CMSC 754: Spring 2026 Dave Mount

Solutions to Quiz 3

Solution 1:

(a) We will show that the expected number of edge flips is at most 2. Consider the triangulation just after the insertion of the ith site. We assert that the average degree of any vertex in the triangulation 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 edge flips per insertion equals the new site's degree minus 2. Since each site is equally likely to be the last added, the expected degree of the last site added is at most 4. It follows that the expected number of edge flips per insertion is at most $4 - 2 = 2$, as desired.

(b) 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. 1(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. 1(b)).

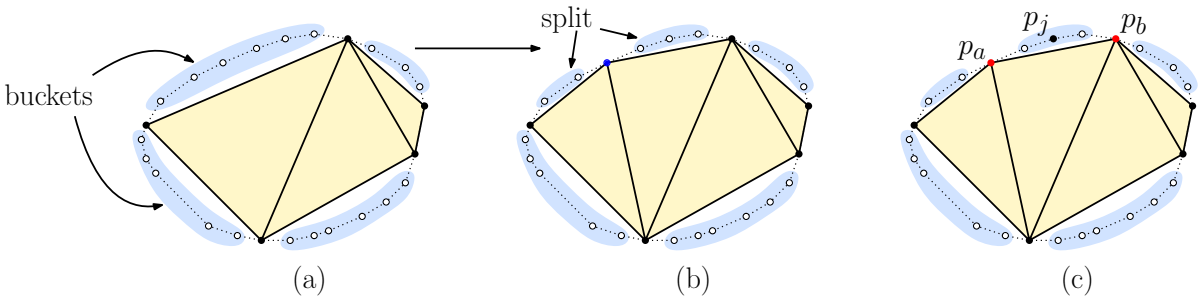

Figure 1: Bucketing vertices.

The time required to rebucket after each insertion is proportional to the number of vertices 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.

We apply a backwards analysis. Let p_a and p_b be the two sites defining the hull edge that p_j lies outside of just after the ith insertion. Observe that p_j is rebucketed if and only if either p_a or p_b was the last to be inserted (see Fig. 1(c)). Since each point is equally likely to have

been the last, a backwards analysis implies that the probability that p_j is rebucketed during stage i is $2/i$. (To simplify things, we're ignoring the boundary case when $i = 1$.) Summing over all stages of the algorithm, it follows that the 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 2: To begin, let $\ell_{\max}$ denote the length of the longest edge in the EMST of P. We assert that the critical radius r^* equals half the length of the longest edge in the Euclidean minimum spanning tree (EMST) of P, that is, $r^* = \ell_{\max}/2$.

We first show that $r^* \leq \ell_{\max}/2$. Since every edge of the EMST has length at most $\ell_{\max}$, the balls of radius r^* placed at the endpoints of each edge cover the entire edge. Since the EMST is connected, it follows that the union of balls is connected (see Fig. 2(a)).

Conversely, to show that $r^* \geq \ell_{\max}/2$, suppose we run Kruskal's MST algorithm, but we restrict ourselves to edges of strictly shorter length than $\ell_{\max}$. Because Kruskal's algorithm adds edges in increasing order of length, it follows that the algorithm will fail to complete the spanning tree. By considering the subtrees of the partial spanning tree, it follows that it is possible to partition the point set into disjoint subsets $P_1, P_2 \subset P$, such that for any $p_1 \in P_1$ and $p_2 \in P_2$, $\|p_1 - p_2\| > \ell_{\max}$ (see Fig. 2(b)). This implies that if we place disks of radius $\ell_{\max}/2$ at all the points of P, no ball of P_1 will overlap any ball of P_2, and hence the balls are not connected.

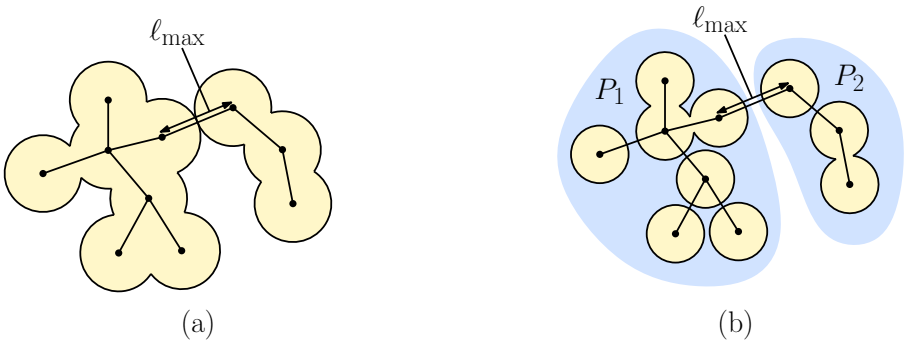

Figure 2: Computing the critical radius.

We conclude that, $r^* = \ell_{\max}/2$. We can compute r^* in $O(n \log n)$ time as follows. Recall that (as proved in class) EMST(P) is a subgraph of DT(P). We first compute the Delaunay triangulation of P, and then compute the MST of the Delaunay triangulation (e.g., by any standard MST algorithm). The overall running time is $O(n \log n)$. When the MST is constructed, we inspect its edges and output half the length of the longest edge.

Solution 3:

(a) We will show that (p, q) is an edge of RNG(P) if and only if interior(lune(p, q)) contains no other site. Let B_p and B_q denote the balls centered at p and q, respectively, whose radii are both equal to $\|p - q\|$ (see Fig. 3(a)). By definition lune$(p, q) = B_p \cap B_q$, and hence

site $r \in \text{interior}(\text{lune}(p,q))$ if and only if $r \in \text{interior}(B_p \cap B_q)$, which holds if and only if $\|r-p\| < \|p-q\|$ and $\|r-q\| < \|p-q\|$, that is, $\max(\|r-p\|, \|r-q\|) < \|p-q\|$. Hence it follows that, (p,q) is an edge of $\text{RNG}(P)$ if and only if there is no site $r \in \text{interior}(\text{lune}(p,q))$.

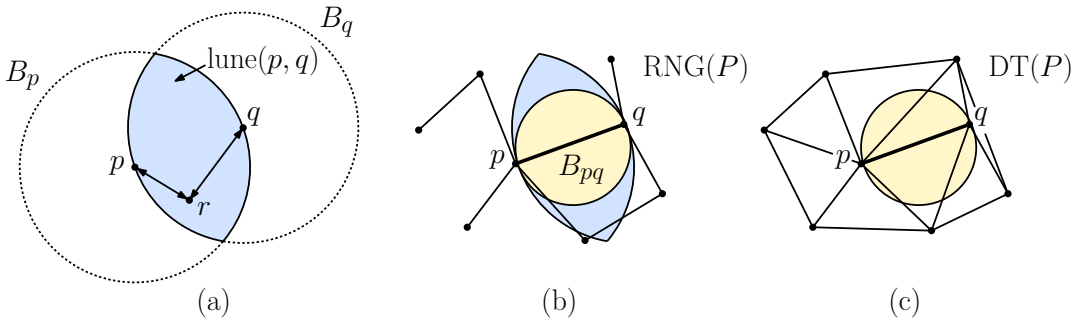

Figure 3: Lunes and the relative neighborhood graph.

(b) To prove that $\text{RNG}(P) \subseteq \text{DT}(P)$, consider any edge $(p,q) \in \text{RNG}(P)$ (see Fig. 3(b)). To prove that $(p,q) \in \text{DT}(P)$ it suffices to show (by the empty-circle property of Delaunay triangulations) that there exists a circle through p and q that contains no other site in its interior. Consider the ball B_{pq} whose diameter is $\overline{pq}$. It is easy to see by a visual inspection that this ball lies entirely within $\text{lune}(p,q)$. (More formally, since $\overline{pq}$ is the diameter of the circle, for any point r in the circle, we have $\|r-p\| < \|p-q\|$ and $\|r-q\| < \|p-q\|$, implying that the circle lies within the lune.) By (a), there is no other site lying within the lune, and implying that there is other site lying with B_{pq}, implying that $(p,q) \in \text{DT}(P)$, as desired.

Solution 4:

(a) The dual set R^* is a set of m lines in $\mathbb{R}^2$ and B^* is a set of n lines in $\mathbb{R}^2$. The dual of the cutting line ℓ is a point ℓ^*, lying at the intersection of a line of R^* and a line of B^* such that $(m-1)/2$ lines of R^* lie above ℓ^* and $(n-1)/2$ lines of B^* lie above ℓ^*.

(b) We perform a topological plane sweep through the dual line arrangement of $R^* \cup B^*$. There are $m+n$ lines, and so the arrangement has total complexity of $(m+n)^2$. The sweep-line status consists of an array of $m+n$ lines that intersect the sweep line, sorted from top to bottom (see Fig. 4(a)). (Even though the above/below counts maintained by the algorithm does not rely on this sorted order, the plane-sweep does need it for determining which events to schedule for processing.) For each entry of this array, that is, for each line of the arrangement, we a count of the number of lines of the same color that lies above it at the current location of the sweep line. In addition, for each pair of consecutive lines in this array, we maintain their intersection point, and if it is to the right of the current sweep line, we create an event for this intersection point. Because there are $m+n$ lines, there are at most $m+n-1 = O(m+n)$ events to maintain.

To start the sweeping process, the lines are sorted (from top to bottom) in increasing slope order and the counts and events are initialized. This can be done in $O((m+n)\log(m+n))$ time.

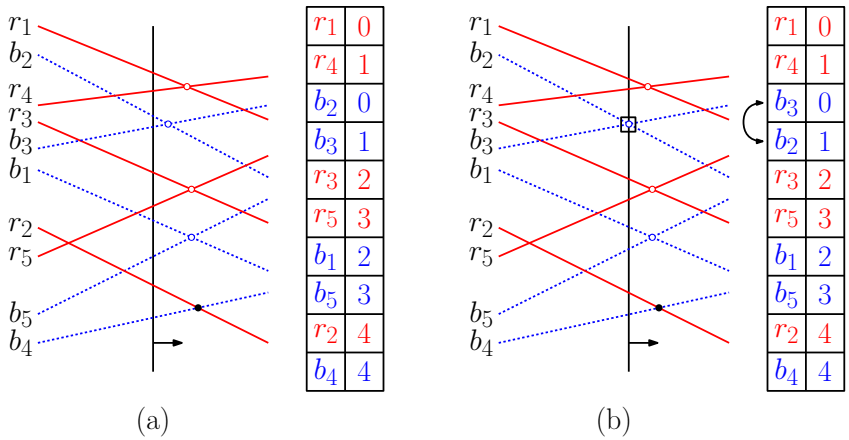

Figure 4: Ham-sandwich cut plane sweep.

The events are the points of intersection of the lines. We swap the two lines in the sweep-line status and their associated counts. If the two lines have the same color, we update their associated counts (see Fig. 4(b)). This affects the adjacency relations of $O(1)$ pairs of lines along the sweep-line status, and we update the future events accordingly by deleting events for those that are no longer consecutive and adding events for those that are. If the lines are of different colors, we check whether the count associated with the red line is equal to $(m-1)/2$ and the count associated with the blue line is $(n-1)/2$. If so, we report this intersection point as the desired HSC point ℓ^*. (We can stop the algorithm at this point, and return ℓ as the final answer.)

Clearly, each event can be processed in $O(1)$ time, since all that is required is swapping two elements in an array. If the events are scheduled using a topological plane sweep, then the overall running time is $O((m+n)^2)$ and the space is $O(m+n)$. Note that by the Ham-Sandwich Theorem, such a cutting line exists, so the algorithm will successfully return before the plane sweep terminates.