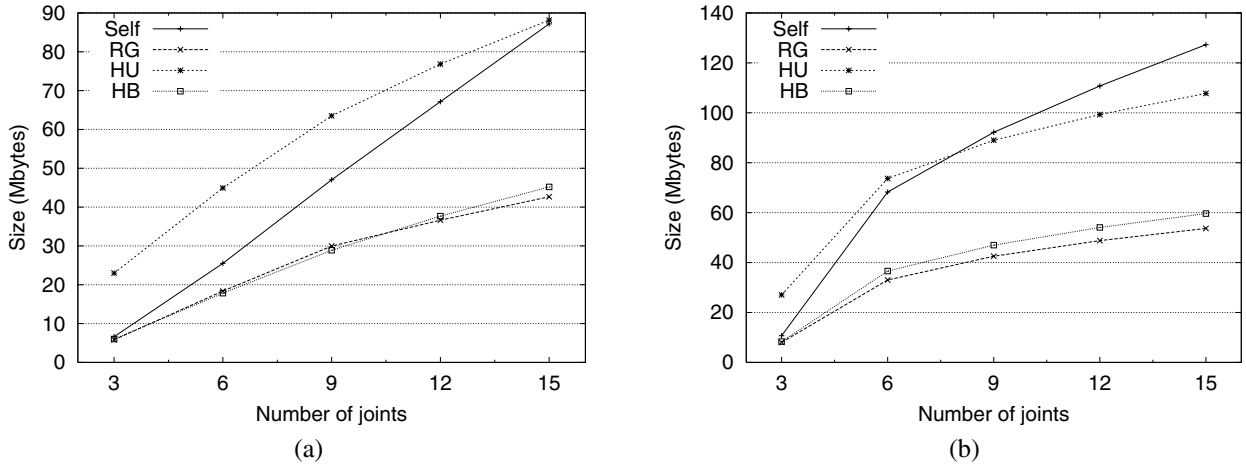



Fig. 21. Compressed file sizes for $\phi_a$ and a 16,384-node roadmap for 3D workspace robots. The roadmaps were constructed with the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function (a) and the $\mathcal{D}^{\mathcal{W}}_{\mathcal{F}}$ distance function (b).

in Figure 22 support our conjecture that those distance functions for which the average swept volume associated with each arc is higher result in less robust roadmaps (i.e., there are more single-cell obstacles that can disconnect the roadmap).

Next, we examine the number of arcs added to the roadmap during the $\rho$-robustness enhancement phase (described in Section ). Figure 23 shows the number of arcs added for each pass over the workspace and the resulting average connectivity of the roadmap measured in arcs per node. We tested planar robots with 2–10 joints, performing five passes over the workspace, and we tested 3D workspace robots with 3–15 joints, performing three passes. The robot type for the 3D workspace robots is that shown in Figure 10(a). As can be seen in Figure 23, we add most of the arcs in the first pass. An explanation for this is that, in the first pass over the workspace, we find most of the "fragile" pieces of the roadmap that can be easily disconnected. The result after processing is a near-linear dependence between the number of arcs per node and the number of joints.

In the remainder of this section, we focus on robots with a 3D workspace such as those shown in Figure 10(a), which we will refer to as type A, and in Figure 10(b), which we will refer to as type B. We will also examine the effect of using the manipulability measure to bias the sampling distribution in the C-space. For each robot, we construct a roadmap and perform $\rho$-robustness enhancement for $\rho = 3$. During the enhancement, when a cube of size $\rho$ disconnects the roadmap, we limit the number of connection attempts between pairs of roadmap components to 100 (we used larger limits for the earlier examples). The effect of such a low limit on the number of connection attempts is to reduce the number of cells for which we can enhance the robustness of the roadmap. It also reduces the computation time needed for enhancement. The number of cells in the workspace that can disconnect the roadmap before and after enhancement are recorded in the table under the column *Breaks*.

We tested three sets of roadmaps, one set with the samples concentrated in regions of low manipulability, one with the samples concentrated in regions of high manipulability, and one unbiased set. To take into account the joint limits in the manipulability, we declared the manipulability to be zero if any joint of the robot was within a factor of 0.016 from a joint limit. For each roadmap, we took 2048 samples and used the $\mathcal{D}^{\mathcal{W}}_{m2}$ distance function to select the arcs. Except where stated otherwise, each node was connected to at least its five nearest neighbors. The results for the robot types A and B are shown in Tables 4 and 5, respectively.

The effects of using manipulability to bias the sampling are interesting. For both robot types, the roadmaps created with nodes biased toward high manipulability were more robust after enhancement, and the enhancement required fewer arcs to produce this result. On the other hand, before enhancement, the roadmaps created with nodes biased toward low manipulability tended to be more robust, particularly for robots with more joints.

Our current algorithm for processing a roadmap to increase its $\rho$-robustness can require significant computation time. The time required for the test procedure, which finds the number of cells that result in the roadmap becoming disconnected, is proportional to the product of the number of nonempty cells in the workspace and the size of the roadmap. The procedure that repairs the roadmap for each of the cells that disconnect the roadmap adds the cost of evaluating the arcs needed to reconnect the pieces of the roadmap. This last step

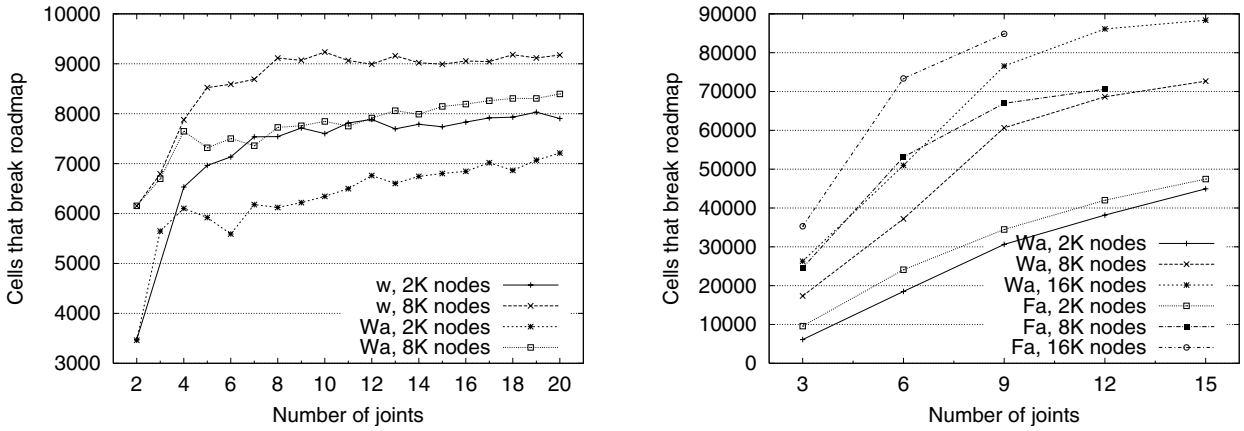



Fig. 22. Test for $\rho$-robustness for roadmaps constructed with different distance functions: (a) planar robots and (b) 3D workspace robots.

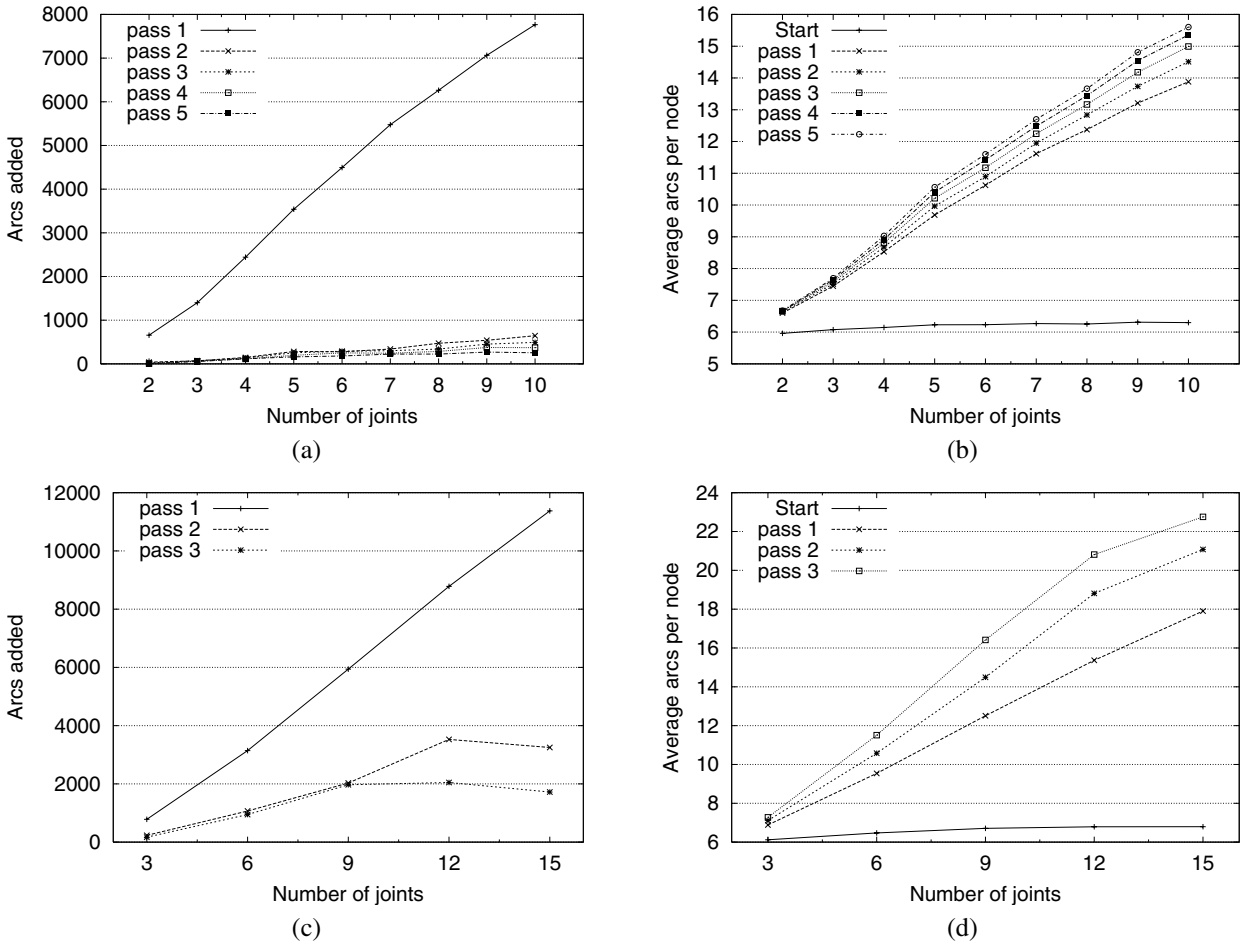



Fig. 23. Arcs added and resulting connectivity at different steps during processing for $\rho$-robustness for 2048-node roadmaps constructed with the $\mathcal{D}_{m2}^{\mathcal{W}}$ distance functions: (a) and (b) planar robots, and (c) and (d) 3D workspace robots.

**Table 4. Results for $\rho$-robustness for a 2048-node Roadmap Created for the Type A Robots**

| | Low Manipulability | | | High Manipulability | | |
|---|---|---|---|---|---|---|
| | Breaks | | Arcs | Breaks | | Arcs |
| Joints | Before | After | Added | Before | After | Added |
| 3 | 6,476 | 55 | 1,218 | 6,065 | 61 | 1,113 |
| 6 | 17,073 | 522 | 5,454 | 19,406 | 266 | 4,798 |
| 9 | 29,449 | 1,265 | 10,298 | 32,058 | 1,081 | 9,163 |
| 12 | 36,592 | 3,303 | 14,153 | 42,585 | 1,753 | 12,915 |
| 15 | 41,426 | 5,921 | 15,777 | 47,908 | 3,685 | 14,960 |

**Table 5. Results for $\rho$-robustness for a 2048-node Roadmap Created for the Type B Robots**

| | Low Manipulability | | | No Manipulability | | | High Manipulability | | |
|---|---|---|---|---|---|---|---|---|---|
| | Breaks | | Arcs | Breaks | | Arcs | Breaks | | Arcs |
| Joints | Before | After | Added | Before | After | Added | Before | After | Added |
| 3 | 5893 | 0 | 1022 | 6233 | 0 | 999 | 5198 | 4 | 863 |
| 6 | 18,426 | 86 | 5047 | 18,347 | 87 | 4639 | 18,005 | 53 | 4143 |
| 9 | 32,505 | 315 | 9156 | 33,853 | 443 | 8478 | 32,667 | 213 | 7392 |
| 12 | 39,663 | 627 | 11,777 | 42,014 | 547 | 11,024 | 44,903 | 446 | 10,160 |
| 15 | 45,863 | 1010 | 13,774 | 51,454 | 880 | 12,863 | 54,543 | 765 | 12,074 |

dominates the computation time for enhancing the $\rho$-robustness of a roadmap.

For example, the test procedure, which finds the number of cells that result in the roadmap becoming disconnected, took 35 min for a 2048-node roadmap for the 3D workspace robots. The times for the 8192- and 16,384-node roadmaps were 3.5 and 8.3 h, respectively. The test times were shorter for the planar robots, since there are fewer cells to test (around 15,000 instead of 300,000). The program that improves the robustness of the roadmap can easily take several days or more, particularly for larger roadmaps and robots with more joints.

### *6.4. Planning Examples*

The implementation of our planner tests for the existence of a path between two selected nodes in the roadmap modified to take into account obstacle constraints. The process for finding a path for fixed-base robots consists of two steps: modifying the roadmap to take into account the obstacles and searching the modified roadmap for a path. The search process itself is fast, taking of the order of tens of milliseconds to search the roadmaps that we have tested. This is true for both robots with 2D workspaces and robots with 3D workspaces. The path smoothing that we apply to the output of the planner takes a variable amount of time, spending more time when the robot is closer to the obstacles along the path and taking more time for robots with more joints.

Modifying the roadmap to take into account the obstacles is slower, and the time required depends on the neighborhood model used to encode $\phi$. The fastest roadmap update times were for $\phi$ encoded with no neighborhood model. The "best" hierarchical neighborhoods had the second fastest update times, and the region-growing neighborhoods had the third fastest times. Nevertheless, the update times for all three of these were close in magnitude. This was not the case for the "union" hierarchical neighborhoods, whose performance was around an order of magnitude worse. This performance gap may be due to our implementation of the decoding procedure for the hierarchical neighborhoods, which decodes $\phi$ for a given cell from the bottom of the hierarchy to the top. It may be more efficient for this case to process the cells from the top of the hierarchy down.

Figure 24 shows an example path generated by our planner for a 20-joint planar robot. In this example, roughly half of the workspace is occupied by the obstacle region. The roadmap used for this plan had 8192 nodes. The time required to process the obstacles shown in Figure 24 was of the order of a few seconds for 2048- and 8192-node roadmaps and $\phi$ encoded using no neighborhood model, using the region-growing neighborhood model, and using the "best" hierarchical neighborhood model. On the other hand, using the "union" hierarchical neighborhood model, processing the obstacles required on average 41 s for the 2048-node roadmap and 167 s for the 8192-node roadmap. These times did not vary much

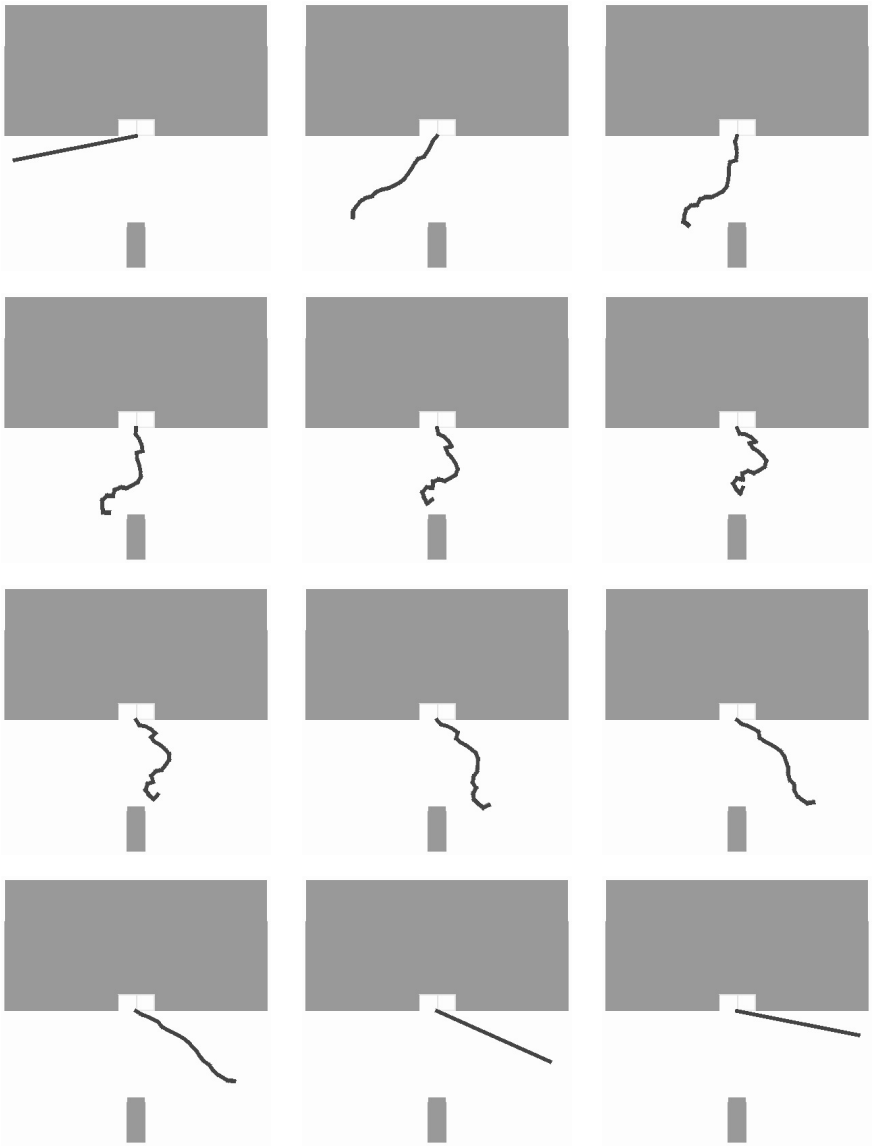


Fig. 24. An example plan for a 20-joint planar robot with joint limits. The sequence of steps is from left to right, top to bottom.

for robots with different numbers of joints. For the example path shown in Figure 24, path smoothing took around 12 s to perform. Therefore, for real-time applications, there are two viable approaches: decode the representation prior to planning (this would allow, for example, the representation to be stored using a small amount of ROM, and then unpacked into RAM at planning time), or use one of the three faster encoding schemes.

Figure 25 shows an example path generated by our planner for a six-joint robot with a 3D workspace and a 2048-node roadmap. We observed a pattern similar to the planar robot case for the processing time required for the different neighborhood models when processing the obstacles in the workspace. For the obstacles shown in Figure 25, the processing time was less than 1 s for the neighborhood models of none, region-growing, and "best" hierarchical, with exceptions for the 12- and 15-joint robots, where the time increased to 2.8 s for the 8192-node roadmaps. The times for the "union" hierarchical neighborhood models were much slower, requiring around 10.5 s for the 2048-node roadmaps and around 43 s for the 8192-node roadmaps. In addition, the times tended to increase with the number of joints, which correlates with the increase in the size of the representation of $\phi$ with the number of joints.

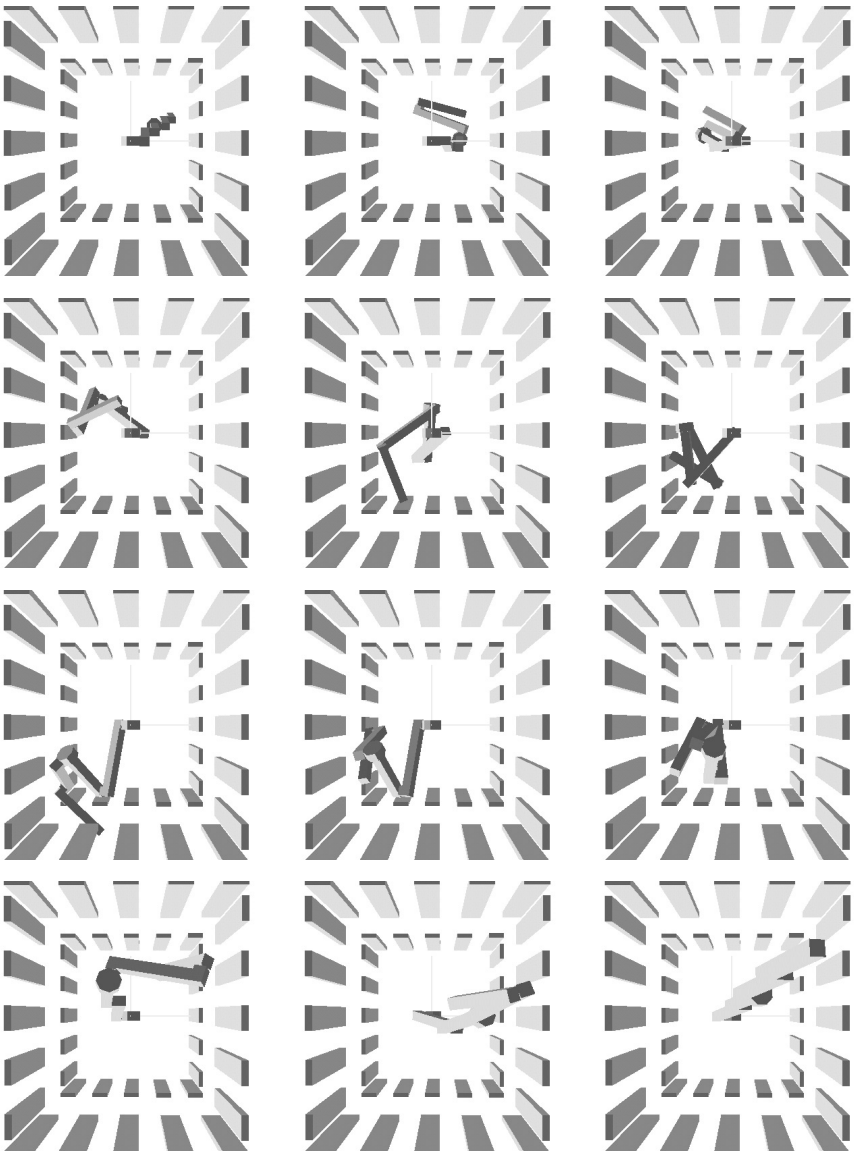


Fig. 25. An example plan for a 6-joint robot with a 3D workspace. The sequence of steps is from left to right, top to bottom.

## 7. Conclusion

We have presented a new method for generating collision-free paths for robots operating in changing environments. We use a preprocessing stage that comprises three steps. First, a roadmap is constructed using random sampling of the C-space. We have used both uniform sampling and importance sampling based on manipulability. Secondly, we encode the mapping from cells in the robot's workspace to nodes and arcs in the C-space roadmap. The redundancy in this mapping is exploited to construct a significantly compressed representation. This mapping and the ability to represent it efficiently is at the heart of our new approach. Thirdly, we enhance the C-space roadmap using the minimum cut set of the roadmap and $\rho$-robustness, a measure that we have defined to capture the robustness of the roadmap to the introduction of obstacles into the robot workspace.

We have presented experimental results for a planner based on our framework. We have examined serial-link manipulators for both 2D and 3D workspaces. For roadmap construction, we have examined the effect of the geometry of the robot on the sampling of its C-space. For the mapping from the workspace to the roadmap, we have examined the size of the mapping, the computation time required to generate the mapping, and the improvement in the size of the mapping gained by using a more efficient encoding. For roadmap

enhancement, we have examined the amount of improvement we can expect using either of the two techniques for a given robot and roadmap. We have concluded our experimental results with some planning examples.

We believe that this work makes an important step in the direction of constructing optimal representations for real time path planning, but there is much work that remains to be done. The approach presented here has a number of limitations. It is not capable of any sort of fine motion planning, and it would not be able to cope with the narrow passage problem that plagues many randomized approaches. Therefore, to solve such problems, it would be necessary to augment our system, possibly by using our current approach to generate gross motion plans, and then incorporating a refinement stage to produce fine motion plans or to cope with narrow passages. We leave these problems for our future work. Furthermore, we have not investigated any sort of incremental methods for updating the representation dynamically if the environment undergoes a series of changes. Such methods are very popular in the computational geometry literature, and we hope to investigate how these methods could be applied to our system. We are also interested in investigating how our approach could be used to plan the motion of mobile manipulators in conjunction with sensor-based exploration tasks. In sum, we believe that the work presented here represents one component in an intelligent robotic system, and that this component can certainly be further improved.

## Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Award No. CCR-0085917 and IIS-0083275.

## References

Abrams, S., and Allen, P. K. 1995. Swept volumes and their use in viewpoint computation in robot work-cells. In *Proceedings of IEEE Conference on Robotics and Automation,* pp. 188–193.

Abrams, S., Allen, P. K., and Tarabanis, K. A. 1993. Dynamic sensor planning. In *Proceedings of IEEE Conference on Robotics and Automation,* pp. 605–610.

Ahuactzin, J.-M., and Gupta, K. 1999. The kinematic roadmap: A motion planning based global approach for inverse kinematics of redundant robots. *IEEE Transactions on Robotics and Automation* 15(4):653–669.

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

Ahuactzin, J.-M., Mazer, E., and Bessière, P. 1995. Fondements mathématiques d'algorithme "fil d'Ariane". *Revue d'Intelligence Artificielle* 9(1):7–34.

Ahuactzin, J.-M., Talbi, E.-G., Bessière, P., and Mazer, E. 1992. Using genetic algorithms for robot motion planning. In *European Conference on Artificial Intelligence*, pp. 671–675.

Amato, N. M., and Wu, Y. 1996. A randomized roadmap method for path and manipulation planning. In *Proceedings of IEEE Conference on Robotics and Automation* Vol. 1, pp. 113–120.

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

Amato, N. M., Bayazit, O. B., Dale, L. K., Jones, C., and Vallejo, D. 2000. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Transactions on Robotics and Automation* 16(4):442–447.

Anshelevich, E., Owens, S., Lamiraux, F., and Kavraki, L. E. 2000. Deformable volumes in path planning applications. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 2290–2295.

Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. 1996. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software* 22(4).

Barraquand, J., and Latombe, J.-C. 1991. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research* 10(6):628–649.

Bell, T. C., Cleary, J. G., and Witten, I. H. 1990. *Text Compression*, Prentice Hall, Englewood Cliffs, NJ.

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

Blackmore, D., and Leu, M. C. 1990. A differential equation approach to swept volumes. In *Proceedings of Rensselaer's 2nd International Conference on Computer Integrated Manufacturing*, pp. 143–149.

Bohlin, R., and Kavraki, L. E. 2000. Path planning using lazy PRM. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 521–528.

Boor, V., Overmars, M. H., and van der Stappen, A. F. 1999. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 1018–1023.

Boussac, S., and Crosnier, A. 1996. Swept volumes generated from deformable objects application to nc verification. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 1813–1818.

Brooks, R., and Lozano-Pérez, T. 1983. A subdivision algorithm in configuration space for findpath with rotation. In *International Joint Conference on Artificial Intelligence*, pp. 799–806.

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

Elias, P. 1975. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory* 21(2).

Furht, B., Greenberg, J., and Westwater, R. 1997. *Motion Estimation Algorithms for Video Compression*. Kluwer Academic, Boston.

Guibas, L. J., Holleman, C., and Kavraki, L. E. 1999. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*.

Han, L., and Amato, N. M. 2000. A kinematics-based probabilistic roadmap method for closed chain systems. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*.

Hankerson, D., Harris, G. A., and Johnson Jr, P. D. 1998. *Introduction to Information Theory and Data Compression*. Discrete Mathematics and its Applications. CRC Press, New York.

Holleman, C., and Kavraki, L. E. 2000. A framework for using the workspace medial axis in PRM planners. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 1408–1413.

Holleman, C., Kavraki, L. E., and Warren, J. 1998. Planning paths for a flexible surface patch. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 21–26.

Horsch, T., Schwarz, F., and Tolle, H. 1994. Motion planning with many degrees of freedom—random reflections at c-space obstacles. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 3318–3323.

Hsu, D., Latombe, J.-C., and Motwani, R. 1999. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications* 9(4/5):495–512.

Hunter, G. M., and Steiglitz, K. 1979. Operations on images using quad trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1(2):145–153.

Hwang, Y. K., and Ahuja, N. October 1988, Path planning using a potential field representation. Technical Report UILU-ENG-8-2251, University of Illinois.

Kambhampati, S., and Davis, L. S. 1986. Multiresolution path planning for mobile robots. *IEEE Journal of Robotics and Automation* 2(3):135–145.

Kaufman, A., and Shimony, E. 1986. 3D scan-conversion algorithms for voxel-based graphics. In *Proceedings of 1986 Workshop on Interactive 3D Graphics*. ACM, New York, pp. 45–76.

Kavraki, L. E., and Latombe, J.-C. 1994. Randomized preprocessing of configuration space for fast path planning. In *Proceedings of IEEE Conference on Robotics and Automation* Vol. 3, pp. 2138–2145.

Kavraki, L. E., and Latombe, J.-C. 1998. Probabilistic roadmaps for robot path planning. In *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, K. Gupta and P. del Pobil, eds. Wiley, New York, pp. 33–53.

Kavraki, L. E., Kolountzakis, M. N., and Latombe, J.-C. 1996. Analysis of probabilistic roadmaps for path planning. In *Proceedings of IEEE Conference on Robotics and Automation* Vol. 4, pp. 3020–3025.

Kavraki, L. E., Lamiraux, F., and Holleman, C. 1998. Towards planning for elastic objects. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*.

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

Khatib, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research* 5(1):90–98.

Kindel, R., Hsu, D., Latombe, J.-C., and Rock, S. 2000. Kinodynamic motion planning amidst moving obstacles. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 537–543.

Koditschek, D. E. 1989. Robot planning and control via potential functions. In *The Robotics Review 1*, pp. 349–367. MIT Press, Cambridge, MA.

Kuffner Jr, J. J., and LaValle, S. M. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 995–1001.

LaValle, S. M., and Kuffner Jr, J. J. 1999. Randomized kinodynamic planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 473–479.

LaValle, S. M., and Kuffner Jr, J. J. 2000. Rapidly-exploring random trees: Progress and prospects. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*.

LaValle, S. M., Yakey, J. H., and Kavraki, L. E. 1999. A probabilistic roadmap approach for systems with closed kinematic chains. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 1671–1676.

Leven, P., and Hutchinson, S. 2000. Toward real-time path planning in changing environments. in *Proceedings of Workshop on Algorithmic Foundations of Robotics*.

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

Lozano-Pérez, T. 1983. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*

Ma, L., Jiang, Z., and Chan, K.Y.T. 2000. Interpolating and approximating moving frames using b-splines. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, pp. 154–164.

Matula, D. W. 1993. A linear time $2+\epsilon$ approximation algorithm for edge connectivity. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pp. 500–504.

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

McLean, A., and Mazon, I. 1996. Incremental roadmaps and global path planning in evolving industrial environments. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 101–107.

Mehrandezh, M., and Gupta, K. 2002. Simultaneous path planning and free space exploration with skin sensor. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 3838–3843.

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

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

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

Overmars, M. H., and Švestka, P. March 1995. A paradigm for probabilistic path planning. Technical Report UU-CS-1995-22, Utrecht University.

Rosenfeld, A., and Kak, A.C. 1982. *Digital Picture Processing*. Academic Press, New York, second edition.

Samet, H. 1984. The quadtree and related hierarchical data structures. *Computing Surveys* 16(2):187–260.

Schwartz, J. T., Sharir, M., and Hopcroft, J., eds. 1987. *Planning, Geometry, and Complexity of Robot Motion*. Ablex, Norwood, NJ.

Švestka, P., and Overmars, M. H. 1995. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 1631–1636.

Trjan, R. E. June 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1(2).

Vallejo, D., Jones, C., and Amato, N. M. October 1999. An adaptive framework for 'single shot' motion planning. Technical Report TR99-024, Department of Computer Science, Texas A&M University, College Station, TX.

van den Bergen, G. 1997. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools* 2(4):1–14.

van den Bergen, G. 1999. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools* 4(2):7–25.

Wilmarth, S. A., Amato, N. M., and Stiller, P. F. 1999. 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*, pp. 173–180.

Witten, I. H., Moffat, A., and Bell, T. C. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, CA, second edition.

Xavier, P.G. 1997. Fast swept-volume distance for robust collision detection. In *Proceedings of IEEE Conference on Robotics and Automation*, pp. 1162–1169.

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

Yu, Y., and Gupta, K. 1999. Sensor-based roadmaps for motion planning for articulated robots in unknown environments: Some experiments with an eye-in-hand system. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*.