The Largest Empty Circle Problem

Megan Schuster
megan@cs.swarthmore.edu

Abstract

The largest empty circle (LEC) problem is defined on a set P and consists of finding the largest circle that contains no points in P and is also centered inside the convex hull of P. The LEC is always centered at either a vertex on the Voronoi diagram for P or on an intersection between a Voronoi edge and the convex hull of P. Thus, finding the LEC consists of constructing a Voronoi diagram and convex hull for P, then searching the Voronoi vertices and intersections between Voronoi edges and convex hull edges to see where the LEC lies. This paper presents a simple $O(n[h+\log n])$ solution to the largest empty circle problem. Though previous work on this problem has found $O(n \log n)$ solutions, we find that for data sets which are somewhat normally distributed, h is small and our simple algorithm performs well.

1 Introduction

The Largest Empty Circle (LEC) problem is defined on a set of points P and consists of locating the largest circle that contains no points of P and is centered inside the convex hull of P. Less formally, this problem finds the point q bounded by the convex hull which maximizes the distance to its nearest neighbor $p \in P$; this point is the center of the largest empty circle. We constrain q to lie within the convex hull of P because otherwise we would simply choose the point at infinity as the center of the LEC.

This problem is sometimes referred to as the Toxic Waste Dump problem, because given the coordinates for a set of cities, the LEC problem would allow you to find the best site for a toxic waste dump by finding the location which is maximally far from every city. This might also be useful for planning locations for new stores. For example, imagine you would like to build a new McDonald's in a metropolitan ares that already has several dozen McDonald's stores. By computing the LEC on the set of existing McDonald's restaurants, you could select a site for a new store which is maximally far from all existing stores to minimize competition with other McDonald's restaurants and situate yourself near people who previously did not have a McDonald's nearby.

The Voronoi diagram for the set P is a useful tool for solving the LEC problem. The Voronoi diagram is a partition of the plane into convex faces such that given a set of points P, each face (hereafter, Voronoi cell) contains exactly one point $p \in P$ and all points in the plane which are closer to p than to any other point in P. Points lying on the edges between Voronoi cells are equidistant between the two points contained in the cells lying to either side of the edge. Given these properties, it seems intuitive that the largest empty circle should be centered on a Voronoi vertex. Since the edges represent all points which are equidistant to the two points they divide, it follows that a vertex, which is simply an intersection between multiple Voronoi edges, would maximizes the minimum distance to all nearby points. Any point that is not on a Voronoi vertex must be closer to one point than any other and as such any empty circle we can draw around it will not be of maximal size.

Based on the above principles, it seems that we might simply be able to draw empty circles around all Voronoi vertices and see which is the largest. However, since we constrain our circle to be centered inside the convex hull of P, we must be slightly more careful in our search for the LEC. First, we must consider only Voronoi vertices which lie inside the convex hull of P. Second, we must consider what happens on the edges of the convex

hull itself. At points where a Voronoi edge intersects a convex hull edge, the distances between each of the two nearest points is maximized, since we are on a Voronoi edge. Such points must also be considered as candidates for the center of the LEC.

2 Related Work

The earliest solution to the LEC problem was presented by Shamos in his Ph.D. thesis (1978). Shamos presented an algorithm that, given the Voronoi diagram and the convex hull, could find the largest empty circle in $O(n)$ time. Unfortunately, this algorithm was based on the assumption that every convex hull edge is intersected by at most two Voronoi edges, which is not always true, as later shown by Toussaint (1983). Because the original Shamos algorithm incorrectly assumes a maximum of two Voronoi edge intersections at each convex hull edge, it can miss intersections at edges with more than two intersections with Voronoi edges and, as such, can fail to recognize the true LEC.

Toussaint (1983) went on to present an algorithm which correctly finds the LEC in all cases. However, the Toussaint algorithm requires $O(n \log n)$ running time when given the convex hull and Voronoi diagram. The algorithm first computes the largest empty circle about each Voronoi vertex which lies in the interior of the convex hull. This step requires $O(n \log h)$ operations; there are $O(n)$ Voronoi vertices for which we must do an $O(\log h)$ point location step to check for interiority to the convex hull (where h is the number of convex hull edges), and computing the largest empty circle about a point can be done in constant time. Once all interior points have been considered, the algorithm computes all intersections between Voronoi edges and convex hull edges. Toussaint uses an $O(\log n)$ algorithm taken from Chazelle (1980) to find the intersections between a line segment and a convex n-gon. Since there are $O(n)$ Voronoi edges (de Berg et al., 2000), this step requires $O(n \log h)$ time overall. By checking all points interior to the convex hull and all intersection points between the Voronoi diagram and the convex hull, all possible sites for the center of the LEC have been considered. All that remains is to report the point about which the largest circle was drawn. Toussaint thus finds the LEC in $O(n \log n)$ time.

Later, Preparata and Shamos (1985) offer an improved version of Shamos's original methods (1978) which no longer relies on the assumption that each convex hull edge is intersected by at most two Voronoi edges. Preparata and Shamos describe an $O(n)$ marching method for finding all intersections between the Voronoi edges and the convex hull, which is an improvement on Toussaint's $O(n \log h)$ method. Preparata and Shamos do not go into detail about how to check Voronoi vertices on the interior of the convex hull to see if they might be the center of the largest empty circle. We can only assume that they, like Toussaint, also require an $O(n \log h)$ technique for checking interior points. This solution is then slightly faster than Toussaint's, thanks to the $O(n)$ intersection location step.

Still, all of the LEC-finding algorithms discussed above require use of the Voronoi diagram and the convex hull. Both of these structures require $O(n \log n)$ steps to compute. Construction of these structures dominates the computation time when finding the LEC, so while there may be some quibbling about the fastest methods for finding intersections between Voronoi diagrams and convex hulls, overall the solution to the LEC problem is at best $O(n \log n)$ regardless of the steps required to actually find the largest empty circle.

In this paper, we present a complete method for finding the LEC, borrowing our approach to the problem from the previous work of Toussaint. The algorithm presented includes a computation of the Voronoi diagram and the convex hull and requires $O(n[h + \log n])$ running time.

3 Methods

3.1 Computing the Voronoi Diagram and Convex Hull

Before we can find the largest empty circle for a set of points P, we must first construct the Voronoi diagram ($Vor(P)$) and convex hull ($CH(P)$) for the set of points. Here, $Vor(P)$ is computed by first finding the Delaunay triangulation, $DT(P)$, which is the dual of $Vor(P)$. Common algorithms for computing $Vor(P)$ involve an $O(n \log n)$ plane sweep and are not dynamically updateable. However, de Berg et al. (2000) describe an incremen-

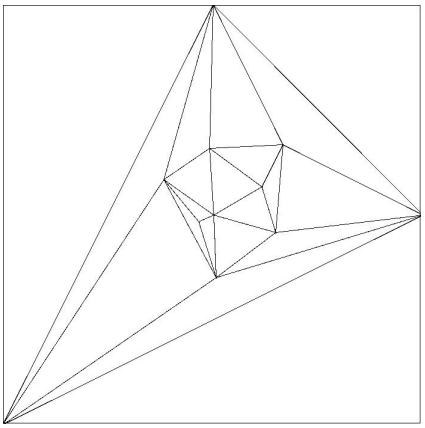

Figure 1: A simple example of the Delaunay triangulation computed on a small set of points.

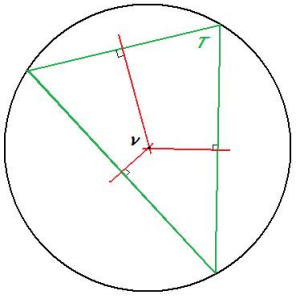

Figure 2: The circumcircle for a triangle T, whose center v is found by perpendicular bisector construction. If T is a triangle in a Delaunay triangulation, the v is the Voronoi vertex obtained when dualizing the Delaunay triangulation to the Voronoi diagram.

tal algorithm for computing $DT(P)$. Because we would like to support dynamic updates to $Vor(P)$, we choose to compute $Vor(P)$ by dualization of a dynamically updateable implementation of $DT(P)$.

3.1.1 The Delaunay Triangulation

de Berg et. al (2000) describe an incremental algorithm for computing Delaunay triangulations, which we employ here to compute $DT(P)$. The Delaunay triangulation is a special type of triangulation which maximizes the minimum angle found in any triangle in the triangulation. An example is shown in Figure 1.

We begin with a very large bounding triangle and add points from P to it one at a time. When adding a point p, we check the current triangulation and locate the triangle T which contains p. We then draw edges between p and each of the vertices of T to re-triangulate the set P. In doing so, however, we may have introduced new triangles with small angles such that we no longer have a Delaunay triangulation. Thus, as we re-triangulate with every added point, we must check the edges of new triangles introduced to see whether they form any small angles. We flip such edges as needed to the opposite corners of the quadrilateral which contains them, recursing on triangles in the neighborhood of newly flipped edges until we have ensured again that the smallest angle in the triangulation is as large as possible.

We represent triangles as nodes in a directed acyclic graph. Whenever a triangle is divided by insertion of a new point or changed due to edge-flipping, it sets pointers to the new triangles which result from these changes. Each triangle node also maintains a pointer for each of its three edges indicating which other triangle neighbors it along that edge, with care being taken to update these neighbor pointers as new triangles are introduced and edge flips are performed. These neighbor pointers are crucial to our ability to dualize $DT(P)$ to $Vor(P)$.

For full details on the computation of Delaunay triangulations, see see de Berg et al. (2000). Once we have a Delaunay triangulation in place, we can dualize it to give the desired Voronoi diagram.

3.1.2 The Voronoi Diagram

Once $DT(P)$ has been computed, it is fairly straightforward to dualize it to $Vor(P)$. The duality between the Voronoi diagram and the Delaunay triangulation is such that every triangle in $DT(P)$ corresponds to a vertex in $Vor(P)$. Triangles which are neighbors in $DT(P)$ have their vertices connected by an edge in the $Vor(P)$ dual space (de Berg et al., 2000).

The coordinates of the Voronoi vertex which correspond to a triangle $T \in DT(P)$ can be found by locating the center of the circle which circumscribes T (Okabe et al., 2000). To compute the circumcircle, we use the perpendicular bisector construction as in Figure 2; the circumcircle for a triangle is centered at the point at which the perpendicular bisectors for the triangle's edges all intersect. To compute $Vor(P)$ from $DT(P)$, then, we first iterate over all triangles in $DT(P)$, computing the circumcircle to find the dual vertex in $Vor(P)$.

Once all Voronoi vertices have been found, we

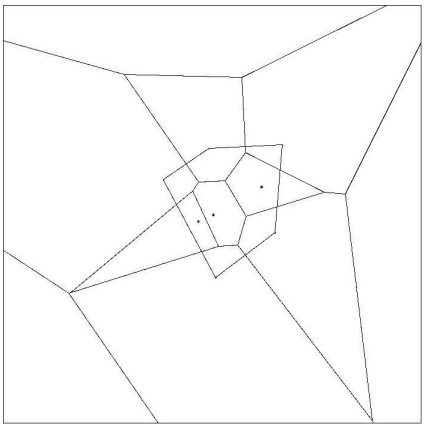

Figure 3: A simple example of the Voronoi diagram and convex hull computed on the set of points shown in Figure 1.

connect them to one another by again iterating over the triangles of $DT(P)$. Here, we make use of the neighbor pointers stored in our Delaunay structure. For each triangle $T \in DT(P)$ and its corresponding point $p \in Vor(P)$, we retrieve all of T's neighbors, T'. For each $t \in T'$, we take its corresponding point $p' \in Vor(P)$ and draw an edge between p and p'. The resulting set of edges describes exactly the Voronoi planar partition.

The dualization from $DT(P)$ to $Vor(P)$ described here can be computed in $O(n)$ time. We must iterate over all triangles in $DT(P)$, of which there are $O(n)$ (de Berg et al., 2000). For each triangle, we must compute the circumcircle to locate the corresponding Voronoi vertex and check three neighbor pointers to draw the appropriate Voronoi edges. These two operations can be done in constant time, giving an overall $O(n)$ runtime for the dualization step.

3.1.3 The Convex Hull

Finally, we must compute $CH(P)$ before we can go on to find the LEC. Here, we use the simple $O(nh)$ Jarvis march algorithm to compute the convex hull. The interested reader should refer to de Berg et al. (2000) for further details on this algorithm.

At this point, we have found the Voronoi diagram and convex hull for our data set (Figure 3). With all necessary data structures in place, we now proceed to find the largest empty circle.

3.2 Finding the Largest Empty Circle

To find the largest empty circle, we first locate all potential centers for that circle, which involves identifying all Voronoi vertices which are interior to $CH(P)$ and finding all intersections between Voronoi edges and convex hull edges. Once all candidate centers have been located, we draw the largest possible empty circle around each and report which was the largest of all.

3.2.1 Checking Interior Voronoi Vertices

We use a naive approach for finding all Voronoi vertices which are interior to $CH(P)$. For each Voronoi vertex, we march counter-clockwise around $CH(P)$, checking whether the vertex lies to the left of the edge. If the vertex lies to the left of all edges in $CH(P)$, we know that it is interior and add it to the list of candidate LEC centers. This requires $O(nh)$ steps, since we must check all h convex hull edges for all $O(n)$ Voronoi vertices.

3.2.2 Finding Convex Hull and Voronoi Edge Intersections

We employ another naive approach for finding all intersections between Voronoi edges and convex hull edges. For every Voronoi edge, we check both endpoints for interiority to $CH(P)$. If one is interior and one is exterior, we know that this edge must intersect $CH(P)$ at some point. We then iterate over all convex hull edges and check for intersection with the Voronoi edge in question. By repeating this process for every Voronoi edge, we are guaranteed to find all intersections between $Vor(P)$ and $CH(P)$.

This step requires $O(nh)$ runtime. We must check all $O(n)$ Voronoi edges, and for any which intersect the convex hull, we must iterate over all h convex hull edges to find the intersection point.

3.2.3 Locating the Largest Empty Circle

Now that we have found all possible points at which the LEC can be centered (as in Figure 4), we have to decide which of these candidates is the actual center of the LEC. To find the largest empty circle that can be drawn around any given candidate center, we exploit the duality between $DT(P)$ and $Vor(P)$. Candidate centers are vertices in Vor space; however, when drawing a circle around a candidate point we want to ensure that this circle does

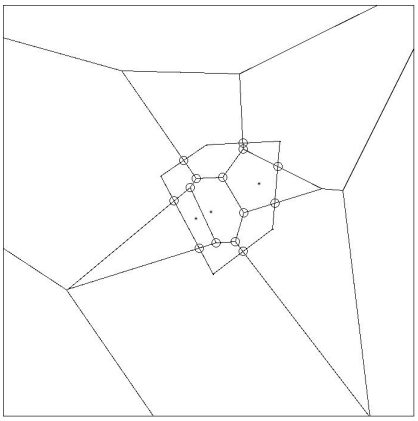

Figure 4: Candidate centers for the largest empty circle for the set of points shown in Figures 3 are outlined here. All candidate centers lie on Voronoi vertices or on intersections between Voronoi and convex hull edges.

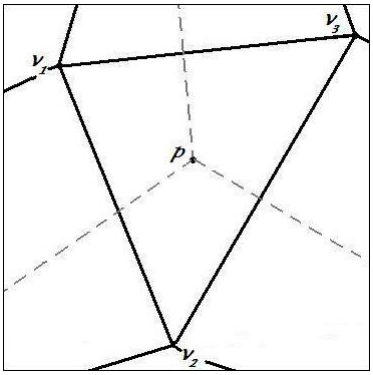

Figure 5: Solid, black lines indicate edges in $DT(P)$. Dashed, grey lines indicate edges in $Vor(P)$. The point p is a Voronoi vertex and is thus a candidate LEC center. Here, p's three nearest neighbors are the three points contained in the Voronoi cells adjacent to p, which are the vertices of p's dual triangle in DT space, v_1, v_2, and v_3.

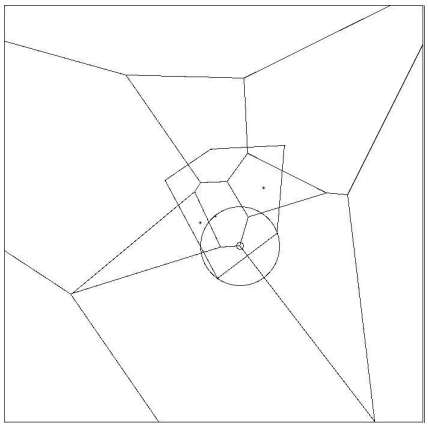

Figure 6: The LEC for our simple set of points.

not contain any points in P, which are vertices in DT space.

Any Voronoi vertex is the intersection of at most three Voronoi edges. This follows directly from the dual relationship between $Vor(P)$ and $CH(P)$; each Voronoi vertex corresponds to one Delaunay triangle and is connected to the vertices which correspond to that triangle's neighbors, of which there are exactly three (except near the edges of the space, where there may be only two neighbors). Since each Voronoi vertex p is incident to at most three Voronoi cells, the three points in P closest to p are those three points which lie in the Voronoi cells to which p is adjacent. Those three points are the vertices of p's dual triangle in $DT(P)$ (see Figure 5). Thus, by dualizing $p \in Vor(P)$ back to $T \in DT(P)$, we can find p's three nearest neighbors. We then choose the closest neighbor and draw a circle about p whose radius equals the distance between that neighbor and p.

We accomplish the dualization of a Voronoi vertex to a Delaunay triangle very simply by building a dictionary of Voronoi vertex-Delaunay triangle pairs as we are doing our initial construction of $Vor(P)$ from $DT(P)$. We then look up candidate vertices in this dictionary to find their nearest neighbors and draw empty circles about them. By drawing empty circles around each candidate vertex, we can locate and report the LEC.

This final step of the algorithm requires $O(n)$ time. Finding the closest point P to any candidate center can be done in constant time by simple dictionary lookup, and we must repeat this process for

Computational Step	Runtime
Compute *DT(P)*	*O(n log n)*
Dualize *DT(P)* to *Vor(P)*	*O(n)*
Compute *CH(P)*	*O(nh)*
Locate Interior Voronoi Vertices	*O(nh)*
Locate *Vor(P)/CH(P)* Edge Intersections	*O(nh)*
Locate LEC from Amongst Candidate Points	*O(n)*

Figure 7: A summary of the steps involved in computing the LEC.

all $O(n)$ candidate centers.

At this point, we need only to report the largest empty circle we have found, as in Figure 6. Figure 7 provides a summary of all steps taken to compute the LEC and their associated runtimes. The overall runtime of our algorithm is thus $O(n[h + \log n])$.

4 Results

We ran the algorithm described above on a set of points corresponding to the latitude and longitude coordinates of all U.S. cities in the 48 contiguous states of population 100,000 or greater (there are 251 such cities). We found the LEC to be centered at -108.2659° latitude, 46.7316° longitude, near Winnette, MT. $Vor(P)$ and $CH(P)$ for this data set are displayed in Figure 8; the resulting LEC is displayed in Figure 9.

The algorithm runs on this dataset of U.S. cities in less than four seconds. In order to analyze the algorithm's overall performance and the performance of intermediate steps within the algorithm, we ran it on a number of data sets ranging in size from 100 points to 10,000 points and recorded the amount of processor time required to compute each step of the algorithm. For these timed tests, we used randomly generated, normally distributed sets of points. The results of these tests are shown in Figure 10.

5 Discussion

Our algorithm was tested on and successfully computed the LEC for the U.S. cities data set as well as randomly generated, normally distributed data sets of up to 10,000 points. It is clear from Figure 10 that the time required to compute the LEC is dominated by the computation of $DT(P)$; all other steps of our algorithm proceed quickly in comparison. For the 10,000 point data set, for example, it took about 33 minutes to compute $DT(P)$. The next slowest step of the algorithm was locating the intersections between $Vor(P)$ and $CH(P)$, which took only 26 seconds.

As mentioned earlier, we used naive approaches to several of the intermediate steps of this algorithm, including the simple Jarvis march for convex hull computation, the point location step for finding all Voronoi vertices interior to the convex hull, and the step for identifying intersections between Voronoi and convex hull edges. Each of these naive steps was $O(nh)$, and for each of these steps we might have used a more sophisticated algorithm in hopes of achieving better runtime for the overall LEC algorithm. For the convex hull, we might have chosen from a variety of $O(n \log n)$ algorithms (see Preparata and Hong (1985), de Berg et al. (2000), for examples). This only improves the speed of computing the convex hull if the number of convex hull edges is somewhat large. For the identification of interior Voronoi vertices, Toussaint (1983) describes an $O(n \log h)$ technique, which is a guaranteed improvement over the $O(nh)$ technique we use, regardless of the distribution of our set of points. For the Voronoi/convex hull intersection location step, Chazelle (1980) describes an $O(n \log h)$ technique for computing the intersections between a set of segments and a convex polygon, and better still, Preparata and Shamos (1985) describe an $O(n)$ march around all Voronoi cells that discovers all intersections with the convex hull.

While the naive approaches to these intermediate steps used in our algorithm could be improved by implementing any of the known faster algorithms mentioned above, the results of Figure 10 suggest that this would provide very little improvement to the running time of our overall LEC algorithm. It is clear that the $O(n \log n)$ computation of $DT(P)$ dominates the computation time for finding the LEC, and thus minor speed-ups to intermediate steps would be of limited value.

It should be noted that the data sets on which we tested our algorithm's running time were all approximately normally distributed. As such, h, the number of convex hull edges, was quite small compared to n, the number of data points (see, for example, Figure 8). Considering the possible applications of the LEC problem, such as toxic waste dump site selection or business location planning, we expect that the data

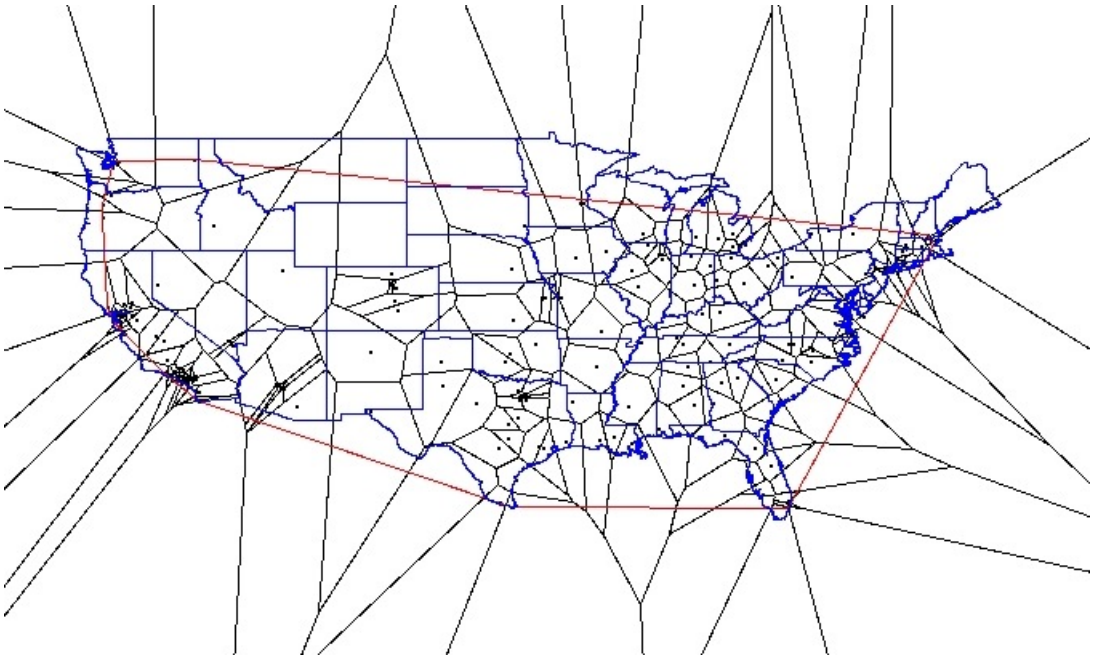

Figure 8: The Voronoi diagram and convex hull for the data set of all US cities of population 100,000 or greater.

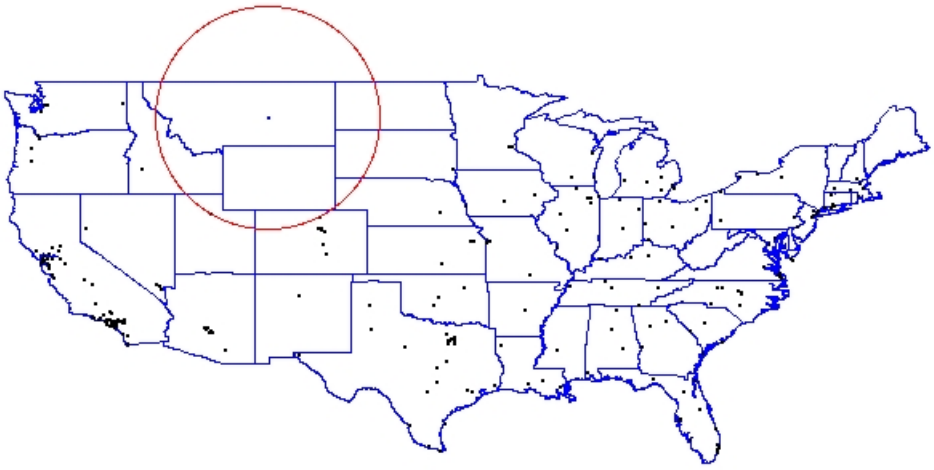

Figure 9: The largest circle which contains no U.S. cities of population 100,000 or greater and is centered within the convex hull of these cities. The center lies at -108.2659° latitude, 46.7316° longitude, near Winnette, MT.

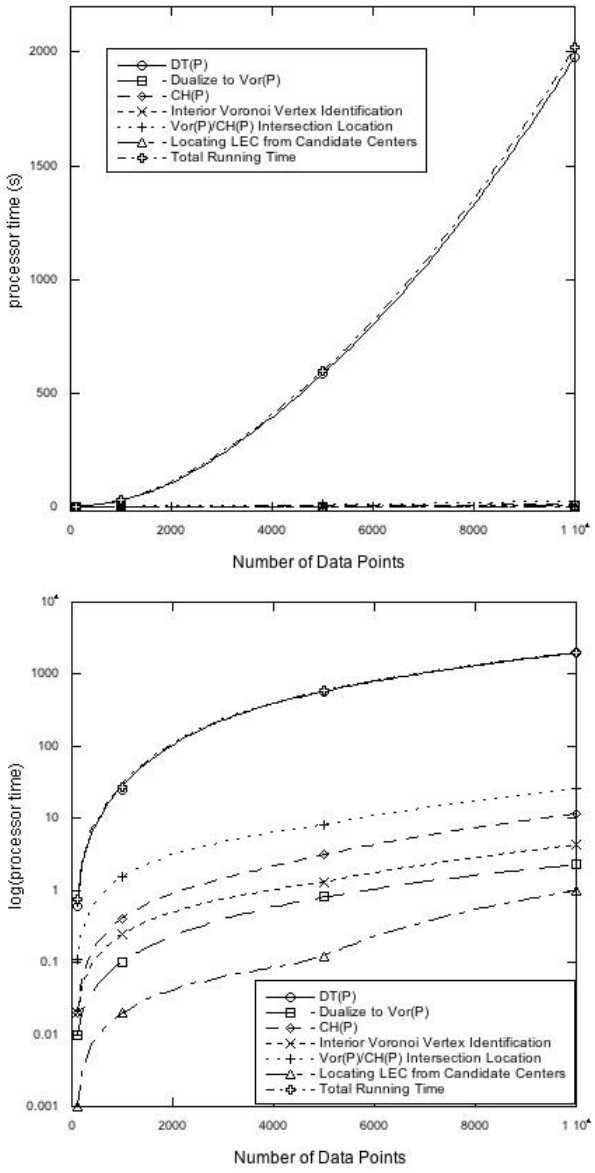

Figure 10: At top, a plot of processor time versus number of data points. The time required for the $DT(P)$ computation is barely distinguishable from the total running time of the entire algorithm. All other algorithmic steps have running times clustered very near to zero for data sets of all sizes and are not distinguishable on this plot. For this reason, a plot of log(processor time) versus number of data points is shown at bottom. The time for the $DT(P)$ computation is again quite similar to the time for the overall algorithm. The next most time consuming step is the identification of $Vor(P)/CH(P)$ edge intersections, but this step is far faster than the computation of $DT(P)$.

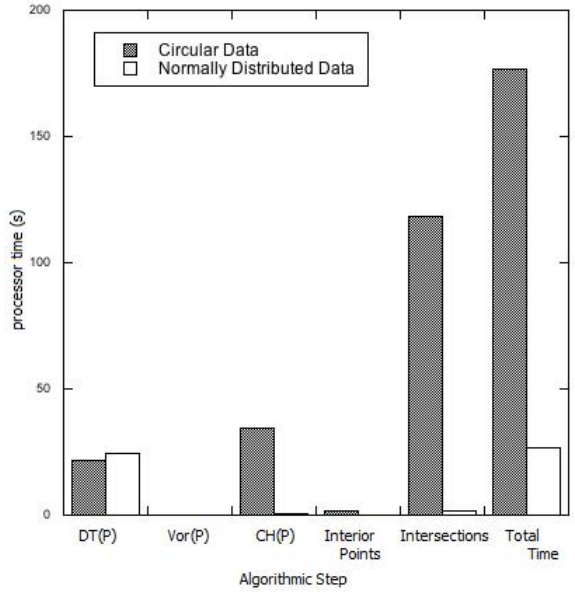

Figure 11: Processor time used by our algorithm on 1000 data points lying on a circle as compared to 1000 normally distributed, randomly generated data points.

sets our algorithm will most commonly encounter will be similar to the data sets on which we have already tested it–somewhat normally distributed, with h small compared to n. However, should we need to compute the LEC on some set of points which is distributed differently (say, in a ring-like shape, where h and n would be similar), we expect our $O(nh)$ intermediate steps to begin to contribute to significantly slower runtimes for our algorithm overall.

To test this idea, we ran our algorithm on a set of 1000 points distributed on a circle so that $h = n$ and compared the resulting performance to the performance on a normally distributed set of the same size. The resulting running times are displayed in Figure 11. Note that the $O(nh)$ check for interior Voronoi vertices is still relatively quick. However, we see that our convex hull algorithm and our technique for locating Voronoi/convex hull intersections take much longer on the circular data than on the normally distributed data, leading to about a sevenfold increase in the overall running time of our algorithm. Thus, if we intend to run our algorithm on non-normally distributed data where h may become large, we would almost certainly benefit from switching to an $O(n \log n)$ convex hull algorithm and an $O(n)$ technique for finding Voronoi/convex

hull intersections. Still, because our algorithm is primarily applicable in domains where data is likely to be normally distributed, such as sets of cities or business locations, we continue to use the naive $O(nh)$ intermediate steps, since they seem to work well for this type of data.

6 Conclusion and Future Work

In this paper, we have describe an algorithm which computes the largest empty circle on a set of points P. The algorithm is $O(n[h + \log n])$ and supports dynamic updates to the set P without heavy re-computation of underlying structures. The algorithm requires the use of the Voronoi diagram, which we compute by dualizing the Delaunay triangulation of P. The convex hull of P must also be computed. We then check all Voronoi vertices and intersections between Voronoi and convex hull edges to see which is the center of the largest empty circle.

Our algorithm is $O(n[h + \log n])$, which is asymptotically worse than the $O(n \log n)$ solutions published by Toussaint (1983) and Preparata and Shamos (1985). This extra h term results from taking naive $O(nh)$ approaches to a few intermediate steps in computing the LEC. However, we have shown that our algorithm's overall running time is not much affected by these $O(nh)$ intermediate steps when our data is more or less normally distributed. Thus, we find that our use of simple, naive techniques at some steps of our algorithm is justified; while asymptotically faster techniques do exist for these steps, these faster techniques are considerably more complicated than the simple approach we take. Because our $O(nh)$ methods contribute very little to the total runtime of our algorithm (Figure 10), we find that we can use simple, straightforward techniques at a negligible cost to the running time of our algorithm.

For future work, it may be useful to implement support for further location constraints on the center of the LEC, such as described in Chew and Drysdale (1986) or Toussaint (1983). For example, we might like to restrict the LEC to be centered in some simple, though not necessarily convex, polygon or set of polygons other than the convex hull. This would have been useful when working with the data set comprised of US cities. Suppose we are using our algorithm to find a toxic waste dump site. Rather than using the convex hull of this data set, which contains parts of Mexico, the Gulf of Mexico, and the Atlantic and Pacific oceans, and does not include all of the land comprising the 48 contiguous states (Figure 8), we might have preferred to constrain the LEC to be centered anywhere on US mainland territory. This would both ensure that the selected site were actually a United States holding, and would also provide a wider selection of possible LEC centers by including more area in the northern United States. Thus, using some sort of simple, polygonal approximation of the 48 contiguous states would be an improvement.

To do this, we would have to change our point location strategy for testing whether a Voronoi vertex is interior to the bounding region, since that region would no longer be convex. The current naive $O(nh)$ technique for finding intersections between the Voronoi diagram and the bounding region would still be effective, but if we were to update to the $O(n)$ marching technique described by Preparata and Shamos (1985), we would have to modify it slightly to deal with non-convex bounding regions.

Our current algorithm avoids complicated intermediate steps and successfully computes the LEC on data sets of varying sizes and distributions. It is quite fast on normally distributed sets. While we could improve its performance on non-normal data sets and support further location constraints on the center of the LEC by using more complicated intermediate steps in our algorithm, for now we stick with the simple solution to the LEC problem and assume it will most often be used on normally distributed data sets.

7 Acknowledgements

I am grateful to Professor Andy Danner for advising this project and for providing a script for graphing the 48 contiguous states. I also thank my Swarthmore College Computer Science Senior Conference classmates for reviewing this paper and providing suggestions for its improvement.

References

B.M. Chazelle, 1980. Computational Geometry and Convexity, Ph.D. thesis, Carnegie-Mellon University.

L.P. Chew and R.L. Drysdale, 1986. Finding Largest Empty Circles with Location Constraints *Dartmouth Computer Science Technical Report PCS-TR86-130*

M. de Berg et al. *Computational Geometry: Algorithms and Applications (2ed)*. Berlin: Springer, 2000. pp 185-197.

A. Okabe et al. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams (2ed)*. Chichester: John Wiley and Sons, Ltd, 2000. pp 43-57.

F.P. Preparata and S.J. Hong. 1977. Convex Hulls of Finite Sets of Points in Two and Three Dimensions. *Communications of the ACM*, v.20, n.2, pp 87-93.

F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. New York, NY: Springer-Verlag, 1985. pp 251-253.

M.I. Shamos, 1978. Computational Geometry, Ph.D. thesis, Yale University.

G.T. Toussaint, 1983. Computing Largest Empty Circles with Location Constraints. *International Journal of Parallel Programming*, v12.5, pp 347-358.