CSCI 534: Homework 05

Nathan Stouffer – Collaborated with Elliott Pryor

Problem 1

In Homework 1, we considered a plane-sweep algorithm for determining whether there is any intersection among a collection of n circles in the plane. Here we consider a variant of this problem. The input consists of a collection of n closed circular disks, all having the same radius. (Via scaling, we may assume that they are all unit disks.) Let $C = \{c_1, \dots, c_n\}$ denote the center points of these disks, and let $\{D_1, \dots, D_n\}$ denote the actual disks. Thus, D_i consists of the points that lie within unit distance of c_i. Let $U = D_1 \cup \dots \cup D_n$ denote the union of these disks. The boundary of U may generally consist of multiple parts, each of which consists of a cycle of circular arcs connected by vertices. (In Fig. 4 the boundary consists of three cycles. The vertices are shown as white dots).

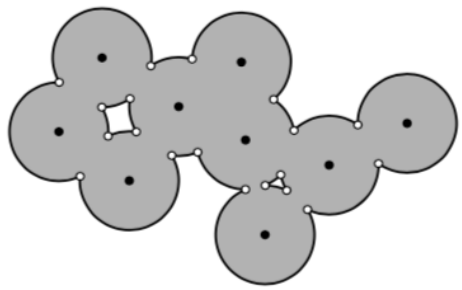

Figure 1: Problem 4: Union of disks

1. Present an algorithm that reports all the vertices on the boundary of U. (Note that circle intersection points in the interior of the union are explicitly excluded.) Your algorithm should run in time $O(n \log n)$. The order in which the vertices are output is arbitrary. (Hint: Don't try to modify the algorithm from Homework 2. A different approach is needed.... think giraffes)

2. Prove that the number of vertices reported by your algorithm is $O(n)$.

Problem 2

Suppose we are given a subdivision of the plane into n convex regions. We suspect that this subdivision is a Voronoi diagram, but we do not know the sites. Develop an algorithm that finds a set of n point sites whose Voronoi diagram is exactly the given subdivision, if such a set exists.

Answer: So we are given a subdivision of the plane into convex regions and we wish to report a set of n sites whose Voronoi diagram is the subdivision (if it possible). Let's consider two trivial cases before giving an algorithm. If $n = 1$, then any point works and if $n = 2$ we can just take any point not on the edge of the subdivision and its reflection. From here on out, we will assume $n > 2$. For general position, we will assume that no vertex in the subdivision has degree higher than 3.

Label the subdivsion D. Suppose for some face in D, we have a single point p that must be the site of the face (if D is a Voronoi diagram). Assume that the faces of D are labeled $1, 2, \ldots, n$. We will prove that we can compute that point p later in the problem.

Here is a quick prose description of the algoritm. We take D to be a DCEL, p is the site we know must exist, and k is the label for the face that p belongs to. We are going to start at p and reflect it across each edge of the face k. If p is a site in a voronio diagram, its reflection across an edge must also be a site. We record every reflected point in an array and add faces to a processing queue as we come across them (note each face can only be added once). If we ever find that a newly reflected point conflicts with a point already found in that face, we know that the subdivision is not a voronoi diagram. We now present the algorithm in pseudocode.

Algorithm 1 Computing the Voronoi Sites

```
function VoronoiSites(D, p = (p_x, p_y), k)
    S ← [null, ..., null] // empty array of size n
    S[k] ← p
    queue ← [k] // a queue for face labels
    while queue is not empty do
        i ← pop the queue
        for every edge e of face i do
            j ← label of the face across e
            q ← reflection of the point S[i] across e
            if S[j] = null then
                S[j] ← q
                add j to the queue
            else if S[j] ≠ q
                return NOT VORONOI
            end if
        end for
    end while
    return S
end function
```

Let's discuss run time. As noted in the prose description, each face can only be added to the processing queue once. This is because once a face is in the queue, it's value in S is no longer null.

Thus the while loop runs n times: once for each face. As a gross upper bound for the for loop, it can run at most E times where E is the number of edges in the DCEL. Since a DCEL represents a planar graph, $E = O(n)$ so our algorithm runs in $N * O(n) = O(n^2)$ time.

And now for correctness!

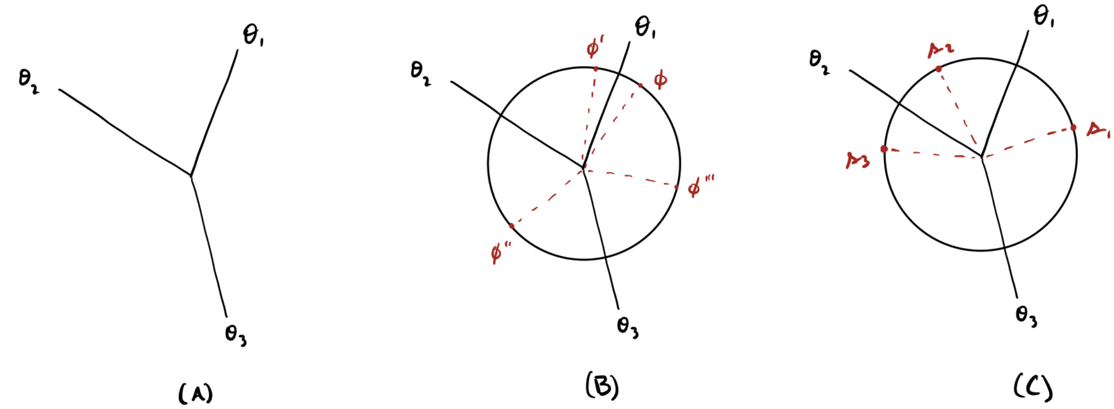

Figure 2: Solving for ϕ

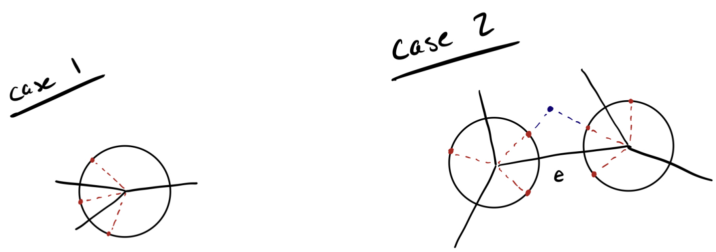

Figure 3: Computing the site