Iteratively Locating Voronoi Vertices for Dispersion Estimation

Stephen R. Lindemann and Peng Cheng
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA
{*slindema, pcheng1*}*@uiuc.edu*

***Abstract*— We present a new sampling-based algorithm for iteratively locating Voronoi vertices of a point set in the unit cube $I^d = [0,1]^d$. The algorithm takes an input sample and executes a series of transformations, each of which projects the sample to a new face of the Voronoi cell in which it is located. After d such transformations, the sample has been transformed into a Voronoi vertex. Locating Voronoi vertices has many potential applications for motion planning, such as estimating dispersion for coverage and verification applications, and providing information useful for Voronoi-biased or multiple-tree planning. We prove theoretical results regarding our algorithm, and give experimental results comparing it to naive sampling for the problem of dispersion estimation.**

I. Introduction

Sampling-based algorithms have been applied to motion planning problems for over a decade [5], [7], [12]–[14], [21]. Multiple-query algorithms like PRMs construct graphs which attempt to represent the connectivity of free space [1], [2], [13], [16], [24], [27]; single-query algorithms construct search graphs which attempt to solve a particular motion planning query [12], [14], [15], [21]. Typically, both multiple- and single-query methods build graphs in which the vertices are points in the state space or the configuration space. Considering the Voronoi diagram of these points can be advantageous; for example, Voronoi information can be used, in varying degrees, to bias the growth of the search tree [14], [17]. Dispersion, which is a uniformity measure closely related to Voronoi diagrams, is a natural way to estimate coverage [9], [15]. In this paper, we present a sampling-based technique for locating the vertices of the Voronoi diagram of a point set P. We assume throughout that the Voronoi diagram is constructed using the standard ℓ^2 metric. In brief, our method takes samples and transforms them into Voronoi vertices by successive projections onto the boundary hyperplanes of the Voronoi facet in which the sample lies. See Figure 1 for an illustration of this process. The Voronoi diagram of the point set is given, and four different samples were taken. The arrows denote the transformations of each sample via projection onto a hyperplane (in two dimensions, there are two such transformations per sample).

Locating the vertices of the Voronoi diagram is very useful and has a number of important applications. The locations of these vertices represent large unexplored regions of the space, and hence provide a natural heuristic for choosing a direction to explore. Consequently, our techniques can be directly integrated into Voronoi-biased planners such as DR-RRT [17]. Second, the distances from the vertices to their nearest neighbors can be used to estimate coverage. Coverage estimation is particularly relevant for *safety falsification tools* applied to verification problems [6], [9]. Rather than proving conclusively that a system is safe (which is extremely difficult to do [10], [11]), a safety falsification tool attempts to disprove safety by finding a trajectory which reaches an unsafe state. Coverage estimation may be useful for providing deterministic safety guarantees when the safety falsification tool is unable to find an unsafe trajectory. Third, Voronoi vertices can be used to inform multiple-tree planners such as [1], [9], [25]. One of the key questions which multiple-tree algorithms must answer is when (and where) to start a new search tree. Since Voronoi vertices represent empty balls in the space, having a set of Voronoi vertices available provides a natural set of candidate locations for starting a new tree. Finally, the set of Voronoi vertices can be used directly to estimate the dispersion of the point set.

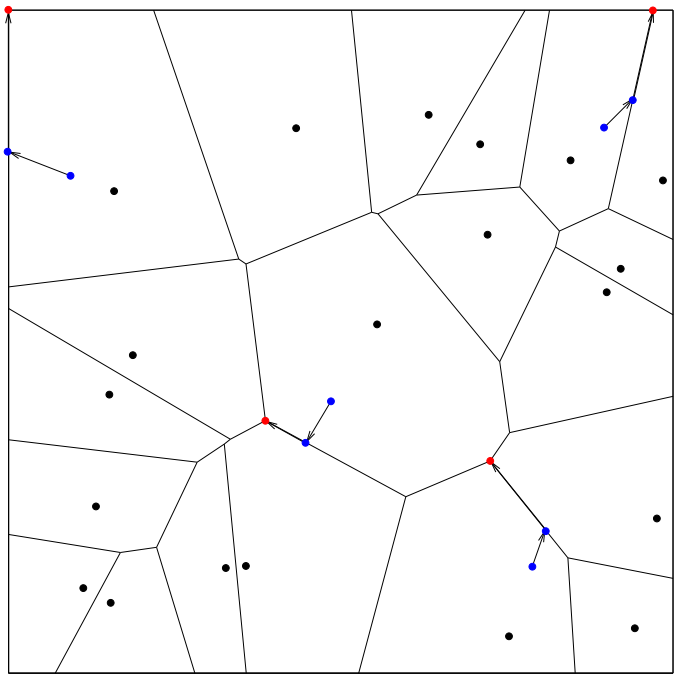

Fig. 1. An illustration of the operation of our algorithm in two dimensions. Given a point set P and its associated Voronoi diagram, the algorithm takes samples from the space and transforms them into Voronoi vertices.

Dispersion is a well-known uniformity measure from classical sampling theory. Given a space X and a point set $P \subset X$, the dispersion of P is defined to be:

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

in which ρ is a metric on X. Intuitively, dispersion is the radius of the largest empty ball (under ρ) in the space. See Niederreiter's monograph [22] for a detailed discussion of dispersion. The dispersion of a point set is intimately connected to the set's Voronoi diagram, because Voronoi vertices are the centers of empty balls in the space, each with a radius which is locally greatest.

II. Related Work

The problem of locating Voronoi vertices has not been extensively addressed in the literature. Most related work in computational geometry focuses on the computation of the entire Voronoi diagram (or, perhaps, the set of all Voronoi vertices), rather than a single vertex or subset of vertices. However, locating Voronoi vertices has been addressed in several places, which we will mention. Most often, the goal is to find the vertices of the *generalized Voronoi graph* (GVG), in which the sites are not points but arbitrary shapes.

Choset and Nagatani address locating Voronoi vertices (they term them *meet points*) in the context of topological simultaneous localization and mapping (SLAM) [8]. Their approach uses a mobile robot to generate the edges of the GVG by an appropriately defined control law. A key difference between their work and the algorithm we present is the fact that their method is designed to be executed by a mobile robot. The robot must move continuously, rather than taking discrete jumps as our method does. Hence, it is able to take advantage of local sensor data to determine a direction to move in for the next time step.

Other related work (also in the context of the GVG) is that of Ferrucci *et al.* [26]; a similar approach with improved convergence results is is given by Anton *et al.* [3]. In brief, their approach is to take a sample at random, compute the nearest point on each of the obstacles, and choose the circumcenter of the nearest of these points as the next sample. They continue this procedure until a Voronoi vertex is found.[1] It is interesting to note the similarities between this approach and Lloyd's algorithm for k-means clustering [19]. In many cases, our algorithm will behave similar to these approaches; however, our method improves upon them in several respects. The most important improvement is that there are cases for these algorithms for which a Voronoi vertex with a small distance to its nearest neighbors has a very large convergence region; our method gives such a vertex a much smaller convergence region and gives the larger convergence regions to Voronoi vertices which are far from their nearest neighbors. An additional difference is that these algorithms are not guaranteed to find a Voronoi vertex (the sequence may oscillate with some finite period). This is the case even when applied to Voronoi diagrams of point sets, rather than generalized Voronoi graphs. In contrast, our algorithm will always terminate at a Voronoi vertex. Finally, these algorithms are not local in the sense that given a sample, the final result is not guaranteed to lie inside the same Voronoi cell as the original sample. Our algorithm always terminates at a Voronoi vertex of the Voronoi cell containing the original sample. This may not be important in the problem of dispersion estimation, but is very relevant if using Voronoi vertex location in the context of coverage estimation or Voronoi-biased planning.

[1] It is possible for the resulting sequence to oscillate with some finite period, rather than terminating at a Voronoi vertex.

Finally, linear programming algorithms can be applied to the problem of locating Voronoi vertices. These may perform well in some situations; however, they are more powerful than is necessary for our problem, and they typically scale very poorly with dimension. Many deterministic algorithms have complexity which is exponential in dimension, and the simple randomized incremental algorithm by Seidel has complexity $O(d^4 d! n)$ [23]. In contrast, our algorithm is very simple and scales well with dimension; for this reason, we expect our algorithm to be more efficient. However, a comparison of our method with linear programming is a subject for future investigation. It may be possible to "scale down" an algorithm like the simplex algorithm so that instead of searching for an optimal vertex, it terminates after the first vertex it encounters. This could be useful for finding Voronoi vertices or estimating dispersion, but we have not examined this to date.

III. Algorithm

In this section, we present an algorithm for finding a Voronoi vertex on the bounded Voronoi diagram of a point set $P \subset I^d = [0, 1]^d$ of n points. We assume that the points are in general position; specifically, we require that in d dimensions no $d+2$ points be cospherical. We consider the vertices of the *bounded* Voronoi diagram; the boundary is taken to be the set of hyperplanes corresponding to $x_i = 0$ or $x_i = 1$, $1 \leq i \leq d$. Each boundary hyperplane is taken to be a facet of the Voronoi diagram. Running our algorithm a number of times will yield a subset of the Voronoi vertices of P.

In brief, our algorithm takes a sample and iteratively transforms this sample until it is a Voronoi vertex. In d dimensions, d transformations are required; after k transformations, the transformed sample satisfies k hyperplane constraints, consisting of point equidistance constraints and boundary constraints. In the case where all constraints are equidistance constraints, after d iterations the sample will be equidistant from $d + 1$ points, which is the case for a Voronoi vertex. See Figure 1 for an illustration.

After the k-th iteration, the sample satisfies k constraints; let k_e be the number of equidistance constraints and k_b the number of boundary constraints (note that $k_e + k_b = k$). Then during the $(k+1)$-th iteration, the sample is transformed to satisfy one additional constraint, while maintaining the previous constraints. Let s^k be the location of the sample after k iterations, and let $p^1, p^2, \ldots, p^{d+1}$ be its $d + 1$ nearest neighbors in P, in ascending order of distance. Let $d_i =$

$\rho(s^k, p^i)$; note that $d_1 = d_2 = \cdots = d_{k_e+1}$. Let $b(i,j) = s_i^k - j, j \in \{0,1\}$. The transformation vector $t^{k+1} = s^{k+1} - s^k$ must maintain the existing constraints; in other words,

$$t^{k+1} \in \text{Null} \begin{bmatrix} d_1 - d_2 \\ \vdots \\ d_1 - d_{k_e+1} \\ b_1(i_1, j_1) \\ \vdots \\ b_{k_b}(i_{k_b}, j_{k_b}) \end{bmatrix}.$$

We choose the direction of the transformation vector t^{k+1} to be that of the projection of the negative gradient $-\nabla d_1$ onto this null space (denote the normalized vector in this direction as u^{k+1}). This guarantees that the sample moves away from its nearest neighbor as much as possible. Since we require that $d_1 - d_2 = 0$, etc., we also guarantee that the sample moves away from all its nearest neighbors as much as possible. So, we have that $t^{k+1} = \alpha u^{k+1}$; all that remains is to determine α. We require that the sample remain within the Voronoi cell that s^0 was in; hence, α should be the smallest value for which $s^k + \alpha u^{k+1}$ lies on a hyperplane that is not an existing constraint.

Essentially, this is the problem of ray shooting in a convex polytope, which is well-studied (see [20], for example). However, methods with fast (i.e., logarithmic) query time require significant preprocessing and utilize exponential-space (in dimension) data structures, making them unsuitable for our application. Hence, we do not adopt such methods, taking a more simplistic approach. Clearly, it is sufficient to look for intersections with the hyperplanes specified by $d_1 - d_2 = 0, d_1 - d_3 = 0, d_1 - d_n = 0$, as well as the boundary hyperplanes; in high dimensions, this linear-time approach may be the best that can be done. In low to moderate dimensions, however, we attempt to speed up the ray shooting procedure through the use of binary search, together with $(d+1)$-nearest neighbor checks.[2] In the worst case, however, our algorithm runs in $O(n)$ time, in which n is the size of the point set.

We begin the binary search by initializing the endpoints of the search interval. The minimum value, α_{min}, we take to be zero; this requires that the sample move away from its nearest neighbor. In order to find the maximum value, α_{max}, we determine where the ray $s^k + \alpha u^{k+1}, \alpha \geq 0$ intersects the boundary, ignoring any boundary hyperplanes the ray may already lie in. Once the interval endpoints have been established, the intersections of the ray with the hyperplanes $d_1 - d_{k_e+1} = 0, \ldots, d_1 - d_n = 0$ are computed. Recall that the hyperplanes $d_1 - d_2 = 0, \ldots, d_1 - d_{k_e} = 0$ are the constraints already being satisfied. The minimum α corresponding to these intersections is taken as the current estimate α_e. A new sample estimate $s_e = s^k + \alpha_e u^{k+1}$ is then computed, and its $(d+1)$-nearest neighbors are found. If s_e is in a different Voronoi region than s^k, then we know that α_e is too large; hence, we set $\alpha_{max} = \alpha_e$ and $\alpha_e = (\alpha_{max} + \alpha_{min})/2$. If s_e is in the same Voronoi region but does not lie on a new hyperplane, we check the hyperplanes corresponding to the $(d+1)$ nearest neighbors of s_e. If a candidate intersection is found, then we set α_e accordingly; otherwise, we set $\alpha_{min} = \alpha_e$ and $\alpha_e = (\alpha_{max} + \alpha_{min})/2$. Finally, if s_e is in the same Voronoi region and satisfies a new hyperplane constraint, then we have found the first intersection and the search terminates.

[2]Using nearest-neighbor software such as ANN [4], we can do approximate nearest neighbors in logarithmic time. Using these data structures, exact nearest neighbors can often be found very quickly as well. However, empirical evidence suggests that these techniques do not offer signifi cant benefi t in high dimensions.

IV. Theoretical Analysis

In this section, we prove several results regarding the convergence and termination of our algorithm. For a d-dimensional convex polytope $P \subset \Re^d$, we define the convergence region of a vertex v to the set of points inside P which our algorithm transforms to v. First, we show each vertex of has a convergence region of positive measure. Then, this can be applied to prove that every vertex in the Voronoi diagram has a positive probability of being returned, assuming a uniform random input sequence.

Lemma 4.1: Consider a d-dimensional convex polytope $P \subset \Re^d$ consisting of the intersection of a finite number of closed positive halfspaces (additionally, let P be bounded), and consider a point $s \in \Re^d$, but not necessarily in P. Note that each vertex v of P is the intersection of d hyperplanes. If s is in the intersection of the d positive halfspaces incident at v, then v has a convergence region of positive measure.

Proof: We prove this by induction on the dimension of the space. For $d = 1$, the statement is clearly true. The polytope P is a closed segment; say, $[x_0, x_1]$. If $s \leq x_0$, then every point on the segment except x_0 will converge to x_1; if $s \geq x_1$, then every point on the segment except x_1 will converge to x_0. In each case, s lies in the positive halfspace corresponding to the vertex to which the points converge. In the case where $x_0 < s < x_1$, we have some points on the segment which converge to x_0, and some which converge to x_1; again, the statement is verified.

Now, we proceed to the inductive step. Assume that the statement is true in d dimensions, and consider dimension $d+1$. Again, we have a point s which lies in the intersection of the $d+1$ positive halfspaces incident at vertex v. Now consider a new point s_i, which is the projection of s onto the hyperplane h_i, one of the hyperplanes incident at v. Now, it is always possible to find a hyperplane h_i and corresponding s_i such that s_i is in the intersection of the other positive halfspaces incident at v. This is fairly obvious, but we will state how this can be done. Consider the projections of s on each of the hyperplanes h_i; this results in a set $S = \{s_i\}_1^{d+1}$. Now, there is at least one element of S which is closest to the original point s (there may be several); let this point be s_i. Then, s_i is in the intersection of all other positive halfspaces incident at v. This is true because s_i being closest implies that it has not crossed any other hyperplane (if it had, some other point in S would be the closest). This result holds even in the case where there are multiple nearest neighbors. This means that we can consider the problem in a lower-dimensional space; the

polytope in d dimensions is the face of the $(d+1)$-dimensional polytope which is supported by the d-dimensional hyperplane h_i, and the new point is the projection s_i of s onto h_i. Then, we know that there is a set of positive measure (as measured in d dimensions) which converges to the vertex v, since we can apply the inductive hypothesis. Finally, all we need to do is to move from the set of positive measure in d dimensions to one of positive measure in $d+1$ dimensions. This is almost trivial; consider the set of all points which interpolate between s the points in h_i which converge to v; this is clearly a set of positive measure. Every point in this set will converge to the region of the hyperplane which will converge to v. However, because s may lie outside of the polytope P, the resulting set is not entirely inside P. It is readily apparent, though, that the intersection of this set with P also has positive measure. Thus, the inductive step has been proven. ■

Theorem 4.2: The convergence region of each Voronoi vertex has strictly positive measure.

Proof: The convergence region of a Voronoi vertex v is defined to be the union of the convergence regions of that vertex in each Voronoi cell for which it is a part. Each of these cells is a convex polytope P in d dimensions, and its site s lies inside P. Hence, s lies in the intersection of the positive halfpaces incident at every vertex of P. By the above lemma, this implies that every vertex has a convergence region of positive measure in each Voronoi cell for which it is a part. Since each Voronoi vertex is part of at least one Voronoi cell, the convergence region of each Voronoi vertex has strictly positive measure. ■

From the previous theorem, we understand that each Voronoi vertex has some positive probability of being selected (assuming a uniform random input sequence). This leads to the following result:

Proposition 4.3: Let the convergence region of the dispersion-inducing Voronoi vertex have measure μ. If the input sample sequence is uniform random, then the probability of finding the exact dispersion after executing our algorithm n times is $1-(1-\mu)^n$.

Proof: This is a straightforward application of basic probability laws. Let the dispersion-inducing vertex be v. If the convergence region of v has measure μ, then the probability of our algorithm returning v after any execution is μ. Hence, the probability of it not returning v is $1-\mu$. Then, the probability of it not returning v after n iterations is $(1-\mu)^n$, and hence the probability of it returning v for one of the first n iterations is $1-(1-\mu)^n$. ■

Hence, we see that the probability of finding the exact dispersion with our algorithm approaches one exponentially with the number of executions of the algorithm. This, of course, depends on the fact that each Voronoi vertex has a convergence region of strictly positive measure.

Finally, we wish to show that the binary search associated with the ray-shooting query is guaranteed to terminate.

Proposition 4.4: The binary search method of finding the result of a ray-shooting query in a Voronoi cell is guaranteed to succeed in finite time.

Proof: Consider a ray shooting query originating at location s_i in the Voronoi cell of site p_1, and let its intersection of the ray with the first hyperplane, h, be s_{i+1}. If h represents a boundary constraint, then we are done since our algorithm checks boundary constraints on the first iteration of the binary search. Hence, assume that h arises from an equidistance constraint between p_1 and another point, p_2. We know that point s_{i+1} is equidistant from p_1 and p_2; this naturally implies that p_2 is one of the $d+1$ nearest neighbors of s_{i+1}. However, it is also true there is an open interval on the segment between s_i and s_{i+1} such that for every point in that interval, p_2 is one of the $d+1$ nearest neighbors. This follows immediately from the general position assumption we make about the point set P. Since this interval is open, the binary search procedure will reach the interval in finite time. When the search reaches the interval, the intersection of the ray with h will be computed, and the search will terminate. ■

Before proceeding to show the empirical performance of our algorithm, we show the following negative result regarding the deterministic performance of our algorithm.

Proposition 4.5: Let the point set P have dispersion δ_P. Consider an input sample sequence S with dispersion δ_S. There exists no α, $0 < \alpha < 1$, such that if $\delta_S < \alpha\delta_P$, our algorithm can guarantee that the exact dispersion of P has been computed.

Proof: See Figure 2 for an example of how a point set can be constructed which has fixed dispersion but for which the convergence region of the dispersion-inducing vertex is arbitrarily small. Intuitively, the size of the the convergence region depends greatly on how close nearby points are to being cospherial. By making points arbitrarily close to being cospherical, the convergence region can be made arbitrarily small; this implies that a point set of arbitrarily small dispersion would be required in order to guarantee that a sample will fall in the convergence region. ■

V. Experimental Results

We performed a number of experiments for the problem of dispersion estimation. We compared the efficiency of our algorithm to naive sampling for point sets in 5, 10, 15, and 20 dimensions. Results for the 10- and 20-dimensional experiments can be seen in Figures 3 and 4. In all cases, the size of the point set was $|P| = 50$. For each experiment, we did 20 runs on each of 20 different point sets. For each point set, we averaged the results for the 50 runs, then normalized them by the maximum dispersion value attained on any run by eithermethod. We thus obtained values on the range [0, 1] for each point set; we then averaged the results across all point sets for that dimension.

We consistently found improvements of greater than 25% in the quality of dispersion estimation versus naive sampling. Dispersion estimation quality was measured versus the number of nearest neighbor checks performed, since the time spent in nearest neighbor checks represents the bulk of the running time of the algorithm.

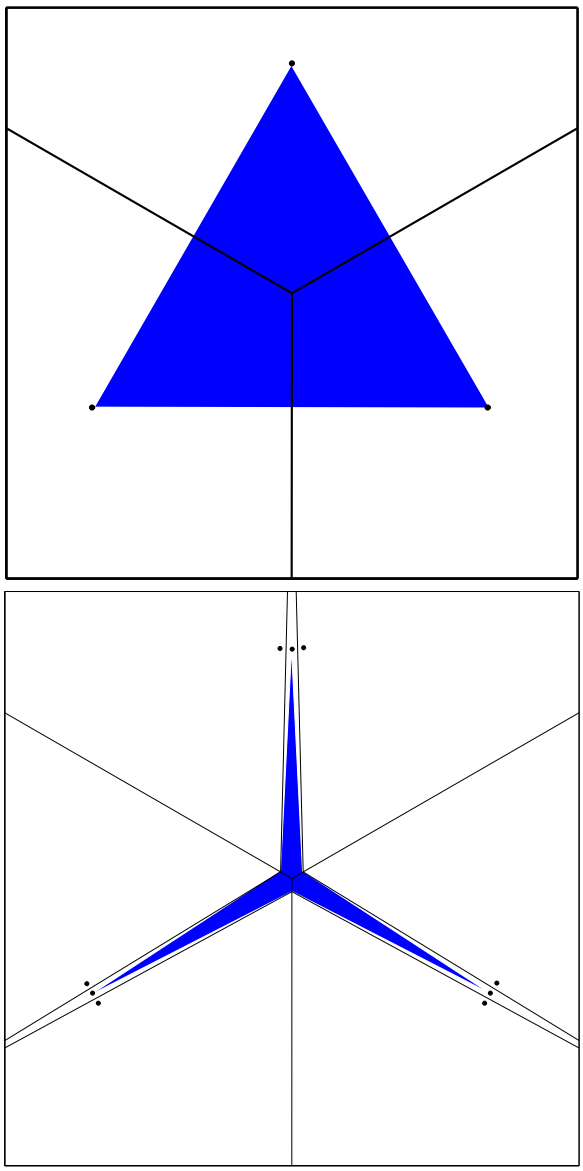

Fig. 2. On the top, the Voronoi diagram of a point set; the convergence region of the central vertex is shaded. On the bottom, the addition of several points to the set does not change the dispersion but can make the convergence region arbitrarily small.

In addition to naive sampling, we compared our method to that of Ferrucci *et al.* [26]; however, their method did not perform well for this problem, and the results are not reported in our figures. Their algorithm frequently found vertices outside the unit cube, which cannot be used to estimate dispersion. Note that this problem did not appear in [26] because they were attempting to locate Voronoi vertices of a generalized Voronoi graph, and the boundaries of the environment effectively force the algorithm to converge inside the environment. If the outside vertices are simply discarded, their method performs worse than naive sampling for our problem; if mapped to the nearest corner, the method performs quite well (better than naive sampling or our method). This is due to the fact that the corners are typically quite distant from their nearest neighbors. However, we do not believe that this improvement is due to any particular property of the algorithm, but is entirely caused by boundary effects. In support of this assertion, we noticed that their algorithm's performance improved relative to the others as dimension increased, which is entirely consistent since boundary effects increase as well.

This raises an interesting question: would it be better just to guess corners (randomly, or using the grid-sampling methods of [18]) in order to estimate dispersion? This might have some limited usefulness, particularly if a numerical estimate of dispersion is all that is desired. However, the Voronoi vertices that our method finds provide a much more comprehensive portrait of coverage than corner information yields. The Voronoi vertices that our method finds give a great deal of local coverage information, which is more useful than information about the corners, which is subject to greater boundary effects. Therefore, we do not believe that corner sampling is particularly useful for coverage estimation in general.

VI. Conclusion

In conclusion, we have introduced a new algorithm for locating Voronoi vertices of point sets in any dimension; our method will find any Voronoi vertex with positive probability. Locating Voronoi vertices has many benefits; they can be used to guide Voronoi-biased planners, used to inform multiple-tree planners, and used for coverage estimation for verification applications. Voronoi vertices can be used to estimate dispersion; experimental results show a significant improvement in estimation quality over naive sampling-based estimation.

In the future, we plan to work to reduce the number of nearest neighbor checks required. Currently, in d dimensions, each sample requires a minimum of d nearest neighbor checks to be transformed into a Voronoi vertex; this could be too expensive in high dimensions. We plan to integrate this with Voronoi-biased motion planning algorithms and to investigate our algorithm's applicability and effectiveness for locating vertices on generalized Voronoi graphs.

Acknowledgments

Thanks to Hamid Chitsaz for several helpful conversations. This work was funded in part by NSF Awards 9875304, 0118146, and 0208891.

References

[1] M. Akinc, K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plakue, and L. E. Kavaki. Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In *11th Int. Symp. Robot. Res.*, 2003.

[2] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans. Robot. & Autom.*, 16(4):442–447, Aug 2000.

[3] F. Anton, J. Snoeyink, and C. Gold. An iterative algorithm for the determination of Voronoi vertices in polygonal and non-polygonal domains on the plane and the sphere. In *14th European Workshop on Computational Geometry*, 1998.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

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

[6] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *Hybrid Systems: Computation and Control*, March 2004.

[7] 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.

[8] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. *IEEE Trans. Robot. & Autom.*, 17(2):125–137, 2001.

[9] J. M. Esposito, J. Kim, and V. Kumar. Adaptive RRTs for validating hybrid robotic control systems. In *Proc. Workshop on Algorithmic Foundation of Robotics*, 2004.

[10] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 373–382, 1995.

[11] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Trans. Autom. Control*, 43:540–554, 1998.

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

[13] 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.

[14] 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.

[15] A. M. Ladd and L. E. Kavraki. Fast exploration for robots with dynamics. In *Proc. Workshop on Algorithmic Foundation of Robotics*, 2004.

[16] J.-M. Lien, S.L. Thomas, and N.M. Amato. A general framework for sampling on the medial axis of free space. In *IEEE Int. Conf. Robot. & Autom.*, 2003.

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

[18] S. R. Lindemann, A. Yershova, and S. M. LaValle. Incremental grid sampling strategies in robotics. In *Workshop on the Algorithmic Foundations of Robotics*, 2004.

[19] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[20] J. Matousek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Comp. Geom.*, 10:215–232, 1993.

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

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

[23] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comp. Geom.*, 6:423–434, 1991.

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

[25] M. Strandberg. Augmenting RRT-like planners with local trees. In *IEEE Int. Conf. Robot. & Autom.*, pages 3258–3262, 2004.

[26] J. Vleugels, V. Ferrucci, M. Overmars, and A. Rao. Hunting Voronoi vertices. *Computational Geometry: Theory and Applications*, 6:329–354, 1996.

[27] 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.

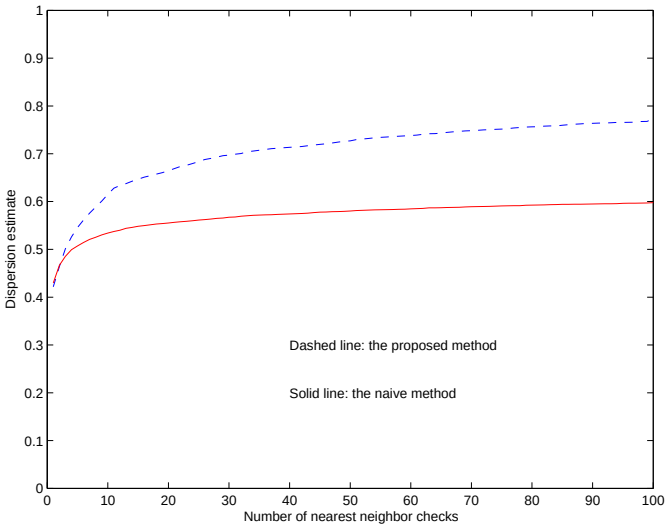

Fig. 3. Dispersion estimation comparison in 10 dimensions.

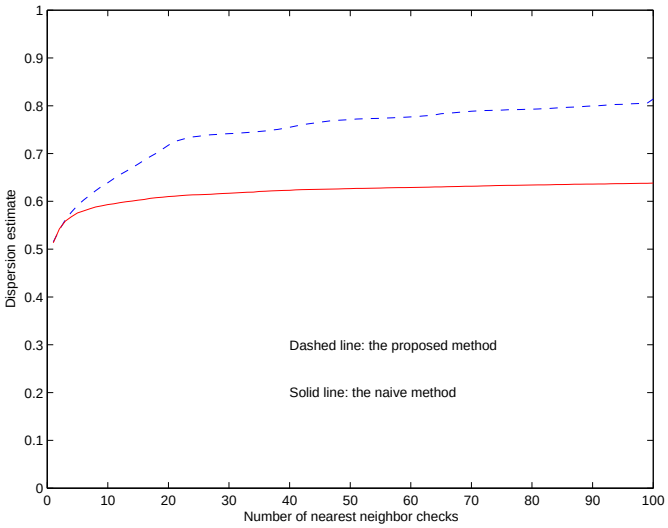

Fig. 4. Dispersion estimation comparison in 20 dimensions.