CMSC 754: Spring 2020 Dave Mount

Homework 3: Voronoi Diagrams, Delaunay Triangulations, and More

Handed out Thu, April 16. Due **Thu, April 30, 9:30am**. (**Updated!**) Late homeworks are not accepted (unless an extension has been prearranged) so please turn in whatever you have completed by the due date. Unless otherwise specified, you may assume that all inputs are in *general position*. Whenever asked to give an algorithm running in $O(f(n))$ time, you may give a *randomized algorithm* whose expected running time is $O(f(n))$.

Problem 1. In class we proved that the Delaunay triangulation of a set of sites in the plane maximizes the minimum angle. (It is the max-min angle triangulation.) Unfortunately, it is not the best triangulation for the following two criteria.

(a) Give an example of a set of point sites in the plane such that the Delaunay triangulation of this set *does not* minimize the sum of edge lengths, among all possible triangulations. In other words, the Delaunay triangulation is *not* the minimum-weight triangulation.

(b) Give an example of a set of point sites in the plane such that the Delaunay triangulation of this point set *does not* minimize the maximum angle, among all possible triangulations. In other words, the Delaunay triangulation is *not* the min-max angle triangulation.

In each case briefly explain your construction. Your example should be in general position (e.g., no four sites should be cocircular).

Hint: In both cases, it is possible build a counterexample consisting of just four points that are nearly co-circular. It suffices to present a single, specific example.

Problem 2. The Delaunay triangulation of a convex polygon is defined to be the Delaunay triangulation whose sites are the vertices of the polygon. As usual, let us assume that the vertices $V = \{v_1, \ldots, v_n\}$ of the polygon are presented in counterclockwise order around the polygon. Also we assume that $n \geq 3$ and no three vertices are collinear. In this problem, we will analyze a randomized incremental algorithm for constructing the Delaunay triangulation of a convex polygon.

The algorithm is similar to the randomized incremental algorithm given in class, but with the following differences. First, we *do not* use any sentinel sites, just the points themselves. We begin by permuting the points randomly. Let $P = \langle p_1, \ldots, p_n \rangle$ denote the sequence of vertices after this permutation has been applied. We start with the triangle $\triangle p_1 p_2 p_3$. Then, we go through the points p_4 through p_n, adding each one and updating the Delaunay triangulation as we go. The insertion process for p_i involves the following steps:

(i) Determine the edge ab of the current convex hull that is visible to p_i (see Fig. 1(a)).

(ii) Connect p to the convex hull by adding the edges ap_i and $p_i b$ to the convex hull (see Fig. 1(b)).

(iii) As in Lecture 13, perform repeated edge flips until all the triangles incident to p_i satisfy the local Delaunay condition (see Fig. 1(c)).

Answer the following questions about this algorithm:

(a) Prove that in any triangulation of an n-sided convex polygon, the number of triangles is $n - 2$ and the number of edges (including the edges of the convex hull) $2n - 3$.

(b) What is the average degree of a vertex in the Delaunay triangulation of n? (Include the two edges of the convex hull that are incident to this vertex.) Derive your answer.

(c) Apply a backwards analysis to bound the expected number of edge flips performed when the ith site is inserted into the diagram (for $4 \leq i \leq n$).

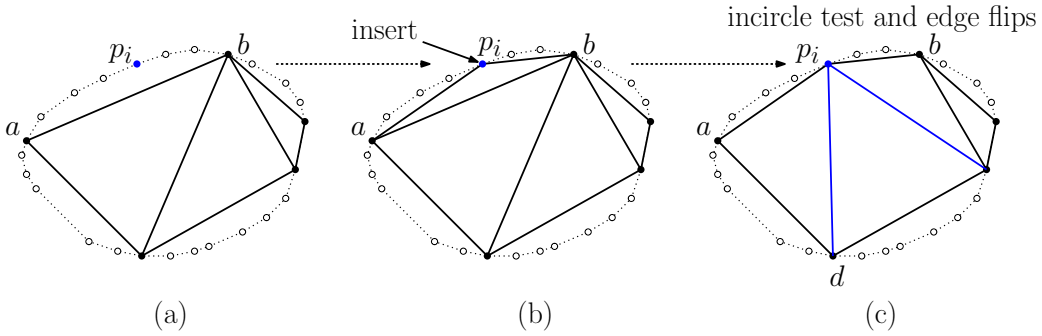

Figure 1: Delaunay triangulation of a convex point set.

(d) In step (i) of the above algorithm, we need to determine the edge ab of the convex hull that is visible to the new site p_i. Describe a method for answering these queries and derive its expected running time. (Hint: As we did in the standard Delaunay Triangulation algorithm, apply some form of bucketing combined with a backwards analysis.) $O(n \log n)$ total expected time is possible, but if you think you can do this in $O(n)$ expected time, see the challenge problem.

Problem 3. In this problem, we will derive a complete version of the empty-circumcircle test for four points in $\mathbb{R}^2$, which considers both the affine and circular parts of the test. Let a, b, and c be three distinct, non-collinear points in the plane, and let us assume that they have been labeled so that $\text{Orient}(a, b, c) > 0$ (that is, they are given in counterclockwise order). Let d be any point in plane. Throughout this problem, you may assume general position: No three points are collinear and no four points are cocircular.

You may access the points *only* through the two determinant-based functions $\text{Orient}(a, b, c)$ (from Lecture 2) and $\text{inCircle}(a, b, c, d)$ (from Lecture 13).

(a) Consider the subdivision of the plane induced by the three lines defining the edges of triangle $\triangle abc$ (see Fig. 2(a)). Present a short code fragment that uses orientation tests to label a point d according to the region in which it lies. For example, given the point d shown in the figure, your code fragment should output the label "2". (Hint: It may simplify your code to think of the labels as binary numbers, e.g., $2 = 010_2$.)

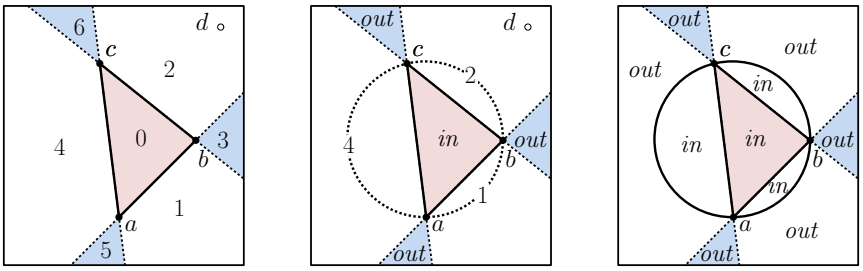

Figure 2: InCircle testing.

(b) Clearly, if d lies in region 0 it is inside the circumcircle of $\triangle abc$, and if it lies in any of the regions 3, 5, or 6, it is outside this circumcircle. Explain how to apply the inCircle function from Lecture 13 to classify d as lying inside or outside the circumcircle of $\triangle abc$ for the remaining regions 1, 2, and

4. (Hint: Recall that this test is based on the sign of a determinant, which was derived under the assumption that the arguments are presented in counterclockwise order.)

Problem 4. Computational biologist often compute physical properties of large molecules. One of these properties is something called the *accessible surface area* of the molecule. We will consider this problem in a 2-dimensional setting.

We model the atoms of our molecule as a set of unit disks, each of radius of 1, centered at a given set of points $P = \{p_1, \ldots, p_n\}$. Let b_i denote the unit disk centered at point p_i. Let $M(P) = \bigcup_{i=1}^{n} b_i$ denote the union of these disks (shaded in blue in Fig. 3(a)). Let $\partial M(P)$ denote its boundary. Observe that $\partial M(P)$ is composed of circular arcs, and it may have multiple connected components (all of which contribute to the accessible surface).

(a) Present an $O(n \log n)$-time algorithm, which given a set P of n points in the plane, computes the length of accessible surface (see the solid curve in Fig. 3(a)).

Note: I am mostly interested in how to identify the circular arcs that make up the boundary of $M(P)$, not the actual perimeter value. To simplify matters, you may assume that you have access to a black-box function that computes the length of a given circular arc.

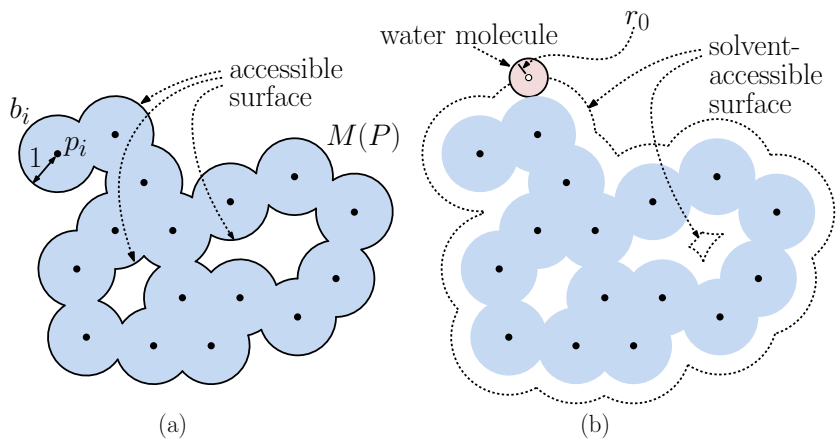

Figure 3: Accessible surface.

(b) In most applications in drug design, the molecule resides in liquid solvent, such as water. A statistic that is more relevant is something called the *solvent-accessible surface*. We model a water molecule as a disk of some radius r_0. Imagine that we roll such a disk around the entire boundary of $M(P)$ and trace out the path taken by the center of this disk (the dashed curve in Fig. 3(b)). As with the accessible surface, this boundary may have multiple components, all of which contribute to the final result.

Present an $O(n \log n)$-time algorithm, which given P and r_0, computes the length of the solvent-accessible surface (the dashed curve in Fig. 3(b)).

Hint: This should be an easy extension to (a).

Problem 5. In linear classification in machine learning it is desirable to determine whether two point sets in $\mathbb{R}^d$ can be partitioned by a hyperplane. We will consider the problem in a 2-dimensional setting. When a perfect partition is not possible, one approach is to find a line that achieves the best possible split. Let A and B be two point sets in the plane. Given a nonvertical line $\ell : y = ax - b$, let ℓ^+ and ℓ^- denote the halfplanes lying above and below ℓ, respectively. Define ℓ's *separation defect* to be

$$\delta(\ell) \;=\; \min\big(|A \cap \ell^+| + |B \cap \ell^-|,\;\; |A \cap \ell^-| + |B \cap \ell^+|\big).$$

So, if ℓ separates A from B the separation defect is zero. Otherwise, it is equal to the number of points that fall on the "wrong" side of the separating line. (For example, the line shown in Fig. 4 has separation defect of three.) In this problem, we will derive a roughly quadratic-time algorithm to compute a line of minimum separation defect.

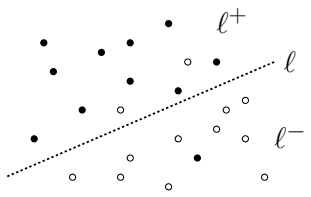

Figure 4: Problem 5: The line ℓ has separation defect 3.

(a) Given two point sets A and B in $\mathbb{R}^2$ and a line ℓ, what is the meaning of ℓ's separation defect in the dual setting? Explain briefly. Assume the standard dual transformation $(a, b) \leftrightarrow y = ax - b$.

(b) Letting $n = |A| + |B|$, present an $O(n^2 \log n)$-time algorithm to compute a line of minimum separation defect. Briefly explain your algorithm and derive its running time. (Note: There may generally be many lines achieving the minimum defect. Your algorithm may output any one of them. If a point lies on ℓ you have a choice of which halfplane to assign it. I will let you decide how to handle this. You may either have your algorithm make the choice explicitly to minimize the defect, or you can just assume that ℓ is chosen so that no point of either set lies on it.)

(Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.)

Challenge Problem 1. Show that Problem 2(d) (locating points in the incremental DT algorithm for convex hulls) can be solved in $O(n)$ time in expectation.

Challenge Problem 2. Prove that if a, b, and c are given in counterclockwise order, the InCircle determinant is positive if and only if d lies inside the circumcircle of $\triangle abc$. (This implies that the affine test used in Problem 3(a) is not needed.)