Building and Navigating a Network of Local Minima *

Seung-Woo Kim and *Daniel Boley*†
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, Minnesota 55455

Abstract

We present a novel method that constructs and navigates a network of local minima of potential fields defined over multi-dimensional spaces. Though motivated by problems of motion planning for robotic manipulators, similar techniques have been proposed for use in other domains such as molecular chemistry and drug design. The method is based on building a roadmap of paths connecting local minima of a potential function. The novel approach consists of an up-hill search strategy used to climb out of local minima and find new nearby local minima, without doubling back on previous local minima. With this up-hill search strategy, one can find local minima otherwise difficult to encounter, and one can focus the search to specific local minima and specific directions from those local minima. The construction of the roadmap can be done in parallel with very little communication. We present extensive simulation results.

1 Introduction and Background

We present a novel fast method that constructs and navigates a network of local minima of potential fields defined over multi-dimensional spaces. The speed of our method comes from the parallelism inherent in the search process and from an up-hill search strategy which leads to a more systematic search of the free space. Such methods have a wide variety of applications from motion planning to chemical analysis at the molecular level and even to drug design. The ability to plan paths quickly is important to make motion planning useful in application areas, such as industrial robotics, teleoperation [13], control of redundant

*This research was supported in part by NSF grants CCR-9628786, INT-9726332, and IIS-9811229.
†Contact author. Electronic mail: `boley@cs.umn.edu`.

robots [10], etc. To keep the exposition as simple as possible, this paper is devoted to the presentation of our methods in the context of the motion planning problem for a robotic manipulator. This problem consists of finding a path for a manipulator from a given initial to a given goal configuration while avoiding obstacles present in the environment. Many algorithms have been developed [12], but most are rarely used in practice because of their computational complexity [7].

Many existing motion planning methods operate in the space of all possible configurations ("C-space"), as opposed to the physical work space ("W-space"). While the W-space has dimension 2 or 3 depending on the physical space traversed by the manipulator, the C-space has dimension equal to the number of degrees of freedom (*dof*) of the robot, Many algorithms operate by computing the set of infeasible configurations (obstacles) and then searching the remaining C-space for feasible paths. To make the methods computationally tractable, various devices have been used. In [3], the C-space was discretized into cells, and the cells that were totally free of obstacles were searched for paths. Unfortunately, small "passageways" between obstacles often did not correspond to any totally free cell and no path could be found through such passageways. Artificial potential fields were used in [3] in order to avoid collisions and/or to focus the search toward the goal. A genetic algorithm approach combined with landmarks was used in [14], where landmarks were placed until a local planner could generate a path. Other methods make assumptions on the type of robot, e.g. taking advantage of the symmetry of the workspace [2], or use a coarse discretization of C-Space.

A random search of the C-space was used in [8, 9], where a significant amount of preprocessing was used to obtain acceptable performance. This approach shares many of the features of the methods proposed in the present paper, including the use of an artificial potential field over the continuous C-space and the construction of a roadmap by starting at many random positions and exploring in random directions. The artificial potential field was designed to keep the robot away from the obstacles. The roadmap was a graph consisting of vertices corresponding to feasible configurations and edges corresponding to feasible straight-line paths connecting those configurations. Our method also uses a randomized search with an artificial potential field, but the generated roadmap has a much more modest

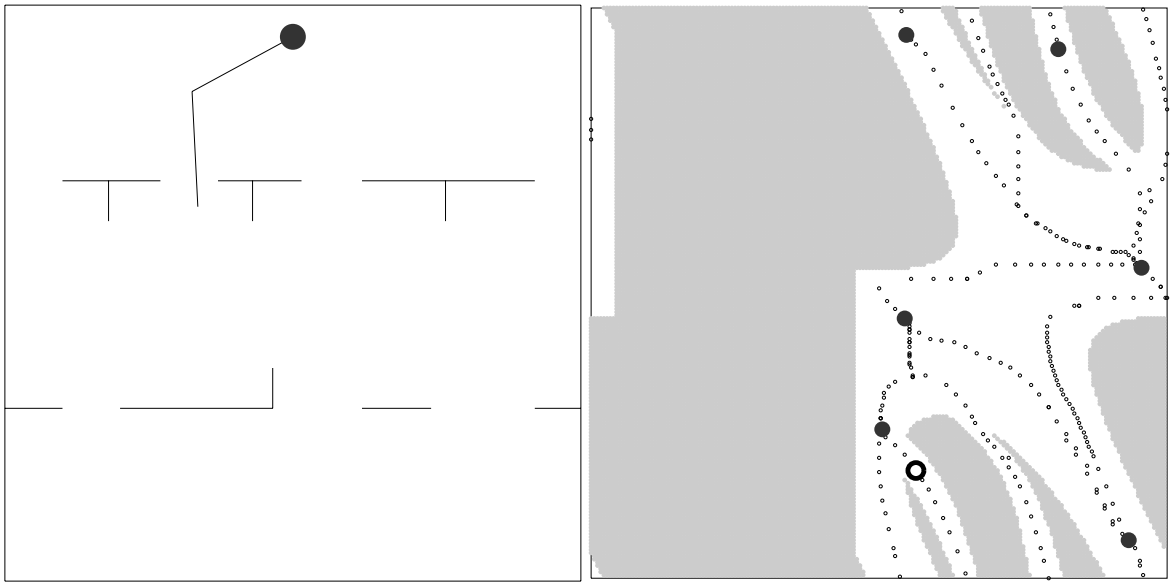
size, since the vertices are limited to the local minima in the potential field and the edges are not limited to straight-line paths. Space does not permit a complete history of previous work in this area, but an excellent summary of previous work can be found in [9, 6].

We propose a method to connect any given initial and goal configurations for a manipulator in a static environment using a roadmap in the C-space. As in [8], we first build a roadmap, which is a connected graph that in some sense covers the entire free space in the C-space. The nodes or vertices in the roadmap are the local minima, and the edges are the paths connecting those local minima. To connect any two arbitrary given configurations, we first connect them to the roadmap, and then search the roadmap for a path between them.

Our method enjoys many favorable properties. Since the vertices are local minima of the potential function, the number of vertices in the roadmap is much more limited than that obtained by a more arbitrary placement as done in [8]. A major novelty is our up-hill search strategy, which constructs paths that tend to follow potential wells (i.e. "valley floors"). This means that we can follow the curved obstacle boundaries in C-space and also find narrow passageways in the free space more easily. As in [8], each segment of the roadmap can be computed independently and in parallel, leading to a high degree of parallelism. Because the vertices are local minima, connecting an arbitrary goal configuration to the roadmap can be accomplished by a very efficient gradient descent method. If the resulting local minimum is not already on the roadmap, the up-hill search strategy is very effective in connecting it up. Furthermore, it will be seen that these paths tend to pass through saddle points in the potential function (see the Appendix), limiting the potential hill that must be climbed along each path. In the context of robot manipulators, this corresponds to staying as far from obstacles as possible. Lastly, the directed randomized parallel search process is effective in finding a large fraction of all the local minima in a short time. This partial graph of local minima is usually sufficient to "cover" the entire space in the sense that it is relatively easy to connect any new local minimum to the existing graph.

Figure 1 shows a 2 *dof* robot example. Figure (a) shows a W-space with the base at the black circle. Figure (b) is the corresponding C-space. The circles in (b) mark the local minima in the potential function, and the small dots mark the intermediate waypoints found by our up-hill search strategy. The hollow circle marks the local minimum very close to the

point corresponding to the configuration in (a).



(a) (b)

Figure 1: A 2 *dof* robot example. (a) shows a W-space with a 2 *dof* robot with the base at the black circle. (b) is the corresponding C-space with the origin at the lower-left corner and the x and y axes representing the first and second joints respectively. The gray regions are the regions of collision with obstacles. (b) also shows the local minima (large black circles) and the intermediate points (tiny dots) within the individual hyperplanes found by the procedure **UpStream** of Fig. 5. The hollow circle is the local minimum close to the point corresponding to configuration (a).

The rest of the paper is organized as follows. Section 2 gives an overview of the process of building a roadmap. Section 3 discusses the potential function and the gradient descent methods based on that potential function. We use our method for robots with many articulated joints in a static 2 dimensional environment as an example to illustrate the methods. Section 4 discusses the process of generating a roadmap using the novel up-hill search strategy. Section 5 discusses finding a collision free path using the generated roadmap. Section 6 shows our experimental results, followed by the conclusion in Section 7.

2 Building the Roadmap

The process of building a roadmap is summarized in Figure 2. Local minima are initially found by descending from randomly chosen feasible positions. From each such local minimum, the method climbs out of the potential well and explores "neighboring valleys" looking for neighboring local minima. This is accomplished using our novel up-hill search strategy. Parallelism is obtained by simultaneously starting this search from many randomly chosen positions. A master process collects all the partial paths generated and assembles the overall graph.

At the roadmap grows to cover the C-space, new local minima become harder and harder to find by a purely random process. Also, the need to make sure that all the existing local minima are connected becomes more critical. Hence at some point we switch to a more focused search in Phase II. In Phase II, we start from the more isolated local minima and explore up-hill in the direction toward the largest connected component in the existing roadmap, with the goal of connecting them.

Phase II is also used when connecting an arbitrary initial or goal configuration to the roadmap. If the local minimum reached by gradient descent from a given goal configuration is not already on the roadmap, Phase II is used to connect it up.

The main features of our method are the following. (a) We limit the situations where a random movement is necessary, (b) we use a potential field to guarantee collision-free paths, (c) we have a systematic strategy for climbing out of a local minimum to find a low energy path to other local minima, (d) we use Gauss-Newton directions to descend along shallow slopes very efficiently, and (e) the overall process easily decomposes for efficient parallel processing.

3 Gradient Descent using a Potential Function

The use of gradient descent methods requires the use of a potential function that has at least one derivative (the gradient). Using a more sophisticated method such as Gauss-Newton requires a potential function with at least two derivatives [1], even though the second

Phase I. Randomized Search.

Repeat until a given percentage of all local minima found are connected together in a large connected component:

1. Select a random starting position and descend to a local minimum $\boldsymbol{\theta}_0$.
2. Explore up-hill from that local minimum in a randomly chosen direction $\mathbf{u}_0$ by repeated use of the ***UpStream*** procedure to find a partial path to neighboring local minima.
3. Connect the new partial path to the existing graph of partial paths.

Phase II. Focused Search.

Repeat until all the local minima are connected:

1. Select an "isolated" local minimum.
2. Explore up-hill from that local minimum in a randomly chosen direction $\mathbf{u}_0$ among those directions leading toward the larger connected graph of minima, using repeated applications of the ***UpStream*** procedure.
3. Connect the new partial path to the existing graph of partial paths.

Figure 2: **GraphBuild** – create a graph (roadmap) of local minima and their connections. Each new combination of position $\boldsymbol{\theta}_0$ and direction $\mathbf{u}_0$ is discarded if already encountered.

derivative (the Hessian) is not computed explicitly. In our approach, we use a potential function that has at least two derivatives except at isolated points. The up-hill search strategy (procedure ***UpStream***) described in the next section is used to carry out a search in the neighborhood of those isolated points where the derivatives may not exist.

3.1 The Potential Function

We define the potential function over the C-space, though its computation is based on the W-space configuration. For simplicity, we limit our discussion to revolute joints, though a very similar formulation could be applied to translational joints. The C-space is a *dof*

dimensional space of joint angles $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_{dof-1})$, where $0 \leq \theta_i < 2\pi$ and $0 \leq i < dof$. The base joint angle θ_0 is set to 0 when the first link is pointing at 3'o'clock and increases counterclockwise. The other joint angles, θ_i, are set to 0 ($1 \leq i < dof$) when the joint is fully stretched and increases counterclockwise. The potential function prevents each joint from going through the fully folded position ($\theta = \pm\pi$).

We define a potential function designed to repel the individual robot links away from the obstacles and from each other. The potential function is expressed in the form of a non-linear least squares functional

$$E(\boldsymbol{\theta}) = \frac{1}{2} \sum_i r_i^2(\boldsymbol{\theta}), \tag{1}$$

where $E(\boldsymbol{\theta})$ is the potential , $r_i(\boldsymbol{\theta})$ are the individual potential functions for each link and each obstacle, defined over the m dimensional space $\boldsymbol{\theta}$. Each r_i is differentiable with respect to $\boldsymbol{\theta}$. The potential function consists of three parts, the potential between the robot links and obstacles, the potential between non-consecutive links, and the potential between consecutive links sharing a joint. We now describe the formulation of each of these components.

The potential resulting from the obstacles and robot links. This is the primary force affecting the robot because we wish to avoid collisions between the robot and the obstacles. In order to compute the potential for a given configuration, a pair of closest points is computed for each pair of link and obstacle segment. In Figure 3, let us denote the closest points between i-th link and j-th obstacle segment as $P_{i,j}$ and $Q_{i,j}$ respectively. Then the Euclidean distance between them is $d_{i,j} = \|P_{i,j} - Q_{i,j}\|_2$. Let $m \triangleq$ the number of links, $n \triangleq$ the number of obstacles, and $r_{i+j*m} \triangleq 1/d_{i,j}$. Then the overall potential E_O arising from the obstacles is

$$E_O = \frac{1}{2} \sum_{i=0}^{\text{BOUND}_O - 1} r_i^2 \tag{2}$$

where $\text{BOUND}_O = mn$.

The potential between non-consecutive links. This component in the potential is designed to prevent self collisions. The construction is analogous to the construction of the potential between a link and an obstacle. Let us denote $s_{i,j}$ as the reciprocal of the

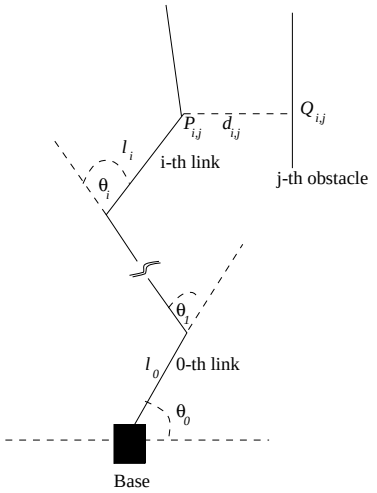


Figure 3: Computing potential fields. $P_{i,j}$ = the closest point between i-th link and j-th obstacle on i-th link, $Q_{i,j}$ = the closest point between i-th link and j-th obstacle on j-th obstacle, $d_{i,j}$ = the Euclidean distance between $P_{i,j}$ and $Q_{i,j}$, l_i = distance between the base of i-th link to $P_{i,j}$, θ_i = i-th joint angle.

distance between i-th link and j-th link, where $2 \leq i < m$ and $0 \leq j < i-1$. Then the potential between non-consecutive links is as follows:

$$E_{NC} = \frac{1}{2} \sum_{i=2}^{m-1} \sum_{j=0}^{i-2} s_{i,j}^2 = \frac{1}{2} \sum_{k=\text{BOUND}_O}^{\text{BOUND}_{NC}-1} r_k^2, \tag{3}$$

where, for convenience, we change the notation s to the common notation r, using the common indexing $r_k = s_{i,j}$ with $k = mn + (i-1)(i-2)/2 + j$, and where $\text{BOUND}_{NC} = \text{BOUND}_O + (m-2)(m-3)/2 + (m-2)$.

The potential between consecutive links. This component of the potential function is designed to limit the movement of the joints, so that two consecutive links cannot fold through each other. Hence, for revolute joints considered in this paper, the potential function must go to infinity as each joint angle approaches π. Therefore, E_C is computed as follows:

$$E_C = \frac{1}{2} \sum_{i=1}^{m-1} 1/\|\pi - \theta_i\|_2^2 = \frac{1}{2} \sum_{k=\text{BOUND}_{NC}}^{\text{BOUND}_{all}-1} r_k^2 \tag{4}$$

where $1 \leq i < m$, and where we again adopt a common indexing with $r_k = 1/\|\pi - \theta_i\|_2$,

$k = (m-2)(m-3)/2 + (m-2) + (i-1)$, and $\text{BOUND}_{all} = \text{BOUND}_O + \text{BOUND}_{NC} + m - 1$.

No potential is applied to θ_0 because the base of the robot is allowed to rotate freely.

The overall potential field considers all these forces in order to insure a feasible path. The total potential E_T is as follows:

$$E_T = E_O + E_{NC} + E_C \tag{5}$$

3.2 Gradient Descent Algorithms

We use two basic algorithms to follow the gradient descent direction. The basic method is the steepest descent method. This method works well when the robot is close to the obstacles. However, it is well known that its convergence rate can be very slow [1]. To accelerate convergence, we use the Gauss-Newton method as we approach a local minimum. The Gauss-Newton method tends to work well in descending along a shallow valley with steep sides. The general descent algorithm first follows the steepest descent (Sec. 3.2.1) direction for a few steps, after which it switches to Gauss-Newton (Sec. 3.2.2) to find a local minimum. To account for the possibility of a discontinuity in the derivative at the local minimum found by Gauss-Newton, some additional probing is carried out around that local minimum, as discussed in Sec. 3.2.3.

3.2.1 Steepest Descent Method

The gradient of the potential function at $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{m-1})^T$ is

$$\nabla E(\boldsymbol{\theta}) = \boldsymbol{\mathcal{J}}^T \mathbf{r} \tag{6}$$

where $\boldsymbol{\mathcal{J}} \in \mathbf{R}^{\text{BOUND}_{all} \times m}$ is the Jacobian matrix of $\mathbf{r}$, defined by

$$\boldsymbol{\mathcal{J}}_{i,j} = \frac{\partial r_i}{\partial \theta_j} \tag{7}$$

To obtain a descent direction, we simply follow the direction of the negative gradient (6). This yields the method of steepest descent.

The Jacobian matrix is composed of three parts corresponding to the three parts forming the potential function.

1. Jacobian due to E_O. The Jacobian matrix is computed directly from (2). Even though the $Q_{i,j}$ in Figure 3 change as the robot moves, they are fixed for simplicity of computation. Let us denote $P_{i,j} = (x_{i,j}, y_{i,j})$ and $Q_{i,j} = (p, q)$. Also, the k-th joint is at (x_k, y_k). The computation of $\boldsymbol{\mathcal{J}}$ due to obstacles is as follows:

$$\begin{aligned}
\boldsymbol{\mathcal{J}}_{i+jm,k} &= \frac{\partial r_{i+jm}}{\partial \theta_k} = \frac{\partial}{\partial \theta_k}\left(\frac{1}{d_{i,j}}\right) = \frac{\partial}{\partial \theta_k}\|P_{i,j} - Q_{i,j}\|_2^{-1} \\
&= \frac{\partial}{\partial \theta_k}\left[(x_{i,j}-p)^2 + (y_{i,j}-q)^2\right]^{-1/2} \\
&= -\frac{(x_{i,j}-p)\frac{\partial x_{i,j}}{\partial \theta_k} + (y_{i,j}-q)\frac{\partial y_{i,j}}{\partial \theta_k}}{\left(\sqrt{(x_{i,j}-p)^2 + (y_{i,j}-q)^2}\right)^3} \qquad (8)
\end{aligned}$$

where $0 \le i < \text{BOUND}_O$, and $0 \le j < m$. The derivative $\frac{\partial x_{i,j}}{\partial \theta_k}$ is calculated as follows:

$$\begin{aligned}
\frac{\partial x_{i,j}}{\partial \theta_k} &= \frac{\partial}{\partial \theta_k}\left[x_0 + \sum_{u=0}^{i} l_u cos(\sum_{v=0}^{u}\theta_v)\right] \\
&= -\sum_{u=k}^{i} l_u sin(\sum_{v=0}^{u}\theta_v) \\
&= (y_k - y_{i,j}) \qquad (9)
\end{aligned}$$

Similarly,

$$\frac{\partial y_{i,j}}{\partial \theta_k} = x_{i,j} - x_k \qquad (10)$$

Therefore,

$$\boldsymbol{\mathcal{J}}_{l,k} = \frac{(x_{i,j}-p)(y_{i,j}-y_k) - (y_{i,j}-q)(x_{i,j}-x_k)}{\left(\sqrt{(x_{i,j}-p)^2 + (y_{i,j}-q)^2}\right)^3} \qquad (11)$$

where $l = i + jm$, $0 \le i < m$, and $0 \le j < n$.

Computation of each term in (11) is not too difficult and is needed for collision checking anyway. However, the size of the Jacobian matrix can be quite large. Therefore, this method may be slow if there are too many obstacles in the W-space. In that case, the potential function calculation might be simplified, especially for obstacles far from individual links. By segmenting the entire C-space into regions, it would be a straighforward process to limit the calculation of the potential function to those obstacles which are close to the relevant links, by examining only those regions holding each link

or adjacent to each link. The potential from farther obstacles can be approximated by grouping them together, or can be disregarded altogether. For simplicity, we do not consider this here.

2. **Jacobian due to E_{NC}.** The Jacobian matrix for non consecutive robot links is computed similarly:

$$\mathcal{J}_{\text{BOUND}_O+l,k} = \frac{(s-p)(t-y_k)-(t-q)(s-x_k)}{\left(\sqrt{(s-p)^2+(t-q)^2}\right)^3} \tag{12}$$

where (s,t) and (p,q) are closest points between i-th link and j-th link respectively, $l=(i-1)(i-2)/2+j$, $2 \le i < m$, and $0 \le j < i-1$.

3. **Jacobian due to E_C.** By taking the derivative of corresponding components of r, we obtain:

$$\mathcal{J}_{\text{BOUND}_{NC}+l,k} = \begin{cases} 0 & \text{if } i \neq k \\ \frac{-1}{(\pi-\theta_i)^2} & \text{if } (\pi-\theta_i) < 0) \\ \frac{1}{(\pi-\theta_i)^2} & \text{if } (\pi-\theta_i) > 0) \end{cases} \tag{13}$$

where $l=i-1$, and $1 \le i < m$.

3.2.2 Gauss-Newton Method

We use the Gauss-Newton method [1] to speed up convergence. The Gauss-Newton method is based on a sequence of linear approximations of $\mathbf{r}(\boldsymbol{\theta})$. If $\boldsymbol{\theta}_k$ denotes the current approximation, then a correction $\mathbf{p}_k$ is computed as a solution to the linear least squares problem

$$\min_p \|\mathbf{r}(\boldsymbol{\theta}_k) + \mathcal{J}(\boldsymbol{\theta}_k)\mathbf{p}\|_2, \qquad \mathbf{p} \subseteq \mathbf{R}^m, \tag{14}$$

and the new approximation is $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \mathbf{p}_k$, where α_k is a step length to be determined. This linear least squares problem is solved using the QR decomposition of $\mathcal{J}(\boldsymbol{\theta}_k)$ [1]. One of the important properties of the Gauss-Newton direction is that if $\boldsymbol{\theta}_k$ is not a critical point, then $\mathbf{p}_k$ is a descent direction [1].

Because we want to find a path to the local minimum as well as the local minimum itself, α_k must be determined in such a way so as to limit the maximum step movement of the robot. Starting with $\alpha_k \triangleq 1/\|\mathbf{p}\|$, the stepsize α_k is halved repeatly until the new potential

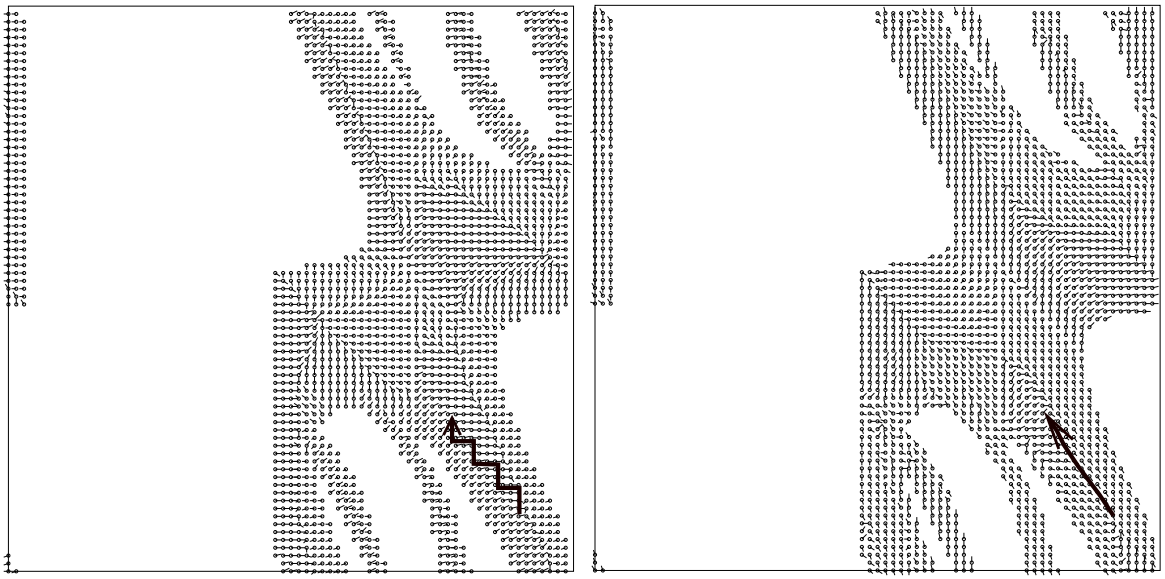


Figure 4: Behavior of gradient descent methods on the example of Figure 1. The left side indicates the steepest descent direction at each point, and the right side shows the Gauss-Newton directions.

$E(\boldsymbol{\theta}_{\mathbf{k+1}})$ is less than the old potential $E(\boldsymbol{\theta}_{\mathbf{k}})$. If the new potential is already less than the old potential at the initial stepsize, then the stepsize α_k is doubled until a predefined limit is reached or the new potential becomes larger than the old potential.

Figure 4 illustrates the behavior of the two gradient descent methods for the C-space of Fig. 1. The left side of Figure 4 indicates the steepest descent directions, which are almost perpendicular to the potential walls from the obstacles. In the narrow passage in the C-space, however, steepest descent method is not very efficient because it tends to oscillate. The right side of Figure 4 shows the descent path for the Gauss-Newton method, without the oscillation.

3.2.3 Discontinuity

Normally, a combination of steepest descent and Gauss-Newton is very effective in finding local minima. Steepest descent is very robust, especially in the steep potential walls near obstacles, while Gauss-Newton is very effective once one has moved away from the immediate neighborhood of an obstacle. However, once a possible local minimum is reached, one must

carry out some additional probing in order to account for the possibility of a discontinuity in the derivatives. Such a discontinuity may arise from the way the potential function is constructed. In our case, the points $P_{i,j}, Q_{i,j}$ (the closest points of approach between a link and an obstacle) may jump as the configuration changes. For example, the points of closest approach between the obstacle and the uppermost link shown in Fig. 3 will jump if the link rotates a little to the right.

The probing technique we use is based on moving a small step in some probing direction and then descending along the gradient to a local minimum. In order to explore new positions, the new gradient descent process is restricted to a hyperplane normal to the original probing direction in a manner described in the next section. A thorough exploration of the neighborhood of the original local minimum is accomplished by probing in this manner in all the coordinate directions. If any probing direction yields a point with a lower potential, this point replaces the original local minimum.

4 Building the Graph by Connecting Local Minima

Our goal is to build a roadmap consisting of all the found local minima together with the computed paths connecting them. The previous section discussed the gradient descent methods used to find a "first" local minimum. To connect different local minima, as well as to discover new nearby local minima, we use an up-hill ascent process in a given exploration direction to climb away from a local minimum and find a path to another local minimum. Our exploration process continues finding new local minima, connecting them to the previous local minima to form a partial path in the overall graph, in such a way as to avoid doubling back to a local minimum already visited. This algorithm has the property of passing through saddle points in the potential function between local minima, which limits the height of the potential hill traversed by a given path.

The roadmap is built by using the above process to create many partial paths. These paths are then connected to form the overall graph. In many cases, the paths are already connected through the random search process, but often a Phase II is needed to focus the search process in order to complete the connections. which are then connected to form the

overall graph.

The heart of the exploration process is the ***UpStream*** algorithm, described in Sec. 4.1. Sections 4.2 and 4.3 discuss the strategy to connect partial paths and the parallelization of the algorithm. A property of passing through saddle points is proved in the Appendix.

4.1 Moving Up Hill

The task here is to find a path from one local minimum to another. This path will form part of a partial path, which will eventually be connected to other partial paths to form one overall connected graph encompassing all the local minima.

The basic up-hill search strategy is given by procedure ***UpStream*** in Fig. 5. Starting at a local minimum, we select a random coordinate direction to move along. We take a small step in this chosen direction, and then descend "laterally" with a gradient descent search restricted to the hyperplane normal to the chosen probing direction, as illustrated in Fig. 6. The procedure is described more formally in Fig. 5. The lateral descent step allows us to follow the "valley floors," at least for some small distances, making it easier to find narrow passageways among the obstacles in C-space. We found that this process is very effective at reaching otherwise difficult to find local minima such as those with relatively small "descent basins." This procedure has already been mentioned as a way to probe in the neighborhood of a prospective local minimum to verify that it is indeed a local minimum (Sec. 3.2.3).

We remark in step 3 of ***UpStream*** that if $E(\boldsymbol{\theta}^{(k+1)}) < E(\boldsymbol{\theta}^{(k)})$ and $E(\boldsymbol{\theta}^{(k-1)}) < E(\boldsymbol{\theta}^{(k)})$ then we are at a top of a hill or "mountain pass." These points are often saddle points in the potential function. A precise statement and proof of this property is in the Appendix.

The main use of the ***UpStream*** procedure is to search for neighboring local minima, connecting them together as outlined in Fig. 2. This procedure is used repeatedly throughout the entire exploration process, but the main use is in Phase I of ***GraphBuild*** to explore paths from found local minima leading to new nearby local minima, and in Phase II to explore paths leading from isolated local minima leading to local minima within the largest connected component of the roadmap. Each exploration process consists of repeated passes through the ***UpStream*** procedure, until an obstacle is encountered or a known local minimum is

reached.

4.2 Connecting Partial Paths

We found that simply searching for local minima from randomly chosen points is not very efficient because the method tends to repeatedly land on certain more easily found local minima while missing many other local minima. The result is that many local minima are connected together in one large connected component, while other local minima are isolated by themselves or in very small connected components. If such isolated local minima are recognized, it is much more efficient to concentrate on such local minima. Our experiments have shown that after many partial paths are found and attached to the graph, a connected subgraph emerges containing the majority of the local minima, while the other local minima are spread among several smaller connected components. Therefore we keep track of the size of the largest graph. When the largest graph contains more than a given percentage of all the local minima found, then the algorithm stops generating random partial paths and switches over to a "Phase II" in which it selects only the local minima in the smaller graphs for more expansion (Fig. 2). The percentage at which the algorithm switches strategy can be varied by the user. The biggest effect of varying the percentage is on the overall cost of the method, as will be seen in Section 6.

The resulting graph consists of vertices representing the local minima in the potential function. The edges in the graph represent the paths connecting those points (Fig. 6). To save space within the process, we limit the information stored for each path to just those items needed to identify and recreate the path when needed (see Sec. 5.3).

4.3 Parallel Implementation of Graph Generation

In order to construct the roadmap, we need to connect the generated partial paths together. Each partial path can be generated independently and in parallel. The only communication required occurs when a partial path is complete and must be connected to the existing graph. Hence the overall process is quite suitable for parallel processing. We use a master-slave scheme for the parallel implementation. The master program is responsible for accepting

1. Follow the selected direction $\mathbf{u}_0$ for a small distance α.

2. Find the value $\boldsymbol{\phi}^{(k+1)}$ in the hyperplane yielding the local minimum in

$$\min_{\boldsymbol{\phi}} E(\boldsymbol{\theta}^{(k)} + \mathbf{u}_0\alpha + \boldsymbol{\mathcal{U}}\boldsymbol{\phi})$$

(the "lateral" movement), and set the new position,

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \mathbf{u}_0\alpha + \boldsymbol{\mathcal{U}}\boldsymbol{\phi}^{(k+1)},$$

where $\boldsymbol{\mathcal{U}}$ is a matrix whose columns form an orthonormal basis of the hyperplane normal to $\mathbf{u}_0$.

3. If $E(\boldsymbol{\theta}^{(k+1)}) > E(\boldsymbol{\theta}^{(k)})$ and $E(\boldsymbol{\theta}^{(k-1)}) > E(\boldsymbol{\theta}^{(k)})$, record a new 'valley" and construct a "side path" using a standard gradient descent method to find the local minimum in the new "valley." (Duplicate local minima found this way are discarded.)

4. Repeat this process setting $k := k + 1$ until an obstacle is encountered.

Figure 5: **UpStream** – Probe in a given selected direction $\mathbf{u}_0$.

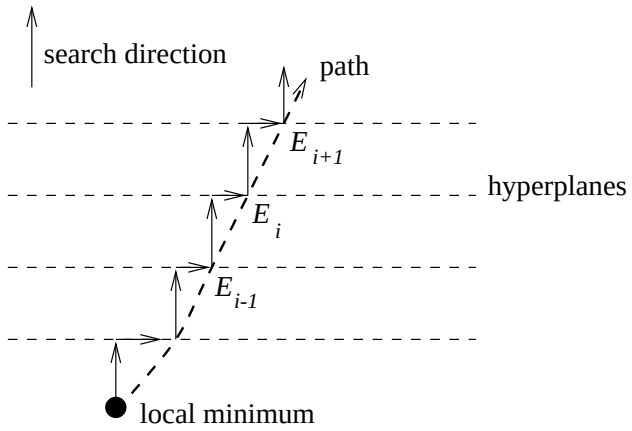


Figure 6: Illustration of procedure **UpStream** of Fig. 5, repeated several times. Each iteration consists of two parts: 1. a fixed movement along the chosen search direction, and 2. a descent to a local minimum within the hyperplane normal to the chosen direction. The iteration is repeated from that intermediate local minimum. Each E_i represents the potential at each intermediate local minimum.

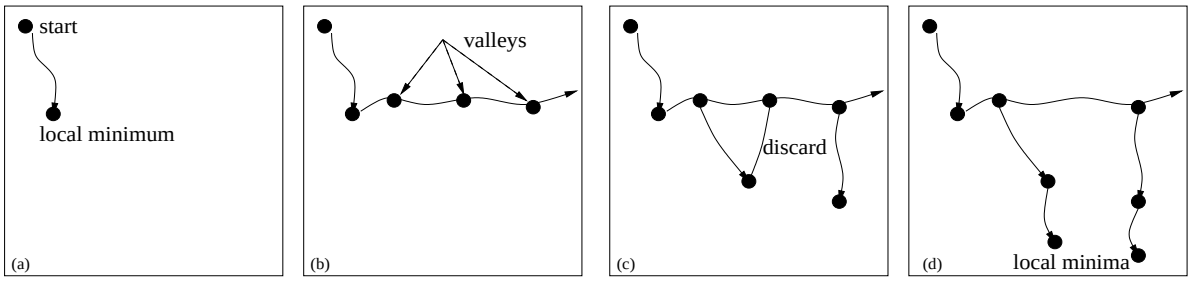


Figure 7: Connecting local minima using repeated passed through procedure **UpStream**. The planner (a) starts from a random configuration to find the local minimum; (b) looks for "valleys" as the starting points for seeking new local minima; (c) finds the local minima by gradient descent, merging those found; (d) probes to find exact local minima.

partial paths and connecting the nodes and edges to the graph, and the slave programs keep generating and sending partial paths to the master program. Generating each partial path generally takes 2 orders of magnitude more computation than gluing it to the graph. The messages in both directions are very short. Therefore we expect linear speedup for up to at least a few hundred processors.

5 Finding a Path from Start to Goal Position

A path connecting two given configurations is found by connecting those configurations to the roadmap and seeking a path connecting those configurations within the roadmap. In this section we discuss the relatively simple process of using the roadmap to connect two given configurations. In the following, Section 5.1 deals with connecting the initial and goal configurations to the graph, Section 5.2 describes the graph search method that was used, and Section 5.3 discusses how to regenerate the detailed path from the path found by the graph search.

5.1 Connecting Input Configurations to Graph

In order to find a path from the initial to the goal configurations, the input configurations must be connected to the graph first. This is done by a straightforward application of

the gradient descent methods of Sec. 3, starting from the initial and goal configurations and descending to their respective local minima. These local minima are often already on the graph, otherwise Phase II of the graph building algorithm (Fig. 2) is used to compute a connection. To be more precise, Phase II of the graph construction algorithm tries to connect small graphs to the largest graph. Therefore, if a local minimum is not found in the graph, it is viewed as a non-connected small subgraph, and a Phase II search is used to connect it to the large graph.

In our implementation, only the local minima and the exploration directions used to climb out of the local minima (the "$\mathbf{u}_0$" in Fig. 5) were saved; the rest of the partial path information was thrown away. Therefore, after the topological path is found in the graph, the planner regenerates the actual path. Of course, one could preserve the partial paths as they are generated during the graph construction phase, but the process of regenerating the detailed partial paths is relatively inexpensive.

One implementation decision is the condition under which the graph building process should switch over from Phase I to Phase II (Fig. 2). Initially we thought that it pays to switch over once we thought that we had found a large fraction of the local minima and had connected most of the found local minima into a large graph. This would make it much more likely that any given new configuration could be easily connected to the roadmap by a simple gradient descent process leading to a local minimum already on the roadmap. However, we discovered experimentally that finding a large fraction of the local minima can be too expensive. In addition, even if the gradient descent process from a given configuration does not reach a local minimum on the roadmap, the Phase II process is very effective in connecting it up. From our experiments, a typical graph contains about half of all the possible local minima, so the probability of finding the local minima in the graph at the first attempt is 50%. However, the graph having half of all the possible local minima can be considered to be very dense in the sense that the planner can find the connections from the other local minima to the graph very easily. Typically, it takes just one or two expansions to reach the graph, in our scenarios.

5.2 Graph Search

Many methods for finding a path in a state-space graph or tree have been proposed in [11], [16], [15]. In our case, the graph search is a lot simpler because: (a) all the nodes and edges are known and fixed, (b) typically, the number of nodes is modest, on the order of thousands. The physical length of the path is not considered in our current implementation. In future work, we will use more advanced methods which take the actual path length into consideration and which are more suitable when the number of nodes becomes very large.

Since we do not consider the physical length of the path, breadth-first search (or BFS [11], [16], [15]) is used, which is simple to implement and reasonably fast. In the worst case, BFS will explore every node in the graph. Thus, this method is complete in the sense that the method will find a path if there is one in the graph. Usually, breadth-first-search can suffer from a large space requirement that is exponential in the depth of the solution. In our case, however, we start with a graph in which all the nodes and edges are known and fixed. Furthermore, the number of nodes rarely exceeds one thousand. Another reason BFS is successful is that even though the branching factor is high, the majority of the newly expanded nodes are already in the BFS queue. In any case, the total number of nodes cannot exceed the total number of local minima. Graphs of this size can be easily handled by modern computers, even by PC-level computers. The experimental results show that BFS can find the solutions in a fraction of a second on a single processor Pentium Pro 200MHz with 64MB of main memory. Of course, in more complex domains, the graph search problem could become a more significant issue.

5.3 Regenerating the Actual Paths

To save space stored within the roadmap and to reduce the volume of information transfered between parallel processors, we discarded the detailed partial path information generated during the graph construction algorithm, leaving only the information necessary to regenerate the paths. The edges in the graph have the following information: (a) the location of the first local minimum, (b) the location of the second local minimum connected thereto, (c) the probing direction in which the planner explored to connect the first local minimum to the

next. Given just this limited information, it is straightforward to regenerate the path with all its intermediate points. Regenerating each partial path can be done in parallel. The path regeneration algorithm takes advantage of this and the regeneration of each component of all the partial paths are done in parallel. Usually, the number of tasks (each corresponding to each component to be generated) is much larger than the number of processors available. The cost of each task can vary a lot depending on the length of each component and the number of valley points traversed. Therefore, a dynamic approach for load balancing is used. The outline of the algorithm is presented below:

1. The master program decomposes all the partial paths and stores them in a list. Each element in the list is the "command" and the information needed to regenerate a component.

2. The master program sends to each processor a "command" from the list.

3. As a slave processor sends the partial path, the master program collects the path and allocates a new "command" (if any) to the slave processor.

4. Once the master program has received all the components, the whole path is returned and the program is finished.

In our experiments we found that the parallel performance of the path regeneration process suffered because of the variability in the lengths of the individual partial paths. Nevertheless, the entire path regeneration process never took longer than 21 seconds with 16 processors in the most complex cases, having about 16 local minima in a single path.

6 Experimental Results

We tested our method on many examples, including a 7 *dof* and an 8 *dof* manipulator illustrated in figures 8 and 9. We used an SGI Challenge Cluster consisting of 3 Challenge L machines and 1 Challenge XL machine. Each Challenge L machine has 4 R10000 processors and Challenge XL has 8 R10000 processors. We used PVM3 (Parallel Virtual Machine) for

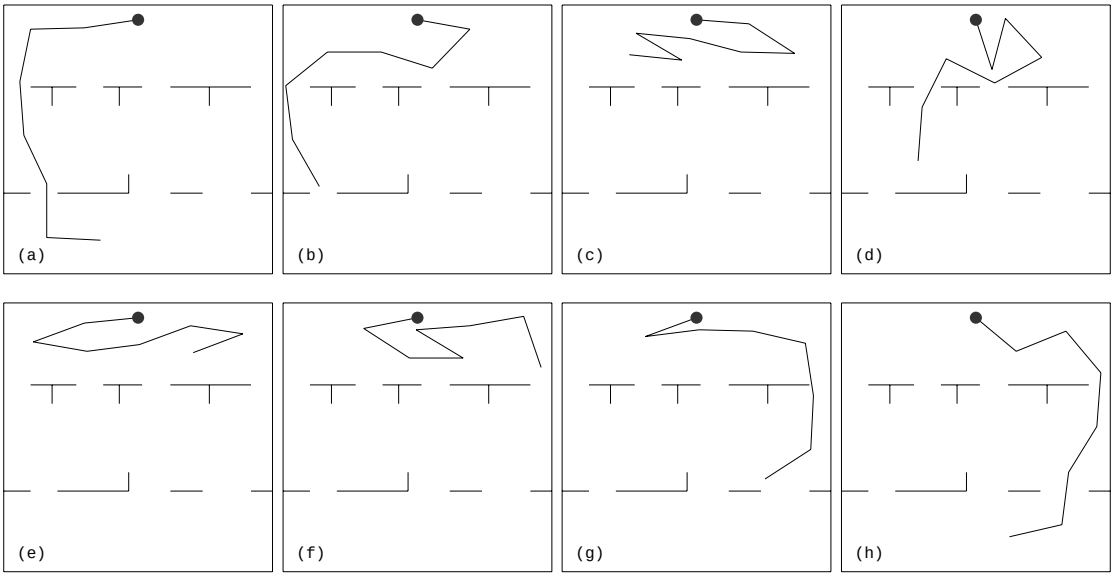


Figure 8: A 7 *dof* robot example from start(a) to goal(h) configuration.

ratio	*node count*		*graph build time*			*graph search time*			*path regen. time*		
P	count	dev	aver	dev	worst	aver	dev	worst	aver	dev	worst
30%	698	105	237	85	350	3	2	6	10	10	21
60%	954	162	463	144	634	2	2	5	8	6	15
80%	993	83	482	97	620	2	1	3	8	2	11

Table I: Performance for Figure 8 with respect to Phase II switch-over ratio P with 16 processors. "aver", " dev", "worst" stand for average, standard deviation, and worst case, respectively. The node count is the number of local minima found. Times are in seconds.

the parallel processing. Figures 8 and 9 show the actual paths connecting the start and goal configurations found by our method.

In Table I, each experiment was run 10 times. Along with the average values, the standard deviations and worst cases are shown. The symbol P represents the ratio of the largest connected component size vs the total number of nodes at which we switch over from Phase I (random generation) to Phase II (focused search). It took about 7 minutes to compute the graph for Figure 8 with the Phase II switch-over percentage of 60%. The time for connecting the Figure 8(a) and Figure 8(h) to the graph after pre-processing was 2 seconds. Regenerating

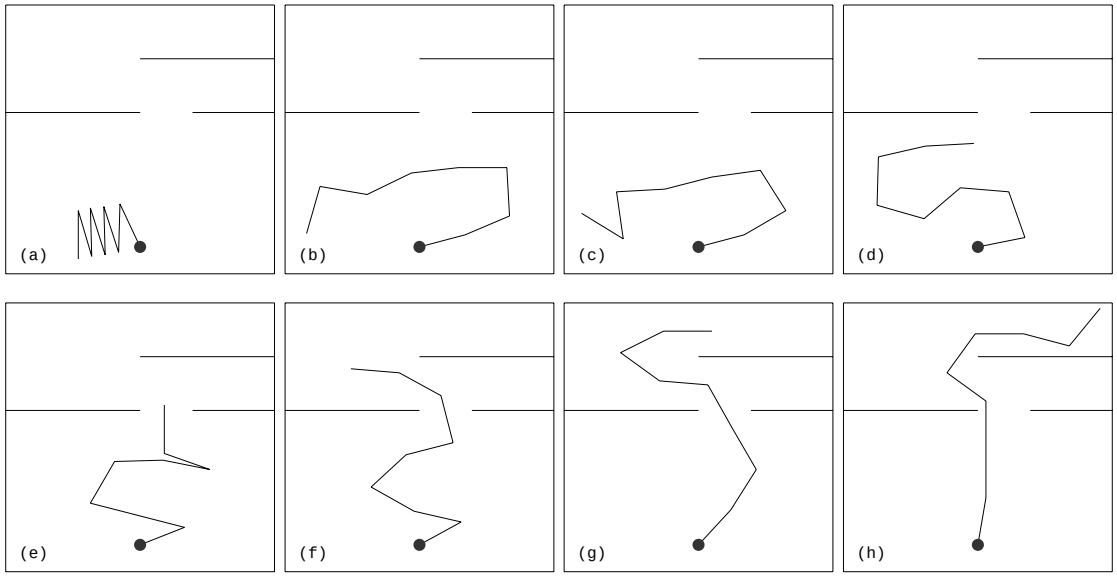


Figure 9: An 8 *dof* robot example from start(a) to goal(h) configuration.

ratio	*node count*		*graph build time*			*graph search time*			*path regen. time*		
P	count	dev	aver	dev	worst	aver	dev	worst	aver	dev	worst
60%	182	25	87	11	96	12	10	23	6	7	16
70%	193	26	93	14	105	11	8	22	5	3	10
80%	257	171	209	155	442	10	3	14	5	1	7

Table II: Performance for Figure 9 with respect to Phase II switch-over ratio with 16 processors, using same notation as in Table I. Times are in seconds.

the path in W-space took about 8 seconds. For this example, the performance of actually finding the path is quite consistent because obstacles are rather evenly distributed in the W-space. Changing the Phase II switch-over ratio directly affects the number of nodes generated and it also affects the path regeneration time. If the ratio is set too low, the resulting graph has fewer nodes and the path generated may not be as good. However, the fastest total time to construct a roadmap and compute a path was obtained by using a relatively low switch-over ratio of 30%.

For the second example (Figure 9), the fastest pre-processing time for Figure 9 was 47 seconds when the Phase II switch-over ratio was set to 60%. However, the performance of

no. procs	*graph build time*			*speedup*	*efficiency*
NP	aver	dev	worst	SU	SC
1	6684	913	7692	1.00	1.00
4	1750	280	2120	3.82	0.95
8	921	153	1100	7.26	0.91
16	482	97	620	13.87	0.87

Table III: Graph build time for Figure 8 with respect to number of processors.

finding the actual path depended more on the location of the input configuration, because if it was in a difficult to find location, extra applications of Phase II were often needed to connect it to the roadmap. Nevertheless, the time to connect a given configuration to the graph never exceeded 10 seconds.

Table III shows the time to build the graph for Figure 8 with a varying number of processors. The Phase II switch-over ratio was set to 80%. NP is the number of processors, SU represents the speed-up, and $SC = SU/NP$. The scalability is quite good as expected up to 16 processors. Future work will include testing scalability on more massively parallel testbeds.

7 Conclusion and Future Work

In this paper, we presented a novel method to combine randomized search and potential fields to solve motion planning problems in 2 dimensional W-space and high dimensional C-space. The experimental results show that this method is very efficient even for very difficult problems with a short period of preprocessing. The results show that even an incomplete roadmap can be a very effective tool for exploring a workspace, and attempting a-priori to find all the local minima may not be the most efficient approach. This is true even when there are hard-to-find local minima such as those in narrow passageways or with small "descent basins." Further study on the effect of varying the switch-over ratio from the general Phase I search to the focused Phase II search is needed.

The differentiable potential function and the up-hill ascent strategies used in this paper

are easily adaptable to 3D W-spaces. From our experiments, the complexity of the problem in terms of local minima has more impact on the performance than the degrees of freedom. In previous work ([4], [5]), the C-space resulting from 3D W-space tends to be simpler than 2D cases. We expect the 3D implementation of our method will have a performance similar to 2D cases, since the complexity of the mathematical computations should not increase by much. If there are many obstacles, it may become necessary to approximate the potential function by speeding up the calculation of the contributions from obstacles that are far from the links. The number of local minima, and hence the size of our graphs, may increase, leading to the need to more specialized graph search algorithms. These aspects, as well as a study of the saddle points in the potential function, will be a focus of our future research.

References

[1] Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, 1989.

[2] P. Adolphs and H. Tolle. Collision-free real-time path-planning in time varying environment. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 445–452, 1992.

[3] J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Trans. Systems, Man, and Cybernetics*, SMC-22(2):224–241, March/April 1992.

[4] D. Challou. Parallel search algorithms for robot motion planning. Ph.D. dissertation, The University of Minnesota, 1995.

[5] D. Challou, M. Gini, and V. Kumar. Toward real-time motion planning. In H. Kitano, V. Kumar, and C. B. Suttner, editors, *Parallel Processing for Artificial Intelligence, 2.* Elsevier, 1994.

[6] D. Henrich. A review of parallel processing approaches to motion planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 1996.

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

[8] L. Kavraki. Randomized preprocessing of C-space for fast path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2138–2145, 1994.

[9] Lydia Kavraki, J.C. Latombe, Petr Svestka, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.

[10] Thierry Laliberte and Clement Gosselin. Efficient algorithms for the trajectory planning of redundant manipulators with obstacle avoidance. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2044–2049, 1994.

[11] Pat Langley. Systematic and nonsystematic search strategies. In *Proc. Int'l Conf. on AI Planning Systems*, pages 145–152, College Park, Md, 1992.

[12] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publ., Norwell, MA, 1991.

[13] V. Lumelsky and Edward Cheng. Real-time collision avoidance in teleoperated whole sensitive robot arm manipulators. *IEEE Trans. Systems, Man, and Cybernetics*, SMC-23(1):194–203, Jan/Feb 1993.

[14] E. Mazer, J.M. Ahuactzin, and P. Bessiere. The ariadne's clew algorithm. *J. Artif. Intel. Res.*, 9:295–316, 1998.

[15] E. Rich. *Artificial Intelligence*. McGraw-Hill, 1991.

[16] P. Winston. *Artificial Intelligence*. Addison-Wesley, June 1992.

Appendix: Saddle Points

In this Appendix, we prove that under certain conditions, the partial paths generated by repeated passes through the ***UpStream*** process passes through saddle points in the potential function. Let $\mathbf{u}_0$ denote the given search direction. We assume that we have a path

$\boldsymbol{\theta}^{(0)}, \boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$ in C-space, where each $\boldsymbol{\theta}^{(k)}$ is the local minimum found within the hyperplane normal to $\mathbf{u}_0$ in step 2 of Fig. 5. We assume that the stepsize α is small enough that the points $\boldsymbol{\theta}^{(0)}, \boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$ lie on a continuous path of configurations where each point on the path is the local minimum with the hyperplane passing through that point, normal to $\mathbf{u}_0$. On this continuous path, we denote by $\boldsymbol{\theta}_0$ a point where the potential function reaches a local maximum along the path, and $\mathbf{x} \triangleq \boldsymbol{\theta} - \boldsymbol{\theta}_0$ the deviation from that point of local maximum. We let $\boldsymbol{\mathcal{U}}$ be constructed such that $[\mathbf{u}_0, \boldsymbol{\mathcal{U}}]$ is an orthonormal matrix.

In a neighborhood of the local maximum $\mathbf{x} = 0$, the potential function is locally a quadratic F of the form:

$$E(\mathbf{x}) \approx F(\mathbf{x}) \triangleq \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c \tag{15}$$

where $\mathbf{A} \triangleq \nabla^2_{\mathbf{x}} E(\mathbf{x})|_{\mathbf{x}=\mathbf{0}}, \mathbf{b} \triangleq -\nabla_{\mathbf{x}} E(\mathbf{x})|_{\mathbf{x}=\mathbf{0}}$,c$\triangleq$ a constant.

For each step α, we denote by $\mathbf{w}_\alpha$ the local minimum within the hyperplane in the basis parametrizing the hyperplane. Specifically, $\mathbf{w}_\alpha \triangleq$ the solution of $\mathbf{w}$ that minimizes $F(\alpha\mathbf{u}_0 + \boldsymbol{\mathcal{U}}\mathbf{w})$ over all $\mathbf{w} \in \mathbf{R}^{m-1}$ in the neighborhood of $\mathbf{0}$, $\forall\alpha$ in the neighborhood of 0, where m is the dimension of the search space:

$$\mathbf{w}_\alpha \triangleq \operatorname*{argmin}_{\mathbf{w}} F(\alpha\mathbf{u}_0 + \boldsymbol{\mathcal{U}}\mathbf{w}). \tag{16}$$

The points $\mathbf{x}_\alpha \triangleq \alpha\mathbf{u}_0 + \boldsymbol{\mathcal{U}}\mathbf{w}_\alpha$ define the path parameterized by α in a neighborhood of $\alpha = 0$ with potential function $G(\alpha)$.

$$G(\alpha) = F(\alpha\mathbf{u}_0 + \boldsymbol{\mathcal{U}}\mathbf{w}_\alpha). \tag{17}$$

In particular, $\mathbf{x}_0 = \mathbf{x}_\alpha|_{\alpha=0} = 0$.

We can now state the theorem that under certain conditions, the point $\boldsymbol{\theta}_0$ which is a local maximum on the path is a saddle point.

Theorem Assume (A1) the Hessian $\mathbf{A}$ and the Hessian restricted to the hyperplane $\boldsymbol{\mathcal{U}}^T\mathbf{A}\boldsymbol{\mathcal{U}}$ are non-singular, and (A2) $\mathbf{w}_\alpha$ exists and is unique $\forall\alpha$ in a neighborhood of $\alpha = 0$. If $G(\alpha)$ achieves a local maximum at $\alpha = 0$, then $\mathbf{x}_\alpha$ is a saddle point of the potential F.

Proof By substituting $\alpha\mathbf{u}_0 + \boldsymbol{\mathcal{U}}\mathbf{w}_\alpha$ for $\mathbf{x}$ in (15), we get

$$\begin{aligned} &F(\alpha\mathbf{u}_0 + \boldsymbol{\mathcal{U}}\mathbf{w}) \\ &\quad = \tfrac{1}{2}\alpha^2\mathbf{u}_0^T\mathbf{A}\mathbf{u}_0 - \alpha\mathbf{b}^T\mathbf{u}_0 + \alpha\mathbf{u}_0^T\mathbf{A}\boldsymbol{\mathcal{U}}\mathbf{w} + \tfrac{1}{2}\mathbf{w}^T\boldsymbol{\mathcal{U}}^T\mathbf{A}\boldsymbol{\mathcal{U}}\mathbf{w} - \mathbf{b}^T\boldsymbol{\mathcal{U}}\mathbf{w} + c \end{aligned} \tag{18}$$

1. First, we prove that the Hessian matrix $\mathbf{A}$ is indefinite.

   (a) If we take the first derivative of (18) with respect to $\mathbf{w}$,

   $$\nabla_{\mathbf{W}} F(\alpha \mathbf{u}_0 + \boldsymbol{\mathcal{U}}\mathbf{w}) = \mathbf{w}^T \boldsymbol{\mathcal{U}}^T \mathbf{A} \boldsymbol{\mathcal{U}} - [\mathbf{b}^T \boldsymbol{\mathcal{U}} - \alpha \mathbf{u}_0^T \mathbf{A} \boldsymbol{\mathcal{U}}] \tag{19}$$

   The Hessian matrix of (18) is

   $$\nabla^2_{\mathbf{W}} F(\alpha \mathbf{u}_0 + \boldsymbol{\mathcal{U}}\mathbf{w}) = \boldsymbol{\mathcal{U}}^T \mathbf{A} \boldsymbol{\mathcal{U}} \tag{20}$$

   Since $\mathbf{w}_0$ is a local minimum in the hyperplane, the Hessian $\boldsymbol{\mathcal{U}}^T \mathbf{A} \boldsymbol{\mathcal{U}}$ is positive definite.

   (b) By setting equation (19) equal to $\mathbf{0}$, we have

   $$\mathbf{w}_\alpha = [\boldsymbol{\mathcal{U}}^T \mathbf{A} \boldsymbol{\mathcal{U}}]^{-1} \boldsymbol{\mathcal{U}}^T [\mathbf{b} - \alpha \mathbf{A} \mathbf{u}_0] \tag{21}$$

   At $\alpha = 0$ we get $\mathbf{w}_0 = [\boldsymbol{\mathcal{U}}^T \mathbf{A} \boldsymbol{\mathcal{U}}]^{-1} \boldsymbol{\mathcal{U}}^T \mathbf{b} = \mathbf{0}$, which by assumption (A1) implies

   $$\boldsymbol{\mathcal{U}}^T \mathbf{b} = \mathbf{0} \tag{22}$$

   and therefore,

   $$\mathbf{b} = \lambda \mathbf{u}_0 \tag{23}$$

   for some λ. (21) reduces to $\mathbf{w}_\alpha = -\alpha [\boldsymbol{\mathcal{U}}^T \mathbf{A} \boldsymbol{\mathcal{U}}]^{-1} \boldsymbol{\mathcal{U}}^T \mathbf{A} \mathbf{u}_0$.

   Define $\boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{U}} [\boldsymbol{\mathcal{U}}^T \mathbf{A} \boldsymbol{\mathcal{U}}]^{-1} \boldsymbol{\mathcal{U}}^T \mathbf{A}$. Then (17) can be written

   $$\begin{aligned} G(\alpha) &= F(\alpha(\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}})\mathbf{u}_0) \\ &= \frac{\alpha^2}{2} \mathbf{u}_0^T (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}})^T \mathbf{A} (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}}) \mathbf{u}_0 - \alpha \mathbf{b}^T (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}}) \mathbf{u}_0 + c \end{aligned} \tag{24}$$

   The second derivative of the function G with respect to α must be negative because it is a local maximum along the curve:

   $$\left. \frac{d^2 G}{d\alpha^2} \right|_{\alpha = 0} = \mathbf{u}_0^T (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}})^T \mathbf{A} (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}}) \mathbf{u}_0 \; < \; 0. \tag{25}$$

From (a) and (b), it follows that the Hessian $\mathbf{A}$ is indefinite.

2. We still need to show that the gradient $-\mathbf{b} = \mathbf{0}$.

$$\begin{aligned}
0 &= \tfrac{dG}{d\alpha}|_{\alpha=0} \\
&= \alpha \mathbf{u}_0^T (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}})^T \mathbf{A} (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}}) \mathbf{u}_0 - \mathbf{b}^T (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}}) \mathbf{u}_0 \\
&= \mathbf{b}^T (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{B}}) \mathbf{u}_0 \\
&= \mathbf{b}^T \mathbf{u}_0 - \mathbf{b}^T \boldsymbol{\mathcal{B}} \mathbf{u}_0 \\
&= \mathbf{b}^T \mathbf{u}_0 - \mathbf{b}^T \boldsymbol{\mathcal{U}} [\boldsymbol{\mathcal{U}}^T \mathbf{A} \boldsymbol{\mathcal{U}}]^{-1} \boldsymbol{\mathcal{U}}^T \mathbf{u}_0 \\
&= \mathbf{b}^T \mathbf{u}_0 \qquad \text{(from (22))} \\
&= \lambda \mathbf{u}_0^T \mathbf{u}_0 \qquad \text{(from (23))}
\end{aligned}$$

Therefore, $\lambda = 0$ and $\mathbf{b} = \mathbf{0}$.

From 1 and 2, it follows that $\mathbf{x}_0$ is a saddle point of the potential function. □