Solutions to Homework 6

Solution 1: This solution is based on a combination of the Voronoi diagram and the Delaunay triangulation T of the point set. First construct $T = DT(P)$. Delete from T any edge whose length is greater than $2r$, and let $T(r)$ denote the resulting planar graph (see Fig. 1(a)). We compute the connected components of this graph. Given any two query points s and t, we find the closest sites in P to these points, called them $p(s)$ and $p(t)$. If $p(s)$ and $p(t)$ are in the same connected component of $T(r)$ and if the distances $\|s - p(s)\|$ and $\|t - p(t)\|$ are both less than or equal to r, then we report that s and t are safely connected (see Fig. 1(a)). Otherwise, we report that they are not.

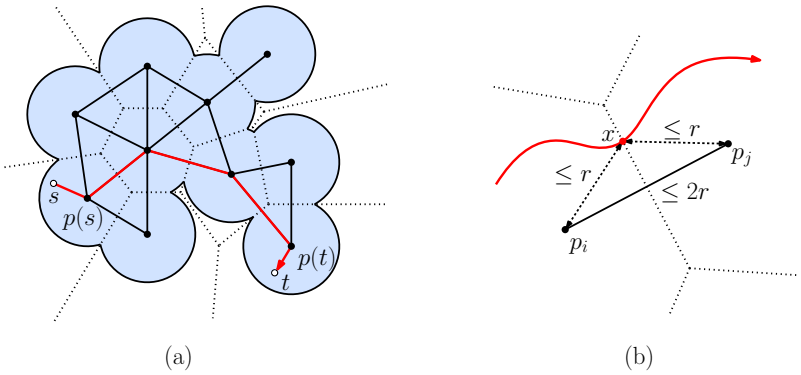

Figure 1: Safe connectivity.

To establish correctness, we show that a safe path exists if and only if the above conditions hold. First observe that if either $\|s - p(s)\|$ or $\|t - p(t)\|$ exceeds r then the initial placements are not safe, and hence there is no safe path. Otherwise, we may safely move s to $p(s)$ and t to $p(t)$. Henceforth, we consider the existence of a safe path between $p(s)$ and $p(t)$.

We assert that a safe path exists between $p(s)$ and $p(t)$ if and only if they are in the same connected component of $T(r)$. Since $T(r)$ consists of edges connecting pairs of sites that are within distance $2r$, it is clear that any path in $T(r)$ is safe. Conversely, consider any safe path between $p(s)$ and $p(t)$, which generally need not travel along the edges of the Delaunay triangulation. We will show that the path can be safely mapped onto the edges of the Delaunay triangulation. Consider any place where this path crosses from one Voronoi cell into another. Let p_i and p_j be the sites associated with these cells at let x be the point on the Voronoi edge where the path crosses (see Fig. 1(b)). Since the path is safe, we have $\|p_i - x\| \leq r$ and $\|x - p_j\| \leq r$, which by the triangle inequality implies that $\|p_i - p_j\| \leq 2r$, which implies that the Delaunay edge (p_i, p_j) is in $T(r)$. By induction, it follows that there exists a path from $p(s)$ to $p(t)$ that lies entirely on Delaunay edges of length at most $2r$, and hence these two sites are in the same connected component of $T(r)$.

The data structure consists of a point-location data structure for the Voronoi diagram (to determine the nearest neighbors of s and t) and a simple labeling of the sites of P based on which connected component they are in. We know from class that the point-location data structure can be built in $O(n \log n)$ time (in expectation), uses $O(n)$ space, and answers queries in $O(\log n)$ time.

The Delaunay triangulation can be built in $O(n \log n)$ time (in expectation), and since it has $O(n)$ edges, its connected components can be computed by DFS in $O(n)$ time.

Solution 2:

(a) We will show that the expected number of edge flips is at most 2. The analysis is similar to the randomized incremental algorithm for the general Delaunay triangulation, but with a small change for the expected degree.

Consider the triangulation just after the insertion of the ith site. By Euler's formula, we know that the average degree in a planar graph with i vertices is $O(1)$. We assert that the average degree in this case is at most 4. To see this, recall from an earlier homework that, by a straightforward application of Euler's formula, a planar triangulation with v vertices and h hull edges has $e = 3v - h - 3$ edges. In our case, $v = h = n$, and hence there are $2n - 3$ edges. Since the sum of vertex degrees in any graph is $2e$, the average degree is $2e/n = (4n - 6)/n = 4 - 6/n \leq 4$.

Adding a site to the convex hull of the current triangulation creates two edges, and each edge flip generates one more incident edge. Therefore, the number of flips per insertion is equal to the new site's degree minus two. Since each of the sites is equally likely to be the last added, the expected degree of the last site to be inserted is at most 4. It follows that the expected number of edge flips is at most $4 - 2 = 2$, as desired.

(b) There are a few different solutions to determining where the new site will be added on the hull. We will employ a bucketing approach, as we did in the incremental Delaunay triangulation algorithm.

We start with the complete list of vertices $\langle p_1, \ldots, p_n \rangle$ in cyclic order around P's boundary. After the ith stage of the algorithm, i vertices have been inserted into the triangulation. The convex hull of this set has i edges, and we partition the remaining vertices among i buckets, one per edge, where each vertex is stored in the bucket associated with the hull edge it is visible to (see Fig. 2(a)). When the next vertex is added, we remove it from its bucket, connect it to the endpoints of the associated hull edge (and initiate the edge-flipping process), and then we split its bucket about this vertex (see Fig. 2(b)).

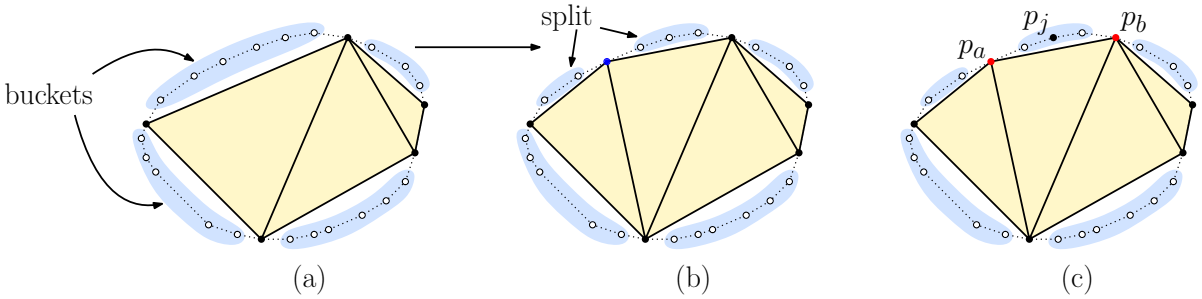

Figure 2: Bucketing vertices.

What is the expected running time for the rebucketing process? Clearly, the time needed for to perform rebucketing after each insertion is proportional to the number of vertices that are rebucketed. Rather than counting the expected number of rebucketed vertices per insertion

and summing over the entire algorithm, we will instead count the expected number of times a fixed vertex p_j is rebucketed throughout the entire course of the algorithm, and then multiply this by n. (Clearly, these two will yield the same result.)

We will do this through a backwards analysis. Suppose that we have inserted i vertices so far. Among the i vertices that are currently in the triangulation, which would have caused p_j to be rebucketed? Clearly, there are only two such vertices, namely, the vertices p_a and p_b defining the hull edge that contains p_j (see Fig. 2(c)). (If either of these had been last, then the creation of edge $\overline{p_a p_b}$ would force p_j to move into this new bucket, and the insertion of any other vertices has no effect on p_j's bucket.) It follows therefore, that the probability that p_j is rebucketed during stage i is $2/i$. Summing over all stages of the algorithm, it follows that total number of times that p_j has been rebucketed is at most

$$\sum_{i=1}^{n} \frac{2}{i} \;=\; 2\ln n + O(1).$$

Since p_j is an arbitrary vertex, it follows that the overall expected rebucketing time is at most $O(n \ln n) = O(n \log n)$, as desired.

Solution 3: Let A^* and B^* denote the dual lines of the points of A and B, respectively. By the order reversal property of the duality transformation, it is easy to see that this problem is dually equivalent to computing a point in the dual plane that minimizes the separation defect with respect to dual lines of A^* and B^* that pass above and below this point. In this setting, we can define the separation defect in the same way, but by considering the number of lines passing above and below the point. So, we will consider the problem entirely within the dual setting.

We will assume the separating line is in general position. Build the arrangement of $(2n)$-element $A^* \cup B^*$. Define the *A-depth* of a point in the arrangement to be the number of lines of A^* that lie on or above this point, and define the *B-depth* analogously for B^*. Perform a traversal of the arrangement (e.g., by depth-first search or through topological plane sweep), keeping track of the A-depth and B-depth of each face in the arrangement as we go. Clearly these level values can be incremented or decremented in $O(1)$ time as we move from one face to another. The A-depth gives the value of $|A \cap \ell^+|$, and the value of $|A \cap \ell^-|$ can determined as n minus the A-depth. Thus, in the traversal, we can update the value of $\delta(\ell)$ in $O(1)$ time for each new face visited, for a total time of $O(n^2)$.

Solution 4: Consider a line segment with endpoints p and q. The duals of these two points are two lines p^* and q^*. A line ℓ stabs the segment $\overline{pq}$ if and only if p and q lie on opposite sides of ℓ, which is equivalent to saying the the dual point ℓ^* lies in the *double wedge* region between p^* and q^* (see Fig. 3(a)).

Our approach to both parts involves traversing all the cells of an arrangement and computing the *wedge depth* of each cell, by which we mean the number of double wedges that overlap this cell (see Fig. 3(b) and (c)). We assert that this can be computed either by traversing the line arrangement of all the segment endpoints (viewed as a planar cell complex) or more practically by performing a plane sweep through the arrangement. To do this, there are two pieces of information to keep track of. For each line ℓ in the sweep-line status we store:

- Is ℓ the upper edge of its double wedge or the lower edge? This information swaps whenever we sweep the central vertex of the double wedge.

- What is the wedge depth of the cell immediately below ℓ in the arrangement.

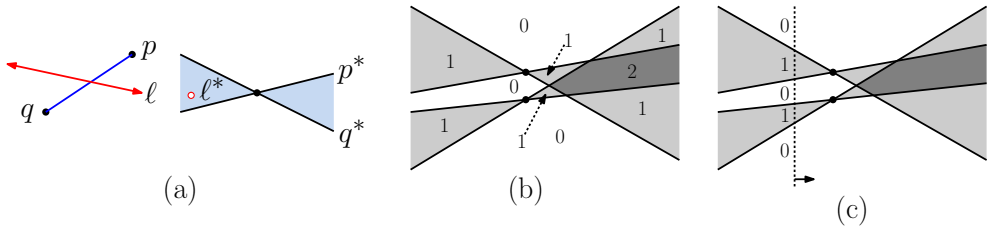

Figure 3: Computing the wedge-depth in a line arrangement.

We assert that we can maintain these two pieces of information in $O(1)$ time for every intersection event we process in the plane sweep. (We omit the straightforward details, but they depend on whether these lines are from the same double wedge or, if not, which of the lines is upper and/or lower.)

(a) For this problem, we are seeking any cell that has wedge depth zero, excluding the cells associated with the upper envelope and lower envelope. Any point in such a cell corresponds to a line that stabs no segment, and has at least one endpoint above and one endpoint below, which implies that at least one entire segment lies entirely above and one entirely below.

(b) For this part, all we need to do is to determine whether there is a cell in the line arrangement whose wedge depth is exactly n. Any point in such a cell corresponds to a line that stabs all the segments. If no such cell exists, then no such stabbing line exists.

(c) Unlike the vertical segment case, where we knew that the upper endpoint lies above stabbing line and the lower endpoint lies beneath it, it is not generally easy to determine which is the case with arbitrary segments (see Fig. 4).

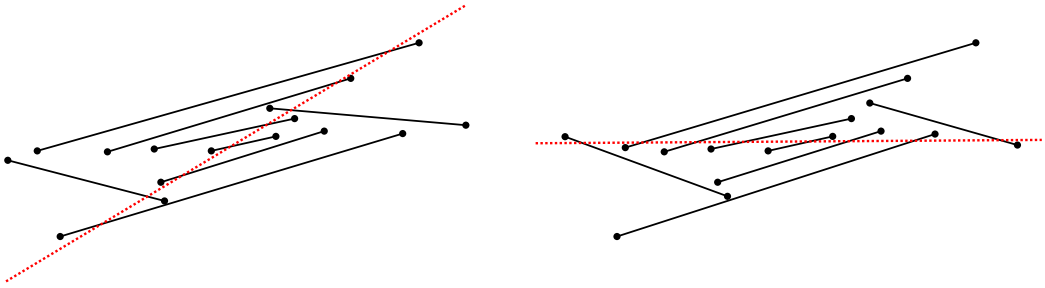

Figure 4: Which side to stab from?

There is a way to solve the problem in $O(n^2)$ time by $O(n)$ invocations of LP. The idea is to sort the line segments by their slopes, and for each consecutive pair of slopes θ_i and θ_{i+1} in sorted order, you can write an LP testing whether there is a stabbing line whose slope lies in the interval (θ_i, θ_{i+1}). Once the slope is constrained, it is possible to know which endpoints are above and which are below.

Solution to the Challenge Problem:

(a) Recall that the arcs of the beach line are parabolic arcs. We claim that any two parabolas intersect in at most two points. To see this, let $y = P(x)$ and $y = Q(x)$ be the equations for two parabolas. The x-coordinates of their common intersection will be at the roots of the equation $P(x) - Q(x) = 0$. This is a quadratic equation in x, and hence it has at most two real roots.

If such an alternation were to occur, then by the continuity of parabolas, between each pair of alternating arcs there must an intersection point. Three alternations would imply the existence of three intersection points, a contradiction.

(b) Let $L(n)$ denote the maximum length of a sequence of n distinct characters that satisfy our constraints. We prove that $L(n) \leq 2n - 1$ by induction on n. This is easily verified for $n = 1$.

Assume the hypothesis holds for any number of distinct symbols strictly less than n. Let the symbols be $\{a_1, \ldots, a_n\}$. Consider the first and last occurrences of the symbol a_n. They subdivide the string into three strings as shown below

$$w_1 \ a_n \ u \ a_n \ w_2,$$

where w_1, u, and w_2 are strings, and w_1 and w_2 contain no occurrences of a_n. Let w be the concatenation of w_1 and w_2. Observe that if symbol $a_i \neq a_n$ occurs in w then it cannot occur in u, and vice versa, since otherwise we would have a forbidden alternation. Thus, if k denotes the number of distinct characters in u then there are $n - k$ distinct characters in w. Clearly there can be no forbidden alternations in u nor in w. Also u cannot have any repeated symbols. It is possible to have a single repeated symbol in w (where w_1 and w_2 come together) so, if we count this repeated character and the lengths of the strings u and w, the total number of symbols in the string is at most

$$L(n) \ \leq \ L(k) + L(n-k) + 1 \ \leq \ (2k-1) + (2(n-k)-1) + 1 \ \leq \ 2n - 1,$$

completing the proof.