A Distributed Greedy Heuristic for Computing Voronoi Tessellations With Applications Towards Peer-to-Peer Networks

Brendan Benshoof Andrew Rosen Anu G. Bourgeois Robert W. Harrison
Department of Computer Science, Georgia State University
bbenshoof@cs.gsu.edu rosen@cs.gsu.edu anu@cs.gsu.edu rharrison@cs.gsu.edu

***Abstract*—Computing Voronoi tessellations in an arbitrary number of dimensions is a computationally difficult task. This problem becomes exacerbated in distributed environments, such as Peer-to-Peer networks and Wireless networks, where Voronoi tessellations have useful applications.**

We present our Distributed Greedy Voronoi Heuristic, which approximates Voronoi tessellations in distributed environments. Our heuristic is fast, scalable, works in any geometric space with a distance and midpoint function, and has interesting applications in embedding metrics such as latency in the links of a distributed network.

I. Introduction

Voronoi diagrams [1] have been used in distributed and peer-to-peer (P2P) applications for some time. They have a wide variety of applications. Voronoi diagrams can be used as to manage distributed hash tables [2], or for in coverage detection for wireless networks [3]. Additionally, Massively Multiplayer Online games (MMOs) can use them to distribute game states and events between players at a large scale [4] [5] [6].

Computing the Voronoi tessellation along with its coprime problem, Delaunay Triangulation, is a well analyzed problem. There are many algorithms to efficiently compute a Voronoi tessellation given all the points on a plane, such as Fortune's sweep line algorithm [7]. However, many network applications are distributed and many of the algorithms to compute Voronoi tessellations are unsuited to a distributed environment.

In addition, complications occurs when points are located in spaces with more than two dimensions. Computing the Voronoi tessellation of n points in a space with d dimensions takes $O(n^{\frac{2d-1}{d}})$ time [8]. Distributed computations often have to resort to costly Monte-Carlo calculations [9] in order to handle more than two dimensions.

Rather than exactly solving the Voronoi tessellation, we instead present a fast and accurate heuristic to approximate each of the regions of a Voronoi tessellation This enables fast and efficient formation of P2P networks. A P2P network built using this heuristic would be able to take advantage of it's available fault-tolerant architecture to route along any inaccuracies that arise. Our paper presents the following contributions:

- We present our Distributed Greedy Voronoi Heuristic (DGVH). The DGVH is a fast, distributed, and highly accurate method, whereby nodes calculate their individual regions described by a Voronoi tessellation using the positions of nearby nodes. DGVH can work in an arbitrary number of dimensions and can handle non-euclidean distance metrics. Our heuristic can also handle toroidal spaces. In addition, DGVH can accommodate the calculation of nodes moving their positions and adjust their region accordingly, while still maintaining a high degree of accuracy (Section II). Even where small inaccuracies exist, DVGH will create a fully connected graph.
- We discuss how P2P networks and distributed applications can use DGVH (Section III). In particular, we show how we can use DGVH to build a distributed hash table with embedded minimal latency.
- We present simulations demonstrating DGVH's efficacy in quickly converging to the correct Voronoi tessellation. We simulated our heuristic in networks ranging from size 500 nodes to 10000 nodes. Our simulations show that a distributed network running DGVH accurately determines the region a randomly chosen point falls in 90% of the time within 20 cycles and converges near 100% accuracy by cycle 30.
- We present the related work we have built upon

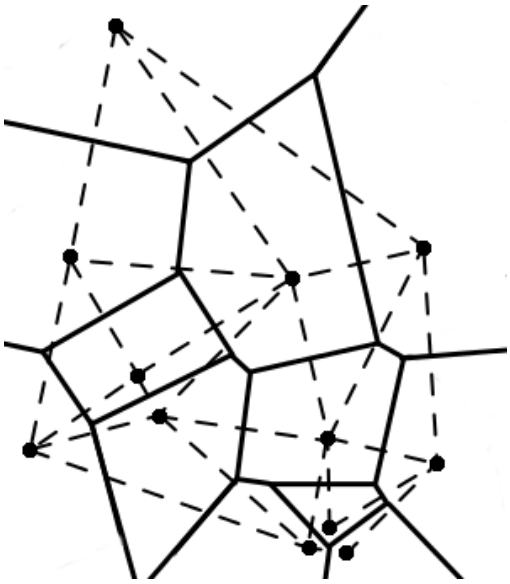


Fig. 1: An example Voronoi diagram for objects on a 2-dimensional space. The black lines correspond to the borders of the Voronoi region, while the dashed lines correspond to the edges of the Delaunay Triangulation.

to create our heuristic and what improvements we made with DGVH (Section V).

II. Distributed Greedy Voronoi Heuristic

A Voronoi tessellation is the partition of a space into cells or regions along a set of objects O, such that all the points in a particular region are closer to one object than any other object. We refer to the region owned by an object as that object's Voronoi region. Objects which are used to create the regions are called Voronoi generators. In network applications that use Voronoi tessellation, nodes in the network act as the Voronoi generators.

The Voronoi tessellation and Delaunay triangulation are dual problems, as an edge between two objects in a Delaunay triangulation exists if and only if those object's Voronoi regions border each other. This means that solving either problem will yield the solution to both. An example Voronoi diagram is shown in Figure 1. For additional information, Aurenhammer [10] provides a formal and extremely thorough description of Voronoi tessellations, as well as their applications.

A. Our Heuristic

The Distributed Greedy Voronoi Heuristic (DGVH) is a fast method for nodes to define their individual Voronoi region (Algorithm 1). This is done by selecting the nearby nodes that would correspond to the points connected to it by a Delaunay triangulation. The rationale for this heuristic is that, in the majority of cases, the midpoint between two nodes falls on the common boundary of their Voronoi regions.

Algorithm 1 Distributed Greedy Voronoi Heuristic

1: Given node n and its list of $candidates$.
2: Given the minimum $table\_size$
3: $short\_peers \leftarrow$ empty set that will contain n's one-hop peers
4: $long\_peers \leftarrow$ empty set that will contain n's two-hop peers
5: Sort $candidates$ in ascending order by each node's distance to n
6: Remove the first member of $candidates$ and add it to $short\_peers$
7: **for all** c in $candidates$ **do**
8: m is the midpoint between n and c
9: **if** Any node in $short\_peers$ is closer to m than n **then**
10: Reject c as a peer
11: **else**
12: Remove c from $candidates$
13: Add c to $short\_peers$
14: **end if**
15: **end for**
16: **while** $|short\_peers| < table\_size$ **and** $|candidates| > 0$ **do**
17: Remove the first entry c from $candidates$
18: Add c to $short\_peers$
19: **end while**
20: Add $candidates$ to the set of $long\_peers$
21: **if** $|long\_peers| > table\_size^2$ **then**
22: $long\_peers \leftarrow$ random subset of $long\_peers$ of size $table\_size^2$
23: **end if**

During each cycle, nodes exchange their peer lists with a current neighbor and then recalculate their neighbors. A node combines their neighbor's peer list with its own to create a list of candidate neighbors. This combined list is sorted from closest to furthest. A new peer list is then created starting with the closest candidate. The node then examines each of the remaining candidates in the sorted list and calculates the midpoint between the node and the candidate. If any of the nodes in the new peer list are closer to the midpoint than the candidate, the candidate is set aside. Otherwise the candidate is added to the new peer list.

DGVH never actually solves for the actual polytopes

that describe a node's Voronoi region. This is unnecessary and prohibitively expensive [9]. Rather, once the heuristic has been run, nodes can determine whether a given point would fall in it's region.

Nodes do this by calculating the distance of the given point to itself and other nodes it knows about. The point falls into a particular node's Voronoi region if it is the node to which it has the shortest distance. This process continues recursively until a node determines that itself to be the closest node to the point. Thus, a node defines its Voronoi region by keeping a list of the peers that bound it.

This heuristic has the benefit of being fast and scalable into any geometric space where a distance function and midpoint can be defined. The distance metric used for this paper is the minimum distance in a multidimensional unit toroidal space. Where $\vec{a}$ and $\vec{b}$ are locations in a d-dimensional unit toroidal space:

$$distance = \sqrt{\sum_{i \in d} (\min(|\vec{a}_i - \vec{b}_i|, 1 - |\vec{a}_i - \vec{b}_i|))^2}$$

Whether or not distance corresponds to actual physical distance or some virtual distance on an overlay depends on the application.

Our heuristic can be overaggressive in removing candidate nodes. For example, if a node is located between two other nodes, such that their midpoint does not fall upon the shared face of their Voronoi regions, then this heuristic will not link the blocked peers. This is demonstrated in Figure 2. Our algorithm handles these cases via our method of peer management (Section II-B).

B. Peer Management

Nodes running the heuristic maintain two peer lists: *Short Peers* and *Long Peers*. This is done to mitigate the error induced by DGVH and provide robustness against churn[1] in a distributed system.

Short Peers are the set of peers DGVH judged to have Voronoi regions adjacent to the node's own. Using a lower bound on the length of *Short Peers* corrects for errors in the approximation as it forces nodes to include peers that would otherwise be omitted. Previous work by Beaumont *et al*. [9] has found a useful lower bound on short peers to be $3d + 1$. Should the number of short peers generated by DGVH be less than the lower bound, the nearest peers not already included in *Short Peers* are added to it, until *Short Peers* is of sufficient size.

[1]The disruption caused to an overlay network by the continuous joining, leaving, and failing of nodes.

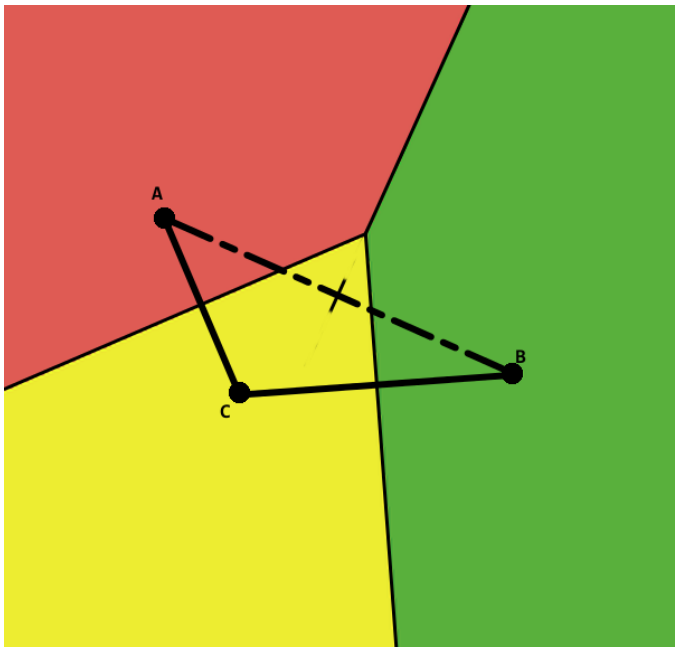


Fig. 2: The edge between A and B is not detected by DGVH, as node C is closer to the midpoint than B is. This is mitigated by peer management polices.

There is no upper bound to the number of short peers a node can have. This means in contrived cases, such as a single node surrounded by other nodes forming a hypersphere, this number can grow quite high. Bern *et al.* [11] found that the expected maximum degree of a vertex in a Delaunay Triangulation is

$$\Theta(\frac{\log n}{\log \log n})$$

where n is the number of nodes in the Delaunay Triangulation. This bound applies to a Delaunay Triangulation in any number of dimensions. Thus, the maximum expected size of *Short Peers* is bounded by $\Theta(\frac{\log n}{\log \log n})$, which is a highly desirable number in many distributed systems [12] [13].

Long Peers is the list of two-hop neighbors of the node. When a node learns about potential neighbors, but are not included in the short peer list, they may be included in the long peer list. *Long Peers* has a maximum size of $(3d+1)^2$, although this size can be tweaked to the user's needs. For example, if *Short Peers* has a minimum size of 8, then *Long Peers* has a maximum of 64 entries. We recommend that members of *Long Peers* are not actively probed during maintenance to minimize the cost of maintenance. A maximum size is necessary, as leaving it unbounded would result in a node eventually keeping track of all the nodes in the network, which would be

counter to the design of a distributed and scalable system.

How nodes learn about peers is up to the application. We experimented using a gossip protocol, whereby a node selects peers from *Short Peers* at random to "gossip" with. When two nodes gossip with each other, they exchange their *Short Peers* with each other. The node combines the lists of short peers[2] and uses DGVH to determine which of these candidates correspond to its neighbors along the Delaunay Triangulation. The candidates determined not to be short peers become long peers. If the resulting number of long peers exceeds the maximum size of *Long Peers*, a random subset of the maximum size is kept.

The formal algorithm for this process is described in Algorithm 2. This maintenance through gossip process is very similar to the gossip protocol used in Beaumont et al.'s RayNet [9].

Algorithm 2 Gossiping

1: Node n initiates the gossip.
2: $neighbor \leftarrow$ random node from $n.short\_peers$
3: $n\_candidates \leftarrow n.short\_peers \cup n.long\_peers \cup neighbor.short\_peers$
4: $neighbor\_candidates \leftarrow neighbor.short\_peers \cup neighbor.long\_peers \cup n.short\_peers$.
5: n and $neighbor$ each run Distributed Greedy Voronoi Heuristic using their respective $candidates$

C. Algorithm Analysis

DVGH is very efficient in both terms of space and time. Suppose a node n is creating its short peer list from k candidates in an overlay network of N nodes. The candidates must be sorted, which takes $O(k \cdot \lg(k))$ operations. Node n must then compute the midpoint between itself and each of the k candidates. Node n then compares distances to the midpoints between itself and all the candidates. This results in a cost of

$$k \cdot \lg(k) + k \text{ midpoints} + k^2 \text{ distances}$$

Since k is bounded by $\Theta(\frac{\log N}{\log \log N})$ [11] (the expected maximum degree of a node), we can translate the above to

$$O(\frac{\log^2 N}{\log^2 \log N})$$

[2]Nodes remove themselves and repetitions from the candidates they receive.

In the vast majority of cases, the number of peers is equal to the minimum size of *Short Peers*. This yields $k = (3d+1)^2 + 3d + 1$ in the expected case, where the lower bound and expected complexities are $\Omega(1)$.

Previous work [9] claims constant time approximation. The reality is that Raynet's leading constant is in the order of thousands. Our algorithm has a greater asymptotic worst case cost, but for all current realistic network sizes it will be more time efficient then RayNet's approximation.

III. Applications

As we previously discussed in Section I, Voronoi tessellation have many applications for distributed systems [3] [4] [5] [6]. We focus our discussion on the two extremes of applications: DHTs, which work with overlay networks, and wireless networks, which need to take literal physical constraints into account.

A. Distributed Hash Tables

Arguably all Distributed Hash Tables (DHTs) are built on the concept of Voronoi tessellation. In all DHTs, a node is responsible for all points in the overlay to which it is the "closest" node. Nodes are assigned a key as their location in some keyspace, based on the hash of certain attributes. Normally, this is just the hash of the IP address (and possibly the port) of the node [12] [13] [14] [15], but other metrics such as geographic location can be used as well [16].

These DHTs have carefully chosen metric spaces such that these regions are very simple to calculate. For example, Chord [12] and similar ring-based DHTs [17] utilize a unidirectional, one-dimensional ring as their metric space, such that the region for which a node is responsible is the region between itself and its predecessor.

Using a Voronoi tessellation in a DHT generalizes this design. Nodes are Voronoi generators at a position based on their hashed keys. These nodes are responsible for any key that falls within its generated Voronoi region.

Messages get routed along links to neighboring nodes. This would take $O(n)$ hops in one dimension. In multiple dimensions, our routing algorithm (Algorithm 3) is extremely similar to the one used in Ratnasamy et al.'s Content Addressable Network (CAN) [14], which would be $O(n^{\frac{1}{d}})$ hops.

DGVH can be used in a DHT to quickly and scalably construct both the Voronoi tessellation and links to peers. In addition, gossip-based peer management policies are extremely efficient in proactively handling node joins

Algorithm 3 Lookup in a Voronoi-based DHT

1: Given node n
2: Given m is a message addressed for loc
3: $potential\_dests \leftarrow n \cup n.short\_peers \cup n.long\_peers$
4: $c \leftarrow$ node in $potential\_dests$ with shortest distance to loc
5: **if** $c == n$ **then**
6: **return** n
7: **else**
8: **return** $c.lookup(loc)$
9: **end if**

and failures. The act of joining informs other nodes of the joiner's existence. This can be done by the joiner contacting each peer in the peer lists of the node previously responsible for the joiner's location.

Node failures can be handled without much effort. When a node attempts to route to or gossip with a node and discovers it no longer exists, it should remove the node from its peer lists and inform all of the nodes it knows to do the same. Since routing is how a node would make a decision on whether a point belongs to a particular Voronoi region, failed nodes don't have any impact on the network's accuracy.

Because the algorithm is defined in terms of midpoint and distance functions, it is not bound to any particular topology or metric space. Our heuristic can be used to create a DHT that uses any arbitrary coordinate system which defines a midpoint and distance definition.

For example, we could use latency as one of these metrics by using it to approximate node locations in the network. This would allow messages to be routed along minimum latency paths, rather than along minimal hop paths. We intend to model this using a distributed spring model.

B. Wireless Coverage

Cărbunar et al. [3] demonstrated how Voronoi tessellations could be used to solve the *coverage-boundary* problem in wireless ad-hoc networks. The coverage-boundary problem asks which nodes are on the physical edge of the network. This knowledge provides useful information to networks. For example, ad-hoc networks operating in no infrastructure or have been set up temporarily can use this knowledge to define the reach of the network's coverage.

The authors showed how a Voronoi tessilation using the nodes as Voronoi generators could solve the coverage-boundary problem. They proved that the node's Voronoi region was not completely covered by the node's sensing radius if and only if the node was on the network's boundary [3].

Chen et al. [18] extended this property for sleep scheduling in wireless sensor networks. A node is allowed to go to sleep and conserve power so long as the area it is detecting can be covered by other nodes. Using Voronoi tessellations, a node knows it can conserve power and not affect the network if and only if it is not on the boundary of the network and its Voronoi vertices[3] are covered by other nodes.

Cărbunar et al.'s algorithm relies on a centralized computation for Voronoi tessellation, as the distributed computation they examined only relied on forming the Delaunay triangulation between nodes within a certain radius, nor could it handle moving nodes. DGVH is not limited to handling nodes within a specified radius, since the peer management spreads information about nodes by gossiping. In addition, so long as a node moving to a new location is treated as an entirely new node by the rest of the network, DGVH can handle moving nodes.[4]

IV. Experiments

We implemented two sets of experiments for DGVH. The first compares the Voronoi tessellations created by DGVH to actual am Voronoi tessellation. Our second set of experiments demonstrates that any errors computer by DGVH are negligible when building a distributed and fault-tolerant systems.

A. Experiment 1: Voronoi Accuracy

B. Experiment 2: P2P Convergence and Routing

Our second set of experiments examines how DGVH could be used to create a DHT and how well it would perform in this task. Our simulation demonstrates how DGVH can be used to create a stable overlay from a chaotic starting topology after a sufficient number of gossip cycles. We do this by showing that the rate of successful lookups approaches 1.0. We compare these results to RayNet [9], which proposed that a random k-connected graph would be a good, challenging starting

[3]Voronoi vertices are the points at which the edges of 3 or more Voronoi cells converge.

[4]A specific node and a specific location must be bound together into a single identity. This means when a node tries to route to a node that has moved using the node's previous location, it should fail, as though that specific node no longer exists. Failure to do this would cause a node to incorrectly determines what falls within its Voronoi region.

configuration for demonstrating convergence of a DHT to a stable network topology.

During the first two cycles of the simulation, each node bootstraps its short peer list by appending 10 nodes, selected uniformly at random from the entire network. In each cycle, the nodes gossip (Algorithm 2) and run DGVH using the new information. We then calculate the hit rate of successful lookups by simulating 2000 lookups from random nodes to random locations, as described in Algorithm 4. A lookup is successful when the network correctly determines which Voronoi region contains a randomly selected point.

Our experimental variables for this simulation were the number of nodes in the DGVH generated overlay and the number of dimensions. We tested network sizes of 500, 1000, 2000, 5000, and 10000 nodes each in 2, 3, 4, and 5 dimensions. The hit rate at each cycle is $\frac{hits}{2000}$, where $hits$ are the number of successful lookups.

Algorithm 4 Routing Simulation Sample

1: $start \leftarrow$ random node
2: $dest \leftarrow$ random set of coordinates
3: $ans \leftarrow$ node closest to $dest$
4: **if** $ans == start.lookup(dest)$ **then**
5: increment $hits$
6: **end if**

Our results are shown in Figure **??** for each dimension. Our graphs show that the created overlay quickly constructs itself from a random configuration and that our hit rate reached 90% by cycle 20, regardless of dimension. Lookups consistently approached a hit rate of 100% by cycle 30. In comparison, RayNet's routing converged to a perfect hit rate at around cycle 30 to 35 [9]. As the network size and number of dimensions each increase, convergence slows, but not to a significant degree.

V. Related Work

While there has been previous work on applying Voronoi regions to DHTs and peer-to-peer (P2P) applications, we have found no prior work on how to perform embedding of an inter-node latency graph.

Backhaus et al.'s VAST [6] is a Voronoi-based P2P protocol designed for handling event messages in a massively multiplayer online video game. Each node finds its neighbors by constructing a Voronoi diagram using Fortune's sweepline algorithm [7]. VAST demonstrated that Voronoi diagrams could be used as the backbone to large-scale applications, although their work focused specifically on using 2-dimensional Voronoi diagrams.

The two DHT protocols developed by Beumont et al., VoroNet [19] and RayNet [9] are two-dimension DHTs that we intend to extend using our technique. VoroNet is based off Kleinberg's small world model [20] and achieves polylogarithmic lookup time. Each node in VoroNet solves its Voronoi region to determine its neighbors and also maintains a link to a randomly chosen distant node. VoroNet focused specifically on the two-dimensional Voronoi computations and the techniques used would be too expensive in higher dimensions and were not resilient to churn [9].

RayNet [9] was based on the work done on VoroNet and used a heuristic to calculate Voronoi tessilations. Like our DGVH, RayNet's heuristic does not solve for Voronoi regions, as that is prohibitively expensive. RayNet uses a Monte-Carlo method to approximate the volume of a node's Voronoi region in constant time. While effective at estimating the Voronoi region, the volume-based Monte-Carlo approximation is expensive and requires multiple samples. This gives the runtime of RayNet's heuristic an enormous leading constant. RayNet does mention the idea of mapping attributes to each axis, but how this can be exploited is left as future work.

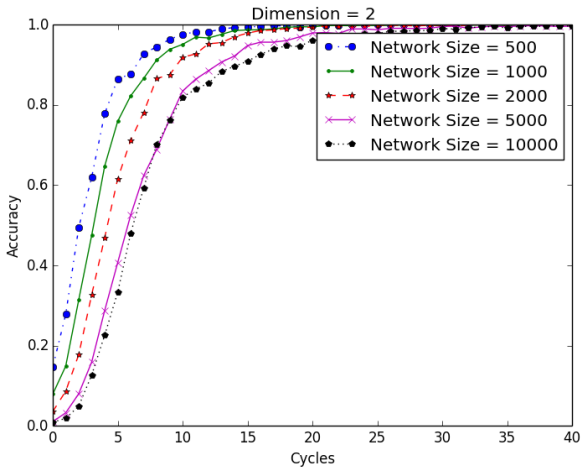
VI. Conclusion and Future Work

Voronoi tessellations have a wide potential for applications in ad-hoc networks, massively multiplayer games, P2P, and distributed networks. However, centralized algorithms for Voronoi tessellation and Delaunay triangulation are not applicable to decentralized systems. In addition, solving Voronoi tessellations in more than 2 dimensions is computationally expensive.

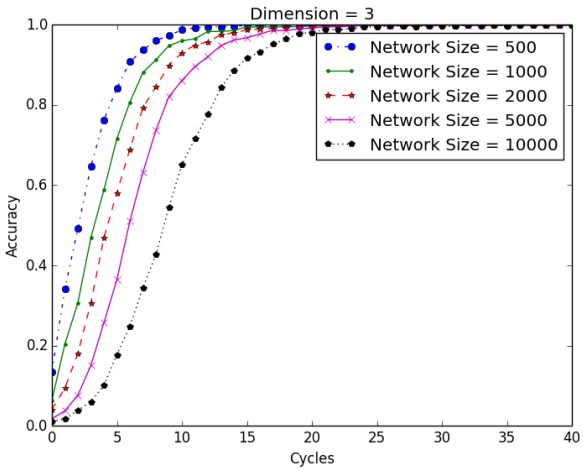
We created a distributed heuristic for Voronoi tessellations in an arbitrary number of dimensions. Our heuristic is fast and scalable, with a expected memory cost of $(3d+1)^2 + 3d + 1$ and expected maximum runtime of $\mathrm{O}(\frac{\log^2 N}{\log^2 \log N})$.

We ran two sets of experiments to demonstrate DGVH's effectiveness. Our first set of experiments demonstrated that our heuristic is reasonably accurate and our second set demonstrates that reasonably accurate is sufficient to build a P2P network which can route accurately.

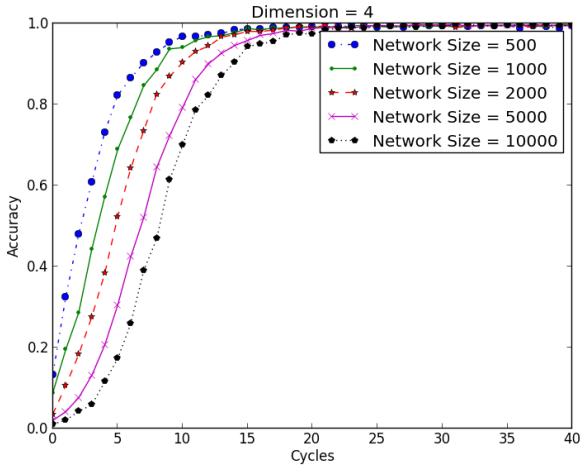
Our next step is to create a formal protocol and implementation for a Voronoi tessellation-based distributed hash table using DGVH. We can use this DHT to choose certain metrics we want to measure, such as latency, or trust, and embed that information as part of a node's



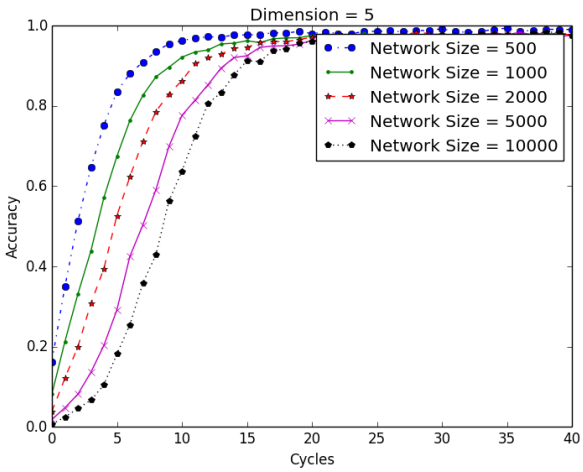
(a) This plot shows the accuracy rate of lookups on a 2-dimensional network as it self-organizes.



(b) This plot shows the accuracy rate of lookups on a 3-dimensional network as it self-organizes.



(c) This plot shows the accuracy rate of lookups on a 4-dimensional network as it self-organizes.



(d) This plot shows the accuracy rate of lookups on a 5-dimensional network as it self-organizes.

Fig. 3: These figures show that, starting from a randomized network, DGVH forms a stable and consistent network topology. The Y axis shows the success rate of lookups and the X axis show the number of gossips that have occurred. Each point shows the fraction of 2000 lookups that successfully found the correct destination.

identity. By creating an appropriate distance measurement, we can route along some path that minimizes or maximizes the desired metric. Rather than create an overlay that minimizes hops, we can have our overlay minimize latency, which is the actual goal of most routing algorithms.

References

[1] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, pp. 345–405, Sept. 1991.

[2] W. Wang, G. Yang, N. Xiong, X. He, and W. Guo, "A general p2p scheme for constructing large-scale virtual environments," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pp. 1648–1655, May 2014.

[3] B. Carbunar, A. Grama, and J. Vitek, "Distributed and dynamic voronoi overlays for coverage detection and distributed hash tables in ad-hoc networks," in *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on*, pp. 549–556, IEEE, 2004.

[4] S.-Y. Hu and G.-M. Liao, "Scalable peer-to-peer networked virtual environment," 2004.

[5] S.-Y. Hu, S.-C. Chang, and J.-R. Jiang, "Voronoi state management for peer-to-peer massively multiplayer online games," in *Consumer Communications and Networking Conference, 2008.*

CCNC 2008. 5th IEEE, pp. 1134–1138, IEEE, 2008.

[6] H. Backhaus and S. Krause, "Voronoi-based adaptive scalable transfer revisited: Gain and loss of a voronoi-based peer-to-peer approach for mmog," in *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '07, (New York, NY, USA), pp. 49–54, ACM, 2007.

[7] S. Fortune, "A sweepline algorithm for voronoi diagrams," *Algorithmica*, vol. 2, no. 1-4, pp. 153–174, 1987.

[8] D. F. Watson, "Computing the n-dimensional delaunay tessellation with application to voronoi polytopes," *The computer journal*, vol. 24, no. 2, pp. 167–172, 1981.

[9] O. Beaumont, A.-M. Kermarrec, and É. Rivière, "Peer to peer multidimensional overlays: Approximating complex structures," in *Principles of Distributed Systems*, pp. 315–328, Springer, 2007.

[10] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[11] M. Bern, D. Eppstein, and F. Yao, "The expected extremes in a delaunay triangulation," *International Journal of Computational Geometry & Applications*, vol. 1, no. 01, pp. 79–91, 1991.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCO*, no. 4, pp. 149–160, ACM, 2001.

[13] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*, pp. 53–65, Springer, 2002.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," 2001.

[15] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*, pp. 329–350, Springer, 2001.

[16] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "Ght: a geographic hash table for data-centric storage," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pp. 78–87, ACM, 2002.

[17] G. S. Manku, M. Bawa, P. Raghavan, *et al.*, "Symphony: Distributed hashing in a small world.," in *USENIX Symposium on Internet Technologies and Systems*, p. 10, 2003.

[18] X. Chen, M. R. Lyu, and P. Guo, "Voronoi-based sleeping configuration in wireless sensor networks with location error," in *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, pp. 1459–1464, IEEE, 2008.

[19] O. Beaumont, A.-M. Kermarrec, L. Marchal, and E. Rivière, "Voronet: A scalable object network based on voronoi tessellations," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–10, IEEE, 2007.

[20] J. M. Kleinberg, "Navigation in a small world," *Nature*, vol. 406, no. 6798, pp. 845–845, 2000.