Incrementally Reducing Dispersion by Increasing Voronoi Bias in RRTs

Stephen R. Lindemann   Steven M. LaValle

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

Abstract

We discuss theoretical and practical issues related to using Rapidly-Exploring Random Trees (RRTs) to incrementally reduce dispersion in the configuration space. The original RRT planners use randomization to create Voronoi bias, which causes the search trees to rapidly explore the state space. We introduce RRT-like planners based on exact Voronoi diagram computation, as well as sampling-based algorithms which approximate their behavior. We give experimental results illustrating how the new algorithms explore the configuration space and how they compare with existing RRT algorithms.

1 Introduction

While sampling-based motion planning algorithms have been widely used for over a decade [2, 4, 9, 14, 8, 10], recent years have seen an increase in attention given to fundamental sampling issues [13]. Most of this work deals with uniform sampling for PRMs [3, 7, 11]. In this paper, we discuss RRTs and how to use sampling techniques to increase their Voronoi bias, with the goal of searching in a way that incrementally reduces dispersion.

Drawn from classical sampling theory, dispersion measures how well a space is covered by a sample set. Formally, the dispersion δ of a point set P is defined to be:

$$\delta(P, \rho) = \sup_{q \in X} \min_{p \in P} \rho(q, p), \qquad (1)$$

in which X is the sample space and ρ a metric on X. Intuitively, dispersion is the radius of the largest empty ball (under ρ) in the space. (For more details about dispersion and uniformity measures, see our previous work in [11] or Niederreiter's comprehensive monograph [15].) For PRMs, one can use dispersion to measure the quality of a sample sequence used to construct the roadmap. For RRTs, the dispersion of the search tree in the configuration space can be seen as indicating the largest region the tree has yet to explore. Hence, a good strategy for exploration is that of *incremental dispersion reduction*: at each step, extend the search tree in such a way as to lower the dispersion as much as possible. Focusing on dispersion reduction promotes growth toward unexplored regions and uniform coverage of the configuration space.

In this paper, we will discuss the original RRT and how its exploration strategy, Voronoi bias, promotes dispersion reduction. We then introduce RRT-inspired planners based on explicit Voronoi diagram construction and which are completely Voronoi-biased. We then give sampling-based algorithms which approximate the behavior of the exact algorithms, combining the greater Voronoi bias of the exact planners with the practical efficiency of RRTs.

2 RRTs and Voronoi Bias

In this section, we will describe RRTs and show how their Voronoi bias leads to dispersion reduction. We will also introduce exact planners based on these ideas.

RRTs are *incremental search algorithms*; they incrementally construct a tree from the initial state to the goal state (bidirectional versions exist as well). At each step, a random sample is taken and its nearest neighbor in the search tree computed. A new node is then created by extending the nearest neighbor toward the random sample. See Figure 1 for pseudo-code; for in-depth description and analysis, see [12].

This exploration strategy has an interesting property: it is characterized by Voronoi bias. At each iteration, the probability that a node is selected is proportional to the volume of its Voronoi region; hence, search is biased toward those nodes with the largest Voronoi regions (representing unexplored regions of the configuration space). This causes RRTs to rapidly explore. Alternatively, RRTs can be seen as attempting to decrease dispersion. The random sample then becomes an estimate of the center of the largest empty ball, and its distance to its nearest neighbor an estimate of the dispersion. The RRT then grows toward that sample, attempting to decrease the dispersion.

It should be noted that for RRTs, Voronoi bias does not play the same role as in some other recent motion planning algorithms [6, 16]. These algorithms compute

```
BUILD_RRT($x_{init}$)
1  $G_{sub}$.init($x_{init}$);
2  for $k = 1$ to $K$ do
3      $x_{rand} \leftarrow$ RANDOM_STATE();
4      $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, G_{sub}$);
5      $u_{best}, x_{new}, success \leftarrow$ CONTROL($x_{near}, x_{rand}, G_{sub}$);
6      if success
7          $G_{sub}$.add_vertex($x_{new}$);
8          $G_{sub}$.add_edge($x_{near}, x_{new}, u_{best}$);
9  Return $G_{sub}$
```

Figure 1: The basic RRT construction algorithm.

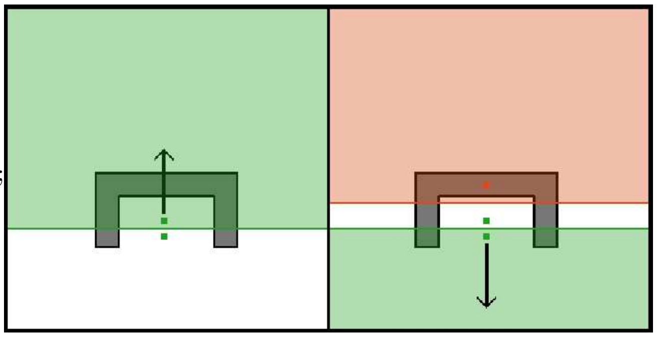

Figure 2: The effect of obstacle nodes: on the left, exploration is trapped by a local minimum; the largest Voronoi region causes growth in the direction of the obstacle. On the right, an obstacle node has been placed which solves this problem; since the obstacle node cannot be selected for expansion, growth proceeds from a different node, away from the local minimum.

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 [17]). In this case, the Voronoi diagram is based on the environment. In RRTs, however, the Voronoi bias is 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.

Using the notion of Voronoi bias, one may easily conceptualize an RRT-like planner based on the exact Voronoi diagram. At each iteration, construct the Voronoi diagram of the search tree (note that this will be a d-dimensional Voronoi diagram). Then, one can use information from the Voronoi diagram to decide which node to expand and in which direction to explore. Two possibilities are readily apparent: either grow from the node with the largest Voronoi region toward its center (this requires calculating the volume of each Voronoi region), or grow toward the Voronoi vertex most distant from the tree from one of its nearest neighbors (there will be $d + 1$ of these, in general). Depending on one's interpretation, either of these approaches is a valid generalization of the basic RRT. The former approach corresponds more closely to the probabilistic interpretation of Voronoi bias (the probability of a node being selected being proportional to its associated Voronoi volume), and the latter more closely to dispersion reduction. In fact, the latter approach is precisely dispersion reduction, since one of the vertices of the bounded Voronoi diagram will be the largest empty ball in the configuration space, and the distance from that vertex to its nearest neighbor is the dispersion.

The above search strategies, as outlined, can easily become trapped when the search tree encounters obstacles in the configuration space (i.e., when it cannot grow in the desired direction due to obstacles). Hence, modifications must be made for these Voronoi-biased planners to be complete. A good way to do this is to introduce "obstacle nodes" into the tree at each point where an obstacle is encountered. The obstacle nodes are used along with the other tree nodes in the Voronoi diagram computation, but they may not be chosen for expansion (they are all leaves in the tree). Using obstacle nodes in this manner tends to make the planner grow along obstacle boundaries and eliminates deadlock conditions. For an illustration of the effect of obstacle nodes, see Figure 2.

Both the volume-based and the dispersion-reduction approaches are 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 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 topologies. Finally, from a purely practical point of view, the high cost of explicitly computing this information may outweigh its value for practical planning purposes.

To illustrate the manner in which an exact Voronoi-based planner explores, we have implemented an exact Voronoi diagram-based RRT for 2-dimensional problems [1]. In Section 3, we will see that only the dispersion-reduction approach is practically feasible for our sampling-based Voronoi-biased planners; hence, we use the dispersion-reduction approach for the implementation of the exact algorithm as well. Figure 3 displays the search trees for two problems.

3 Increasing Voronoi Bias in Practice

3.1 Sampling-based Voronoi-biased planning algorithms

Planners based on exact Voronoi computations suffer from problems similar to those faced by classical combinatorial motion planners. Just as combinatorial

[1]We use CGAL (http://www.cgal.org/) for Voronoi diagram construction.

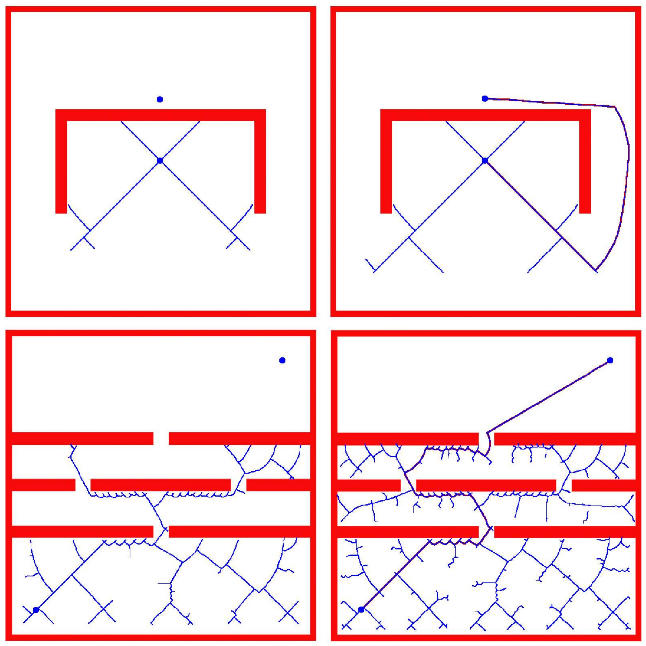
Figure 3: Exploration using an exact Voronoi-biased planner. On the left is the search tree in the middle of the search process; on the right is the final tree. Note the high degree of symmetry and small number of branches.

motion planners constructed an explicit representation of configuration space obstacles, exact Voronoi-biased planners must compute Voronoi diagrams in potentially high dimensions. In each case, complex geometric algorithms must be used to obtain the desired information. Just as the success of modern motion planning algorithms may be attributed to a shift from combinatorial algorithms to sampling-based ones [13], it is reasonable to expect that one may approximate the behavior of exact Voronoi-biased algorithms using sampling-based techniques. Sampling-based algorithms of this type will exhibit a degree of Voronoi bias, with a cost much less than that of exact construction. In this section, we introduce sampling-based versions of both the volume-based and dispersion-reducing Voronoi-biased RRTs.

We have seen that RRTs grow in a Voronoi-biased manner due to the way they process random samples drawn from the configuration space. What would happen, then, if instead of taking a single sample, one took K samples? One can hold an election between the nodes of the search tree; the node which is nearest to the most samples is selected for expansion and grown toward the average of the samples it collected. Then, it holds that the probability of a node being selected is related to its Voronoi volume, corresponding directly to the notion of Voronoi bias in the original RRT. In addition to being a multi-sample version of an RRT, a planner of this type can be viewed as an approximation of the exact planner;

```
BUILD_VB-RRT($x_{init}$)
 1  $G_{sub}$.init($x_{init}$);
 2  **for** $k = 1$ **to** $K$ **do**
 3      ADD_NEW_SAMPLE($s, S$);
 4  FIND_NEAREST_NEIGHBORS($S, G_{sub}$);
 5  **for** $x = 1$ **to** $X$ **do**
 6      $x_{best} = \arg\max(x.sampleCount, x \in G_{sub})$;
 7      $x_{next} \leftarrow x_{best}.sampleAverage$;
 8      $u_{best}, x_{new}, success \leftarrow$ CONTROL($x_{best}, x_{next}, G_{sub}$);
 9      **if** success
10          $G_{sub}$.add_vertex($x_{new}$);
11          $G_{sub}$.add_edge($x_{near}, x_{new}, u_{best}$);
12      FIND_NEAREST_NEIGHBORS($S, G_{sub}$);
13  Return $G_{sub}$
```

Figure 4: The basic volume-based RRT construction algorithm.

the fraction of samples a node collects approximates the volume of its Voronoi region, and the average of its samples an approximation of the center of that region.

Upon reflection, it is apparent that it is not necessary to select K new samples at each iteration. If K samples are selected at some iteration which are uniform over the configuration space, they are equally uniform at the next iteration. Hence, they may be reused. This is highly advantageous, because if samples are not reused, the cost of doing K nearest-neighbor queries per node expansion is prohibitive. If at some point during the search it is determined that the initial K samples are insufficient, more may be added. Obstacle nodes can straightforwardly be incorporated into this volume-based planner and are in fact necessary for its operation (without them, the planner can reach deadlock in the same way as the exact planner). Pseudo-code for this planner (which we denote VB-RRT) is given in Figure 4

Constructing a sampling-based dispersion-reducing planner proceeds similarly. Again take K samples, and sort them in decreasing order of distance from their nearest neighbors in the search tree; this ranks them according to how well they estimate the largest empty ball in the space. Then choose the sample most distant from its nearest neighbor, and grow from that neighbor toward the sample. If this fails (an obstacle is encountered), take the next best sample and repeat the process. If this fails for all K samples, then add more and continue. Obstacle nodes can be incorporated into this planner as well, but unlike the previous case, they are not essential. Even if the planner *appears* "stuck" for a short time, adding additional samples will eventually allow it to proceed. Pseudo-code for this algorithm (denoted DR-RRT) is given in Figure 5. As before, this planner can be interpreted in two different ways. It can be seen as a multi-sample RRT (under the dispersion-reduction interpretation), or as an approximation of an

```
BUILD_DR-RRT($x_{init}$)
1   $G_{sub}$.init($x_{init}$);
2   for $k = 1$ to $K$ do
3       ADD_NEW_SAMPLE($s, S$);
4   FIND_NEAREST_NEIGHBORS($S, G_{sub}$);
5   for $x = 1$ to $X$ do
6       $x_{next} = \arg\max(sample.ownerDistance, s \in S)$;
7       $x_{best} \leftarrow x_{next}.owner$;
8       $u_{best}, x_{new}, success \leftarrow$ CONTROL($x_{best}, x_{next}, G_{sub}$);
9       if success
10          $G_{sub}$.add_vertex($x_{new}$);
11          $G_{sub}$.add_edge($x_{near}, x_{new}, u_{best}$);
12      FIND_NEAREST_NEIGHBORS($S, G_{sub}$);
13  Return $G_{sub}$
```

Figure 5: The basic dispersion-reducing RRT construction algorithm.

exact dispersion-reducing planner.

3.2 Relating Voronoi Bias and Derandomization

Thus far, we have seen two different types of Voronoi bias. In the original RRT (and in its volume-based multi-sample extension), Voronoi bias has to do with the fact that a correlation exists between the probability of a node's selection and the volume of its Voronoi region. Alternatively, one may view Voronoi-biased behavior as that which approximates the behavior of an exact Voronoi-biased planner. While the first point of view is inconsistent with RRT derandomization, the second perspective on Voronoi bias is perfectly in harmony with it [2].

In the original (single-sample) RRT, random sampling is crucial to Voronoi bias. If one replaces these random samples with deterministic ones, no statement about node selection probability can be made. One may still argue that, over time, the deterministic uniform sequence will cause the search tree to grow in all directions equally. It may be that this is sufficient for RRTs to perform well; however, it is no longer Voronoi bias, strictly speaking. Viewing the original RRT from the approximation perspective is tenuous at best, since a single sample can hardly be seen as approximating information from an exact Voronoi diagram.

In contrast with this, the multi-sample RRT planners are much more reasonably viewed as approximations of exact Voronoi-biased planners. Together with the fact that even with random samples there is no simple expression relating the probability of a node being selected to the volume of its Voronoi region, this suggests that random sampling is no longer necessary for claims of Voronoi bias. Hence, either deterministic or randomized sampling methods can be used. It is worthwhile to note that as K approaches infinity, even the use of random samples converges to deterministic behavior. Thus the multi-sample Voronoi diagram approximation perspective provides a solid foundation for derandomizing RRTs.

[2]That is, assuming one doesn't mind a derandomized rapidly-exploring random tree

3.3 Experimental results

Our experimental goals are twofold: first, to gain intuition about Voronoi-biased exploration; and second, to compare the practical performance of our strongly Voronoi-biased planners to standard RRTs. To accomplish the first goal, we show pictures of solutions for twosimple 2-dimensional problems; see Figure 6 for results corresponding to those shown in Figure 3. These are intended to illustrate how Voronoi-biased planners explore the search space. From this figure, one can see that as the number of samples in the sample set S increases, the resulting search tree looks increasingly like the one produced by the exact Voronoi planner. Even with fewer samples, the growth patterns are quite different from the standard RRT, which produces a tree with many more branches.

For the second goal, we present results for several problems of varying difficulty. Since one key benefit of RRTs is their ability to solve problems with nonholonomic constraints, we show how our planner solves these problems in addition to giving results for standard path-planning problems. All experiments use the dispersion-reducing RRT planner, which is more efficient than the volume-based RRT. This is because the use of obstacle nodes, which are necessary for the volume-based RRT, creates a problem similar to the "narrow passage" problem which PRM planners are subject to. Obstacle nodes cause situations in which it is necessary to sample a tiny region in order for the search to make progress. This causes the volume-based RRT to be impractical in its current form. However, the dispersion-reducing RRT does not use obstacle nodes and hence does not suffer from this problem; for the rest of this section, we deal exclusively with DR-RRT.

Our experimental results, corresponding to the problems shown in Figure 7, are shown in Figures 8, 9, and 10. For these experiments, the DR-RRT planner used was a dual-tree version analogous to a dual-tree RRT-Connect. With respect to running time, the DR-RRT showed small improvements for two of the six problems and was significantly better for the two bent corridor problems. For the six-dimensional bent corridor, the speedup factor was nearly 5; for the eight-dimensional corridor, the success rate was over twice that of RRT-Connect. With respect to numbers of search tree nodes and collision checks, DR-RRT was better regularly superior to RRT-Connect. Often, DR-RRT required 25-50% of the nodes and collision checks required by RRT-Connect. The difference between execution time results and node and collision check results suggests that the current per node cost of

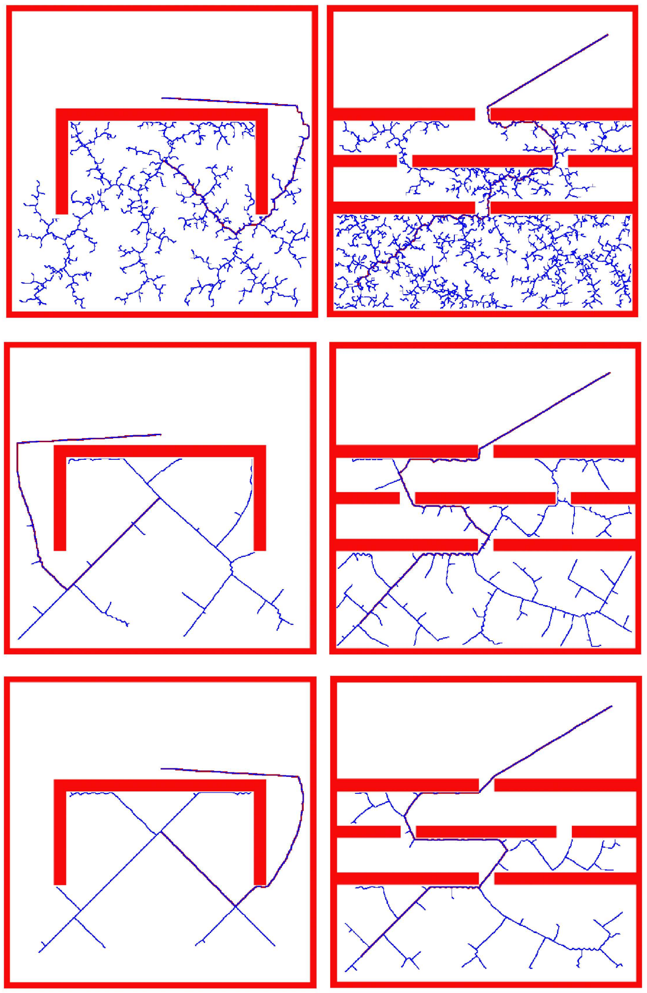

Figure 6: Varying degrees of sampling-based Voronoi bias. On the top is a standard RRT; in the middle a dispersion-reducing RRT with initial $|S| = 100$; on the bottom a dispersion-reducing RRT with initial $|S| = 1000$.

DR-RRT is high. We believe that this is in fact the case, and we will discuss ways which the algorithm and its implementation can be improved to decrease per node cost and consequently execution time. Overall, these initial results are very encouraging; in conjunction with the improvements we will now discuss, they show potential for a highly effective and practical motion planner.

The simplicity of the basic RRT algorithm is a source of both elegance and efficiency. In the quest to achieve more Voronoi-biased exploration, the planners introduced in this paper lose some of this simplicity. It is consequently important to carefully examine the sources of computational cost in these Voronoi-biased planners. Doing so will allow interpretation of experimental results as well as illuminating areas that, when improved, will yield a better-performing planner. First, examine the costs associated with building an RRT. At each iteration, a sample is taken, its nearest neighbor in the search tree is found, and an attempt to expand that node is made. If large portions of the configuration space are unreachable from their nearest neighbors, it may take many iterations before the expansion is successful and a new tree node is produced. This can happen when the space has local minima, for example. Now, consider the cost involved in building a DR-RRT. At each iteration, a sample s is chosen from S as the most desirable direction to expand. Its nearest neighbor is already known (the nearest neighbor is computed once when a sample is originally added to S, and updated thereafter), and an expansion is attempted. If a new node is created, then it must be determined which samples this new node is nearest neighbor to, and updates made accordingly. This step is one of the chief bottlenecks of the algorithm; in the worst case, it is possible that $|S|$ metric computations will have to be performed. Using the insight that the query being performed (i.e., given a set of balls and a query point, return all balls the point lies inside of) is similar to a stabbing query [5], one can build data structures to reduce the number of metric calls required. Given a set of axis-aligned boxes and a query point, a stabbing query returns every box which contains the query point. Using multi-level segment trees, these queries can be answered in $O(\log^d n + k)$ time (our current approach is somewhat less sophisticated). Since we are actually interested in balls, not boxes, we approximate the balls by their bounding boxes and use those in the data structure. Each returned box is then checked to see if the ball it represents contains the query point. Depending on the size of the ball radii, there is significant potential for metric call reduction.

The simple data structure we implemented is suitable for an ℓ^2 metric on a d-dimensional space of the form $R^i \times (S^1)^{d-i}$; test results for metric call savings are shown in Figure 11. We implemented versions that found the best splitting for each cell and those which tested only a small subset of splittings, and found the latter to be more cost-effective. The tests were run by

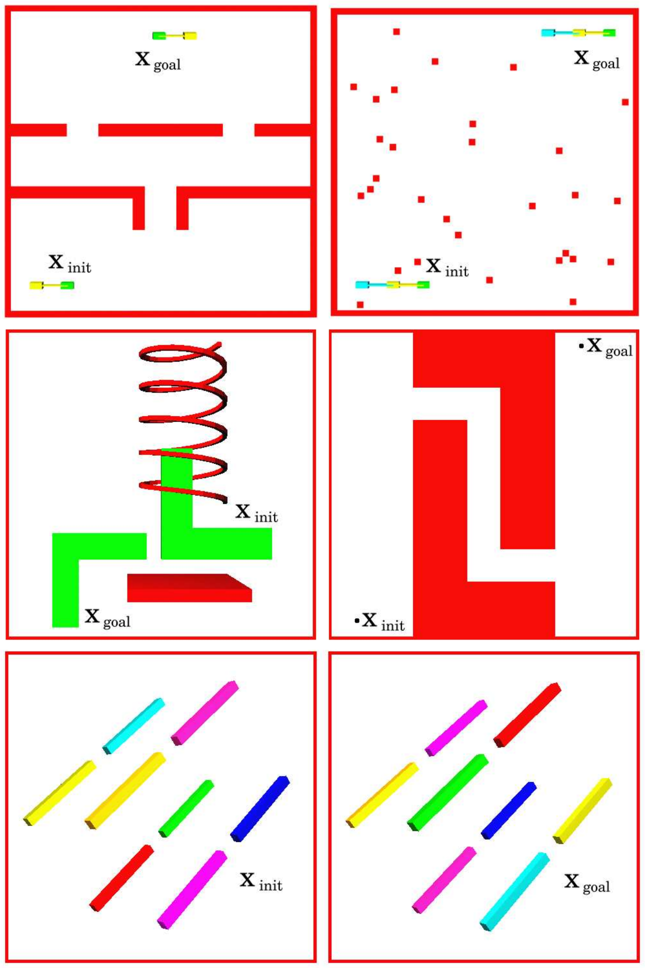

Figure 7: Snapshots of problem environments. Top to bottom, left to right, they are: a car with a trailer (5 dof); a car with two trailers (6 dof); a rigid body problem (6 dof); a 2-d projection of a bent corridor (6, 8 dof); and a multi-body problem (48 dof). The radius of the bent corridor is 0.15.

Problem	Dim	RRT-Connect	DR-RRT
Trailer (N)	5	175.69 (47)	130.13 (44)
2 Trailers (N)	6	35.48 (50)	52.81 (49)
Spring	6	0.25 (50)	0.4 (50)
Corridor	6	75.21 (50)	15.75 (50)
Corridor	8	361.55 (17)	257.98 (36)
Multi-body	48	119.34 (50)	178.82 (50)

Figure 8: Comparisons of the planning times required for our experiments (in seconds). Results are averaged over 50 trials; the two nonholonomic problems are denoted by (N). The corridor experiments were terminated if solutions were not found within 1200 seconds; all other experiments were terminated if solutions were not found within 600 seconds. The numbers in parentheses give number of successes out of 50. All experiments were done on a 2.4 GHz PC running Linux.

Problem	Dim	RRT-Connect	DR-RRT
Trailer (N)	5	5782.38	3439.27
2 Trailers (N)	6	2685.2	1764.8
Spring	6	1972.12	1710.72
Corridor	6	17706.38	4030.8
Corridor	8	50247.35	25005.5
Multi-body	48	14848.62	17044.8

Figure 9: Comparisons of the numbers of nodes required for our experiments.

Problem	Dim	RRT-Connect	DR-RRT
Trailer (N)	5	2260261	300551
2 Trailers (N)	6	675464	186774
Spring	6	5875	7269
Corridor	6	53534	15011
Corridor	8	150148	85063
Multi-body	48	44800	70887

Figure 10: Comparisons of the numbers of collision checks required for our experiments.

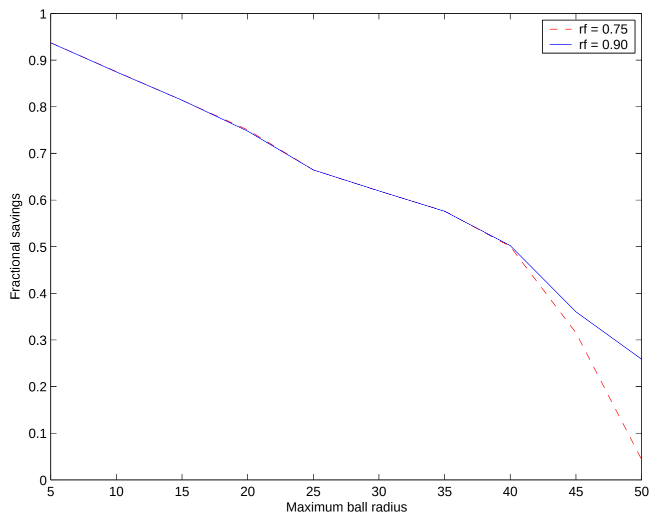

Figure 11: Plots of metric call savings for a simple data structure answering stabbing queries. The rf parameter represents the strictness of the splitting requirements (higher values are more lenient). Each data point represents the average of 100 trials.

generating 1000 6-dimensional balls, with random center and radius (the radius was chosen uniformly up to some maximum value), building the data structure, and then making 1000 random queries. The time required to build the data structure was consistently less than 5 ms.

A disadvantage of this approach is that it introduces the costs of building additional data structures, incrementally updating them (if possible), and determining when they need to be rebuilt. Given the need for the planner to work well for many different topologies and metrics, adapting traditional stabbing query data structures while making them as fast as possible is a challenge (perhaps inspiration may be drawn from [1]). At the present time, our implementations are only partly effective in reducing the solution time; there is room for improvement.

Making the nearest-neighbor update step as efficient as possible is likely the single largest practical challenge to improve the performance of DR-RRT. Another place where changes could lead to improved performance is the sample selection criteria. Currently, the DR-RRT chooses a sample from S as representing a good exploration direction (specifically, as being the best estimate of the largest empty ball in the configuration space). This selection rule is motivated solely by the desire to induce Voronoi bias in the planner, and does not take into account any other considerations. For example, there could be a sample which has been unsuccessfully grown toward by a large number of different nodes in the tree (in whose Voronoi regions the sample lied at some point in the search). If a new node is generated and becomes the nearest neighbor to this sample, then it is likely that another connection attempt will be made, in spite of all the previous failures. For a problem using thousands of samples, it is possible (and, perhaps, common), for the creation of a new node to immediately result in hundreds of unsuccessful connection attempts (and corresponding collision checks). Hence, there is significant potential for savings if the planner is more careful about which samples it chooses to expand toward, taking into account more information than simply a sample's distance to its nearest neighbor.

4 Conclusions and Future Work

In conclusion, we discussed the exploration strategy of RRTs, their Voronoi bias, and the goal of dispersion reduction. We then introduced exact planners designed that explore using the Voronoi diagram of the search tree. These exact planners are capable of finding the largest empty ball in the configuration space and shrinking it, thus reducing dispersion. Finally, we presented sampling-based algorithms which can approximate, with arbitrary precision, the behavior of the exact algorithms, for more general problems and with less cost. These sampling-based algorithms can effectively use either deterministic or random samples. The use of high numbers of samples promotes predictable and highly Voronoi-biased behavior, but at a high cost; low numbers of samples yield greater Voronoi bias than RRTs, while still keeping the cost low. These planners heavily bias search toward unexplored regions of the configuration space, focusing on reducing dispersion.

The experiments presented in this paper indicate both that the sampling-based dispersion-reducing RRT works well in practice and that there is ample room for improvement. In the future, we plan to address the two main practical considerations addressed above: efficient nearest-neighbor update and the sample selection criteria. Making improvements in these areas will hopefully result in a planner that is both satisfying in its theoretical basis and highly efficient in its operation.

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

References

[1] 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.

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

[3] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1481–1487, 2001.

[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. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 1997.

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

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

[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, M. S. Branicky, and S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*. Selected for publication in late 2003 from among papers that appeared at the 2002 Workshop on the Algorithmic Foundations of Robotics.

[12] 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.

[13] S. R. Lindemann and S. M. LaValle. Steps toward derandomizing RRTs. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2003. Under review.

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

[15] H. Niederreiter. *Random Number Generation and Quasi-Monte-Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1992.

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

[17] 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.