Steps Toward Derandomizing RRTs

Stephen R. Lindemann Steven M. LaValle

Dept. of Computer Science
University of Illinois
Urbana, IL 61801 USA
{slindema, lavalle}@uiuc.edu

Abstract

We present two motion planning algorithms, based on the Rapidly Exploring Random Tree (RRT) family of algorithms. These algorithms represent the first work in the direction of derandomizing RRTs; this is a very challenging problem due to the way randomization is used in RRTs. In RRTs, randomization is used to create Voronoi bias, which causes the search trees to rapidly explore the state space. Our algorithms take steps to increase the Voronoi bias and to retain this property without the use of randomization. Studying these and related algorithms will improve our understanding of how efficient exploration can be accomplished, and will hopefully lead to improved planners. We give experimental results that illustrate how the new algorithms explore the state space and how they compare with existing RRT algorithms.

1 Introduction

For over a decade, randomized algorithms such as the Randomized Potential Field Planner (RPP) [3], the Probabilistic Roadmap (PRM) family [1, 9, 21, 24, 25], Rapidly-Exploring Random Trees (RRTs) [10, 12, 13], and others [4, 8, 17], have dominated the field of motion planning. Recently, a great deal of attention has been given to comparing random versus deterministic sampling in the context of PRMs [6, 11]. A recent survey of this field, termed *sampling-based motion planning*, is given in [15]. In this paper, we discuss the role of randomization in RRTs, and introduce two new planners which move toward their derandomization.

Randomization is a common algorithmic technique, and it is of great value in many contexts. Sometimes, it is used to defeat an adversary who might gain an advantage from learning one's deterministic strategy (e.g., cryptographic or sorting algorithms). Randomization is also useful for approximation or in conjunction with amplification techniques (e.g., the randomized min cut algorithm). It also allows for probabilistic performance analysis, which can be very useful. For problems of numerical integration, randomization can sometimes defeat the "curse of dimensionality" [22].

In the context of the original PRM, the primary use of randomization is to uniformly sample the configuration space. Recently, the usefulness of randomization for this purpose has been challenged; proponents of deterministic sampling argue that there are deterministic sequences satisfying other uniformity measures (e.g., discrepancy and dispersion) which perform at least as well as random sampling. Furthermore, these methods give deterministic guarantees of convergence (such as resolution completeness). Currently, work is being done both to study the performance of various random and deterministic sampling sequences in the PRM [11], and to construct new deterministic uniform sequences with other properties that are useful for motion planning [14].

It cannot be denied that contemporary motion planning algorithms, many of which use randomization, are very efficient and able to solve many challenging problems. This might lead one to conclude that randomization is the key to their effectiveness; however, this is not necessarily the case. On the contrary, randomization can easily become a "black box" which obscures the reasons for an algorithm's success. Hence, attempts to derandomize popular motion planning algorithms do not reflect antipathy toward randomization, but rather the desire to understand the fundamental insights of these algorithms. After studying an algorithm in both its randomized and derandomized forms, it will be possible to intelligently decide which to use, or whether some mixture of the two is appropriate. This approach might even result in algorithms that combine deterministic and randomized strategies in a way that achieves the benefits of both. For example, [20] uses a combined sampling strategy for constructing a Visibility PRM; they recursively divide the space into quadrants (a deterministic strategy) and choose a random sample within each quadrant. In the area of low-discrepancy sampling, Wang and Hickernell have constructed and analyzed randomized Halton sequences [23]. Geraerts and Overmars have also investigated randomizing Halton points [6].

We believe that a great deal of work remains to investigate deterministic variants of contemporary motion planning algorithms. We have already mentioned efforts

to derandomize PRMs; namely, those which attempt to use deterministic uniform sampling methods. It is also interesting to consider derandomizing RRTs; this is a very challenging task, due to the way that randomization is used in RRTs.

2 Randomization in RRTs

In the case of RRTs, derandomization is more difficult than with PRMs. In the original PRM, the primary use of randomization is to produce a uniformly distributed sample sequence in order to cover the space; for RRTs, the use of randomization is more subtle. As opposed to the former case, simply replacing random samples with deterministic ones will not capture the essence of the exploration strategy of RRTs. In order to understand the best way to derandomize RRTs, we will outline the role of random sampling in the basic RRT algorithm.

The basic RRT algorithm operates very simply; the overall strategy is to incrementally grow a tree from the initial state to the goal state. The root of the tree is the initial state; at each iteration, a random sample is taken and its nearest neighbor in the tree computed. A new node is then created by growing the nearest neighbor toward the random sample. For an in-depth description and analysis of RRTs, see [13].

In [13], it is argued that RRTs explore rapidly because samples "pull" the search tree toward unexplored areas of the state space. This occurs because the probability that a vertex is selected for expansion is proportional to the area of its Voronoi region. Hence, a node at the frontier of the tree is likely to be chosen to grow into previously unexplored territory. This argument can be made because the samples are independent and taken from a uniform random distribution. RRTs are consequently able to explore in a Voronoi-biased manner, with very low cost (generating random samples is inexpensive). This appears to be the primary use of randomization in RRTs. In this light, it may not be beneficial to simply replace random samples with deterministic ones in RRT planners. In a uniform deterministic sequence, samples are not independent from each other; consequently, one cannot argue about the probability of a vertex being selected in the same way as one can when using random samples. It is also clear that in the context of RRTs, the concepts of *Voronoi bias* and *deterministic sampling* are not necessarily related. In the case above, one could use deterministic sampling and have little or no Voronoi bias; as we will see later, one may freely vary the amount of Voronoi bias using random sampling.

It should be noted that for RRTs, Voronoi bias does not play the same role as in some other recent motion planning algorithms [5, 19]. These algorithms compute the discretized GVD (generalized Voronoi diagram) of the environment and use this information to sample near the medial axis of free space (for another medial axis-based approach, see [24]). In this case, the Voronoi diagram is based on the environment. In RRTs, however, the Voronoi bias occurs with respect to the Voronoi diagram of the nodes in the search tree; that is, the nodes in the search tree with the largest Voronoi regions tend to be selected for exploration.

One may use the concept of Voronoi bias to construct a deterministic RRT. Simply construct the d-dimensional Voronoi diagram of the nodes in the tree and use this information (along with a decision rule) to incrementally grow the search tree. Two possible decision rules are: grow toward the centroid of the largest Voronoi region (this requires calculation of the volumes of the Voronoi regions); or, attempt to reduce the size of the largest empty ball (the center of the largest empty ball is a Voronoi vertex). Either of these approaches is theoretically feasible, since it is well-known how to construct Voronoi diagrams in arbitrary dimensions. Practically, however, it is no simple task to robustly compute Voronoi diagrams in d dimensions, and implementing algorithms that do so is quite difficult. In addition to this, most construction methods use an ℓ_2 metric in Euclidean space; this is more restrictive than is appropriate for general motion planning problems, which may have different metrics and whose topologies are are often more complicated. Finally, the cost of explicitly computing this information may be prohibitive from a practical point of view.

While these Voronoi bias-maximizing approaches are worth exploring, we do not seek to do so in this paper. Instead, we propose two algorithms which lie between the original RRT and the fully Voronoi-biased approaches. These algorithms are more Voronoi-biased than the original RRT, but not as much so as those based on explicitly-constructed Voronoi diagrams. In related work, we introduce another derandomized RRT variant, which is based on inremental dispersion reduction [16]; however, that approach will not be discussed in this paper.

3 A spectrum of RRT-like planners

We wish to construct planners that take the idea of Voronoi bias and emphasize it more strongly than in the original RRT algorithm. However, to avoid the difficulties with explicitly constructing Voronoi diagrams, we desire an approximate, sampling-based approach. Hence, our new algorithms will lie in the middle of a spectrum of RRT-like planners, with the original RRT on one side and a deterministic Voronoi diagram-based method on the other.

We have seen that the RRTs grow in a Voronoi-biased manner due to the way they process the random samples drawn from the configuration space. What would happen, then, if instead of taking a single sample, one took k samples? One can sort the nodes in the tree according to how many samples they were the nearest neighbor for, and grow from the nodes which collected the most samples. We call this algorithm the Multi-Sample RRT (MS-RRTa), and pseudo-code for the basic algorithm is given in Figure 1. Note that during a particular itera-

```
BUILD_MS-RRTA(x_init)
 1  G_sub.init(x_init);
 2  for x = 1 to X do
 3      for i = 1 to k do
 4          x_rand ← RANDOM_STATE();
 5          x_near ← NEAREST_NEIGHBOR(x_rand, G_sub);
 6          x_near.sampleCount += 1;
 7          x_near.sampleAverage += x_rand;
 8      x_best ← max(x.sampleCount, x ∈ G_sub);
 9      x_next ← x_best.sampleAverage/x_best.sampleCount;
10      u_best, x_new, success ← CONTROL(x_best, x_next, G_sub);
11      if success
12          G_sub.add_vertex(x_new);
13          G_sub.add_edge(x_near, x_new, u_best);
14      CLEAR_SAMPLE_INFO(G_sub);
15  Return G_sub;
```

Figure 1: The basic MS-RRTa construction algorithm.

```
BUILD_MS-RRTB(x_init)
 1  G_sub.init(x_init);
 2  for k = 1 to K do
 3      ADD_NEW_SAMPLE(sample, sampleList);
 4  for x = 1 to X do
 5      x_best = max(x.sampleCount, x ∈ G_sub);
 6      x_next ← x_best.sampleAverage/x_best.sampleCount;
 7      u_best, x_new, success ← CONTROL(x_best, x_next, G_sub);
 8      if success
 9          G_sub.add_vertex(x_new);
10          G_sub.add_edge(x_near, x_new, u_best);
11      REDISTRIBUTE_SAMPLES(sampleList, G_sub);
12  Return G_sub
```

Figure 2: The basic MS-RRTb construction algorithm.

tion, a node grows toward the average of the samples it collected; this may be viewed as an estimate of the centroid of that node's Voronoi region (clearly, it is guaranteed to be within the node's Voronoi region). Also, as k approaches infinity, we probabilistically obtain the exact Voronoi volumes and Voronoi region centroids. This means that one may view k as a knob which changes the behavior of the algorithm from randomized to deterministic (hence we refer to it as partially randomized). We also see that the cost of running this algorithm grows linearly with k, since at each iteration k nearest-neighbor queries must be performed. Presumably, having more Voronoi bias will result in better exploration and consequently fewer nodes in the search tree; however, the cost of each node grows as the Voronoi bias is increased. Hence, the best performance is achieved by finding the best value of the parameter k, which is not necessarily a simple task.

Our second algorithm is similar to the first Multi-Sample RRT. Its differences are based on two key observations. First, if one can obtain approximate Voronoi information from a set of k uniformly distributed random samples, one may also obtain it from k uniformly distributed deterministic samples (the Voronoi information depends only on uniformity, not randomness). Hence, as long as one has a sequence with good incremental quality (i.e., the set of samples $\{s_i, \ldots, s_{i+k}\}$ from the sequence is uniformly distributed for all i, k), one may take k samples from that deterministic sequence and expect the algorithm to work as well as with random samples (in our experiments so far, we have used Halton points [7]). Second, if some set of k samples gives a good approximation, then there is no need to pick k new samples during the next iteration. Instead, the old samples may be used again, saving the cost of doing k nearest-neighbor queries at each iteration. At most, one must do k metric evaluations per new node, which can yield significant savings. To distinguish the algorithm resulting from these observations from the previous one, we denote this one as MS-RRTb; pseudo-code is given in Figure 2. In both of these algorithms, one may reach a point where the k samples no longer provide enough resolution to make a good choice about where to extend the tree; in the case of the MS-RRTa, one may decide to take $k + k_2$ samples each iteration instead of k. In the case of MS-RRTb, one can add k_2 new samples to the previous set and continue as before.

Each of these algorithms takes the key feature of RRTs, Voronoi bias, and uses sampling techniques to increase that bias. Another approach, based on dispersion reduction, is given in [16]. In the next section, we discuss a few implementation details of our algorithms, and some experimental results.

4 Implementation Details and Experimental Results

Under certain situations, the performance of RRT algorithms can be degraded by local minima. For example, imagine the scenario where a node has a large Voronoi region but is prevented from growing due to proximity to an obstacle. In many cases, this causes little trouble because eventually another node will be chosen to expand. However, both of the MS-RRT algorithms exhibit greedy, Voronoi-biased behavior. This causes problems with local minima to be magnified. To address this problem, both of our planners include *obstacle nodes*, which are nodes in the search tree representing configurations in the obstacle region of C-space. They are generated when an obstacle is encountered during a connection attempt. These nodes are not candidates for expansion (i.e., they are all leaves in the tree), but they are allowed to "own" samples. As a result, nodes which are selected for expansion do not repeatedly grow toward a local minimum. Unfortunately, obstacle nodes can have problems of their own. For certain difficult problems (e.g., those with narrow corridors), many obstacle nodes

can be created, which can significantly degrade performance because more samples are required to decide how to grow the tree. Consequently, we are currently investigating alternative approaches which combine the local minimum-avoiding effects of obstacle nodes without their disadvantages.

Each of our new algorithms has bottlenecks that do not appear in standard RRT planners. As seen from Section 3, MS-RRTa does a large number of nearest-neighbor queries; consequently, the performance of this algorithm depends both on the number of samples taken per iteration and on the efficiency of the nearest-neighbor calculation. Doing many nearest-neighbor queries is unavoidable for this method, but it is possible to reduce the cost of these queries to a manageable level. We do this by using a nearest neighbor package based on Kd-trees [2, 18]. Likewise, MS-RRTb has a bottleneck as well: updating the nearest neighbor for each sample after adding new nodes to the tree. Currently, we use the naive approach which calls the metric function for each new node and each sample, updating the nearest neighbor where appropriate. It should be possible to accelerate this by using an appropriate data structure (most likely, some form of a Kd-tree); developing a way to do this is not trivial, however, and we have not yet done so. Once this is accomplished, however, MS-RRTb should speed up significantly, particularly for difficult problems which require large numbers of samples.

A few other implementation details will suffice to introduce some experiments. First, one of the best RRT planners is RRTConCon, a variant which uses two trees (one starting at the initial state, the other at the goal state) and is more greedy than the basic RRT (see [13] for details). Hence, our experiments use corresponding versions of the MS-RRT planners (except where otherwise noted). Also, MS-RRTa defaults to random sampling for a particular iteration if it is unable to grow the tree in the attempted Voronoi-biased manner. In experiments using the basic algorithms, we use the basic RRT which has been modified to attempt to connect to the goal after each iteration (we denote this variant as ModRRT). Finally, our experiments below are all holonomic. This is a departure from typical RRT applications, since RRTs can easily be applied to systems with dynamics. However, our new algorithms are somewhat metric-sensitive (which is to be expected, since they are strongly Voronoi-biased, and the Voronoi diagram depends on the metric), and for systems with dynamics finding an appropriate metric is difficult. We are currently considering ways to resolve this difficulty.

First, we present two two-dimensional examples to illustrate how emphasizing Voronoi bias affects the growth of the search tree. The first example consists of a simple local minimum separating the initial and goal states (see Figure 3). Both MS-RRT planners initially grow into the local minimum and create an obstacle node upon encountering the obstacle; they then grow the other direction and around the local minimum. Observe the effects of randomization in MS-RRTa: while the deterministic MS-RRTb always chooses a single direction to grow, randomization in the MS-RRTa sometimes causes it to grow in both directions (sometimes, it grows in a single direction like MS-RRTb). Also, the modified RRT planner (ModRRT) requires significantly more planning iterations to solve the problem than either MS-RRT algorithm. This is primarily due to the effect of obstacle nodes, which cause MS-RRTs to avoid obstacles more than an ordinary RRT. Our second example is shown in Figure 4. The algorithms behave in a manner similar to the previous example. Third, in our description of the MS-RRTa algorithm, we mentioned that one could view the number of samples taken per iteration as a knob which changes behavior from a small degree of Voronoi bias (as in the basic RRT) to a large degree of Voronoi bias. In Figures 5 and 6, we show how varying the value of k affects the growth patterns.

Finally, we give two six-dimensional problems to illustrate our algorithms' performance for these problems. The algorithms presented are not capable of outperforming RRTConCon with respect to solution time; however, it they do represent reasonable approaches to planning. An alternative RRT-based approach, which is based on incremental dispersion reduction, and uses many of the ideas from this paper, has led to improved performance over RRTConCon, based on our recent experiments [16].

5 Conclusions and Future Work

In conclusion, we have discussed the role of randomization in RRTs and introduced two new algorithms which increase Voronoi bias. By studying these algorithms, insight may be gained into the reasons for RRT algorithms' effectiveness at solving motion planning problems. Decreasing the effect of randomization allows us to isolate certain aspects of the algorithms' behavior, without the inherent "sloppiness" that results from randomization. Understanding the key reasons for RRTs' effectiveness is the first step toward making more efficient planners, which may or may not utilize randomization.

Our long-term goals include developing efficient planners based on insights gained from studying the algorithms presented in this paper, as well as other algorithms from the spectrum of RRT-like planners. In the near future, we plan to implement and study other RRT-like planners similar to those presented here; we hope to examine both sampling-based approaches and those based on explicit Voronoi computations. We also would like to study the performance of different sampling techniques (randomized and deterministic) in these different planners; this will show whether or not randomization is of value in RRT algorithms. We believe that studies of this type will greatly increase our understanding of efficient motion planning and configuration space exploration, and will enable us to develop better-performing

algorithms.

Acknowledgments This work was funded in part by NSF Awards 9875304, 0118146, and 0208891.

References

[1] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE Int. Conf. Robot. & Autom.*, pages 113–120, 1996.

[2] A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 632–637, 2002.

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

[4] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *IEEE Int. Conf. Robot. & Autom.*, pages 709–714, 1995.

[5] M. Garber and M. C. Lin. Constraint-based motion planning using voronoi diagrams. In *Proc. Workshop on Algorithmic Foundation of Robotics*, 2002.

[6] R. Geraerts and M. H. Overmars. A comparative study of probabilistic roadmap planners. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, December 2002.

[7] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numer. Math.*, 2:84–90, 1960.

[8] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. Comput. Geom. & Appl.*, 4:495–512, 1999.

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

[10] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 995–1001, 2000.

[11] S. M. LaValle and M. S. Branicky. On the relationship between classical grid search and probabilistic roadmaps. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, December 2002.

[12] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[13] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001.

[14] S. R. Lindemann and S. M. LaValle. Incremental low-discrepancy lattice methods for motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, 2003.

[15] S. R. Lindemann and S. M. LaValle. Current issues in sampling-based motion planning. In P. Dario and R. Chatila, editors, *Proc. Eighth Int'l Symp. on Robotics Research*. Springer-Verlag, Berlin, 2004. To appear.

[16] S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing voronoi bias in RRTs. In *IEEE IEEE International Conference on Robotics and Automation*, 2004. Under review.

[17] E. Mazer, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. Artificial Intell. Res.*, 9:295–316, November 1998.

[18] D. M. Mount. ANN programming manual. Technical report, Dept. of Computer Science, U. of Maryland, 1998.

[19] C. Pisula, K. Hoff, M. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelerated voronoi sampling. In *Proc. Workshop on Algorithmic Foundation of Robotics*, 2000.

[20] B. Salomon, Maxim Garber, Ming. C. Lin, and Dinesh Manocha. Interactive navigation in complex environments using path planning. In *Proceedings of the ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics*, 2003.

[21] T. Simeon, J.-P. Laumond., and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), 2000.

[22] J. F. Traub, G. W. Wasilkowski, and H. Wozniakowski. *Information-Based Complexity*. Academic Press Professional, Inc., San Diego, 1988.

[23] X. Wang and F. J. Hickernell. Randomized halton sequences. *Mathematical and Computer Modelling*, 32:887–899, 2000.

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

[25] Y. Yu and K. Gupta. On sensor-based roadmap: A framework for motion planning for a manipulator arm in unknown environments. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 1919–1924, 1998.

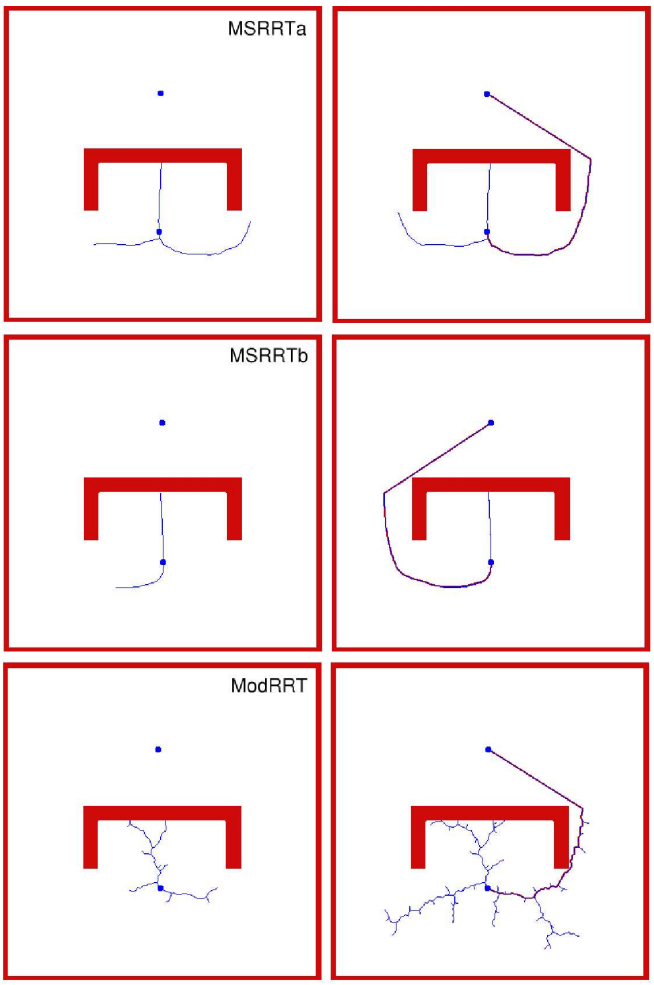

Figure 3: A simple 2-d example. Top row, from left to right: MS-RRTa after 81 iterations (81 nodes, 1 obstacle node), MS-RRTa at completion (114 iterations, 114 nodes, 1 obstacle node). Second row: MS-RRTb after 41 iterations (41 nodes, 1 obstacle node), MS-RRTb at completion (81 iterations, 81 nodes, 1 obstacle node). Third row: ModRRT after 92 iterations (81 nodes), ModRRT at completion (249 nodes, 384 iterations).

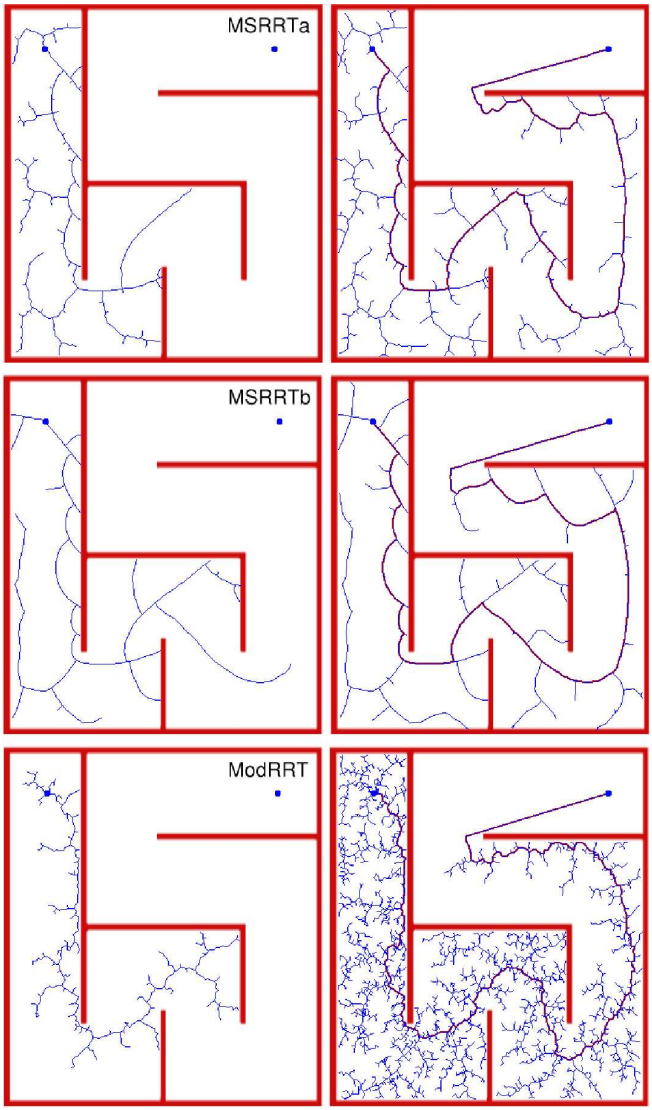

Figure 4: A 2-d maze. Top row, from left to right: MS-RRTa after 400 iterations (400 nodes, 14 obstacle nodes), MS-RRTa at completion (989 iterations, 989 nodes, 39 obstacle nodes). Second row: MS-RRTb after 400 iterations (400 nodes, 14 obstacle nodes),MS-RRTb at completion (762 iterations, 762 nodes, 25 obstacle nodes). Third row: ModRRT after 858 iterations (400 nodes), ModRRT at completion (4869 iterations, 2896 nodes).

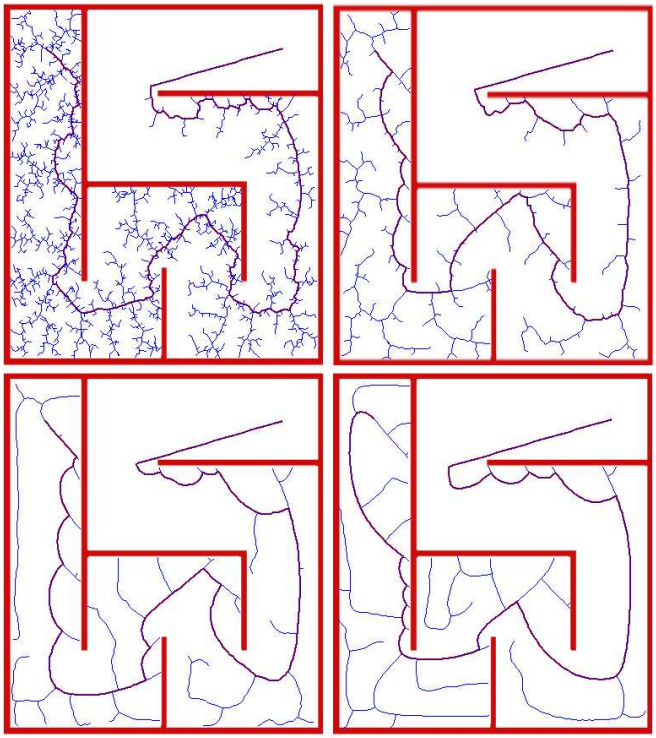

Figure 5: Top row, from left to right: MS-RRTa with $k = 10, k = 100$. Bottom row, MS-RRTa with $k = 1000, k = 10000$.

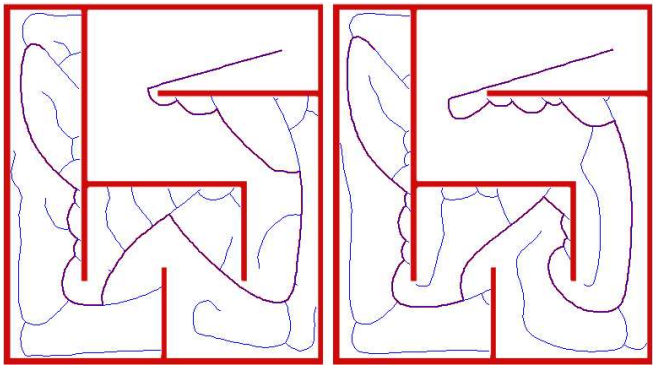

Figure 6: From left to right: MS-RRTb with $k = 1000, k = 10000$.

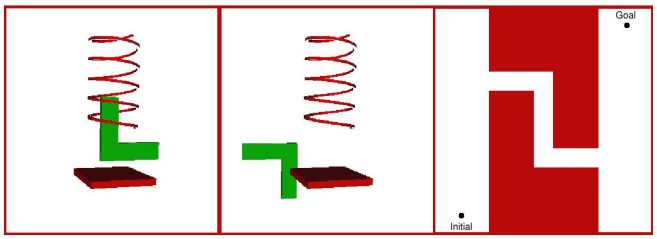

Figure 7: The first two frames are the initial and goal configurations of a 3d rigid body example. The third is a two-dimensional representation of a 6-d bent corridor problem.

Prob.	Dim	RRTConCon	MS-RRTa	MS-RRTb
LM	2	0.01	0.01	0.01
Maze	2	0.19	0.21	0.17
Spring	6	0.7652	8.37	6.01
Corr.	6	101.18	1170	444.6

Figure 8: Comparisons of the planning times required for our experiments. RRTConCon and MS-RRTa results are averaged over 100 trials. Implementations were done in Gnu C++ on a 2.0GHz PC running Linux.

Prob.	Dim	RRTConCon	MS-RRTa	MS-RRTb
LM	2	113.4	91.1 (2.1)	88 (2)
Maze	2	456.49	389.3 (28.7)	450 (31)
Spring	6	2272.5	3392 (76.33)	1875 (31)
Corr.	6	14085	18975 (773)	29223 (1173)

Figure 9: Comparisons of the number of nodes corresponding to the results of the previous figure. Where applicable, the number of obstacle nodes is given in parentheses.

Prob.	Dim	RRTConCon	MS-RRTa	MS-RRTb
LM	2	320.2	274.67	255
Maze	2	1411.41	1279.6	1365
Spring	6	6793.17	11850	6405
Corr.	6	42494	55686	80750

Figure 10: Comparisons of the number of collision checks corresponding to the results of the previous figure.