Homework 2: Plane Sweep and Linear Programming

Handed out Thursday, Feb 27. Due at the start of class on Tuesday, Mar 10. 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 given in *general position*. Also, when asked to give an algorithm with running time $O(f(n))$, it is allowed to give a *randomized* algorithm with *expected* running time $O(f(n))$.

Problem 1. You are given an axis-parallel rectangle R and a collection of closed circular disks $D = \{d_1, \ldots, d_n\}$. Assume that each disk d_i is represented by its center point p_i, which lies within R, and its (positive real) radius r_i. Note that the disks may extend outside of R, and one disk may be contained within another.

(a) The elements of D are said to form a *packing* of R if every point of R lies within *at most* one disk of D (see Fig. 1(a)). Present a plane-sweep algorithm that determines whether D is a packing of R. **Hint:** $O(n \log n)$ time is possible.

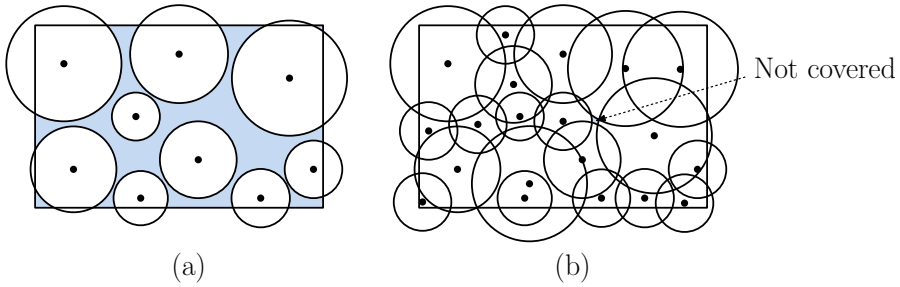

Figure 1: Problem 1: Packing and covering a rectangle

(b) The elements of D are said to form a *cover* of R if every point of R lies within *at least* one disk of D. (The disks of Fig. 1(b) fail to be a cover because of the small gap shown in the figure.) Present a plane-sweep algorithm that determines whether D is a cover of R.

Hint: $O((n + m) \log n)$ time is possible, where m is the number of intersection points lying within R between the boundaries of disks. The running time should *not* depend on the number of intersection points that lie outside of R.

In both cases, you may assume access to primitive operations on circles and disks (e.g., computing the intersection points of two circles or the containment of one disk within another). As always, briefly justify your algorithms' correctness and derive their running times.

Note on terminology: By the term *circle*, we mean the set of points of $\mathbb{R}^2$ that are equidistant from a center point. The associated *closed disk* is the set of points that lie at or within this distance, and the associated *open disk* is the set of points that lie strictly closer to the center. Given a center point $p \in \mathbb{R}^2$ and radius $r \geq 0$, we can define the circle, closed disk, and open disk as the set of points $q \in \mathbb{R}^2$ such that $\|q - p\| = r$, $\|q - p\| \leq r$ and $\|q - p\| < r$, respectively, where $\|q - p\|$ denotes the Euclidean distance between q and p.

Problem 2. Explain how to solve each of the following problems in linear (expected) time. Each can be modeled as a linear programming (LP) problem, perhaps with some additional pre- and/or post-processing. In each case, explain how the problem is converted into an LP instance and how the answer to the LP instance is used/interpreted to solve the stated problem.

(a) You are given two point sets $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$ in the plane. Let $p_i = (p_{i,x}, p_{i,y})$ and $q_i = (q_{i,x}, q_{i,y})$. You are told that these two sets are separated by a vertical line $x = x_0$, with P to the left and Q to the right (see Fig. 2(a)). Show how to efficiently compute the line equations (in the form $y = ax + b$) for each of the following "tangent lines" (that is, support lines) of both $\text{conv}(P)$ and $\text{conv}(Q)$.

(i) ℓ_1: Lies above both P and Q
(ii) ℓ_2: Lies below both P and Q
(iii) ℓ_3: Lies above P and below Q
(iv) ℓ_4: Lies below P and above Q

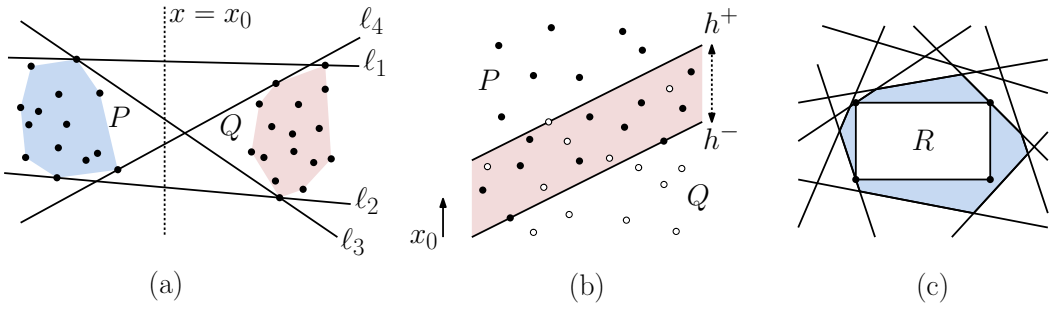

Figure 2: Problem 2: LP applications

Note that the convex hull is *not* given, just the points. In each case, your algorithm should run in time $O(n + m)$. (Briefly justify correctness and explain the running time in each case.)

(b) The following problem is related to the convex of *support vector machines* from machine learning. We are working in $\mathbb{R}^d$ (where d is a constant), where each point x is represented by a coordinate vector $(x_0, \ldots, x_{d-1})$. For the sake of illustration, let's think of the x_0 axis as pointing "upwards". Any nonvertical hyperplane can be expressed as d-vector of real coefficient $a = (a_0, \ldots, a_{d-1})$, by the equation $x_0 = a_0 + \sum_{i=1} d - 1 a_i x_i$.

You are given two sets of points $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$ in $\mathbb{R}^d$. Explain how to compute the equations of two parallel hyperplanes:

$$h^+ : x_0 = a_0^+ + \sum_{i=1}^{d-1} a_i x_i \qquad \text{and} \qquad h^- : x_0 = a_0^- + \sum_{i=1}^{d-1} a_i x_i,$$

such that every point of P lies on or above h^- and every point of Q lies on or below h^+ (see Fig. 2(b)).

The *signed vertical distance* between h^+ and h^- is the vertical distance between these hyperplanes if h^+ lies above h^- and the negation of this distance otherwise. Your hyperplanes should be chosen to minimize the signed vertical distance between them. (Thus, if h^+ lies below h^-, you want to maximize the distance between them.)

Your algorithm should run in time $O(n + m)$. (Briefly justify your algorithm's correctness and explain its running time.)

(c) You are given a set of n halfplanes $H = \{h_1, \ldots, h_n\}$ in the plane. For concreteness, let us assume that each halfplane h_i is given by the inequality $a_i x + b_i y \leq 1$, for some real coefficients (a_i, b_i). Show how to compute the axis-parallel rectangle R of maximum perimeter that lies within the intersection of these halfplanes. (You may assume that R is given by the x- and y-coordinates of its left, right, top, and bottom sides.)

Your algorithm should run in time $O(n)$. (Briefly justify your algorithm's correctness and explain its running time.)

Problem 3. The purpose of this problem is to get practice with designing and analyzing randomized incremental algorithms. You are to devise a randomized algorithm for computing the upper hull of a set of points P in the plane. The approach is as follows:

(1) Deterministically, compute the leftmost and rightmost points, denoted v_1 and v_n, respectively. The initial upper hull is the segment $\overline{v_1, v_n}$.

(2) Randomly permute the remaining points, which we denote as $\langle p_2, p_3, \ldots, p_{n-1} \rangle$.

(3) For i running from 2 to $n-1$, add point p_i to the hull as follows (see Fig. 3(a)):

  (a) Determine the edge $\overline{v_j v_{j+1}}$ of the hull lying above or below p_i (see Fig. 3(b)).

  (b) If p_i lies below this edge, then ignore it.

  (c) Otherwise, walk outwards to the left and right of $\overline{v_j v_{j+1}}$ to compute points of tangency with respect to p_i, and delete any vertices that lie beneath these tangent lines. Add p_i to the hull.

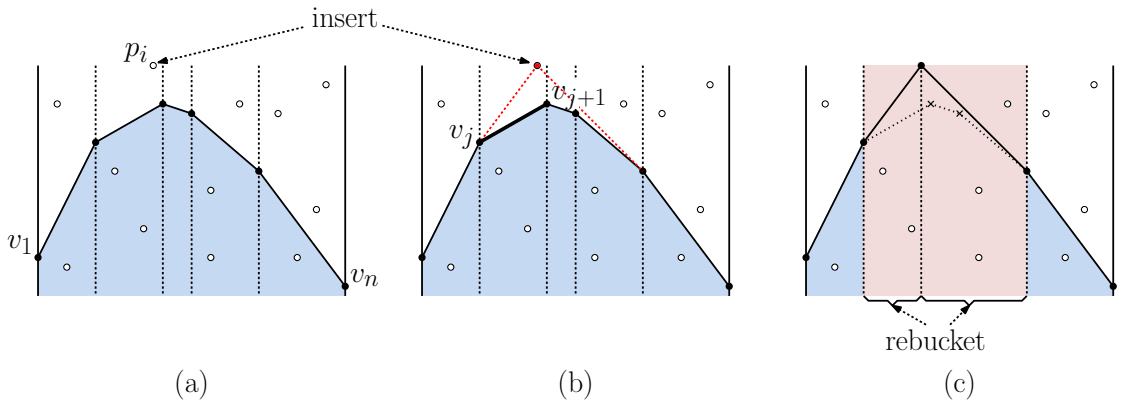

Figure 3: Incremental construction of the upper hull.

How do we determine the edge lying immediately above or below p_i? We maintain a collection of *buckets*, one per edge of the hull, where each contains all the points that lie vertically above or below this edge. Whenever a new point is added to the hull, the points lying within the bucket of a deleted edge need to be "rebucketed". We assume that the points within each bucket are sorted by their x-coordinates. We merge the points from the deleted buckets (in time proportional to the number of points) and then walk through this sorted list and assign these points to their new buckets.

(a) Show that (ignoring the time for rebucketing) the above algorithm runs in $O(n)$ time.

(b) Show that, assuming that points are inserted in random order, the expected number of times any point is rebucketed is $O(\log n)$. In particular, show that for any point p, the probability that p is rebucketed during the ith insertion is at most c/i, where c is a constant. What is the value of c?

It is interesting to note that, unlike the other convex hull algorithms we have seen so far, this one generalizes to arbitrary dimensions.

Problem 4. In this question, we will explore an alternate way in which to define point-line duality in the plane. (This generalizes to any dimension, but the plane is the simplest case to work with.) Any line ℓ in the x, y-plane that does not pass through the origin can be expressed by the equation $\ell : ax + by = 1$. Let us define the *polar dual* of ℓ, denoted ℓ^* to be the point (a, b). Similarly, given a point $p = (a, b)$, define its dual to be the line $p^* : ax + by = 1$. Clearly, this mapping is self-inverse, since $p^{**} = p$ and $\ell^{**} = \ell$.

(a) Given a line $\ell : ax + by = 1$, define its *(open) inner halfplane*, denoted $\ell_<$, to be set of points that lie on the same side of ℓ as the origin, that is $(x, y) \in \ell_<$ if $ax + by < 1$ (see Fig. 4(a)). Define $\ell_>$ analogously for points on the opposite side of ℓ, and define $\ell_\leq$ and $\ell_\geq$ analogously for the closed halfplanes. We say that point p is *inside* (resp., *outside*) ℓ if $p \in \ell_<$ (resp. $\ell_>$).

Prove that polar duality is "order reversing" by showing that point p lies inside/on/outside of ℓ if and only if ℓ^* lies inside/on/outside of p^*, respectively (see Fig. 4(b)).

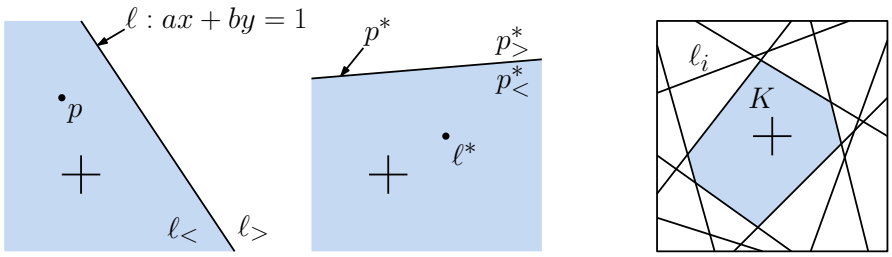

Figure 4: Polar dual.

(b) Prove that polar duality is "incidence preserving" by showing that ℓ_1 and ℓ_2 intersect at point p if and only if the dual line p^* passes through dual points ℓ_1^* and ℓ_2^*.

(c) Let $L = \{\ell_1, \ldots, \ell_n\}$ be a set of lines in $\mathbb{R}^2$ and let K be the intersection of the associated (closed) inner halfplanes, that is, $K = \bigcap_{i=1}^n \ell_\leq$ (see Fig. 4(c). Let's assume that K is not empty (which implies that it contains the origin).

Describe the relationship between K and the convex hull of the polar dual set of points $L^* = \{\ell_1^*, \ldots, \ell_n^*\}$? In particular, explain how the CCW order of edges around the boundary of K relates to the order of points around $\text{conv}(L^*)$. Justify your answer.

Challenge Problem. (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.)

Let P be a simple polygon with n sides. We say that two vertices v_i and v_j of P are *monotonically reachable* if there is an x-monotone path from v_i to v_j. (Fig. 5 shows a number of vertices of P that have x-monotone paths between them.) Present an $O(n \log n)$ time algorithm that computes a count of the total number of monotonically reachable pairs.

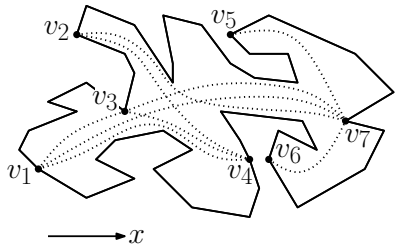

Figure 5: Some of the monotonically reachable pairs for the vertices shown: $\{v_1, v_3\}$, $\{v_1, v_4\}$, $\{v_1, v_7\}$, $\{v_2, v_4\}$, $\{v_2, v_7\}$, $\{v_3, v_4\}$, $\{v_3, v_7\}$, $\{v_5, v_7\}$, $\{v_6, v_7\}$.)

Hint: Note that the number of pairs can be quadratic in n. To achieve a running time of $O(n \log n)$ you cannot count pairs one by one, you need to count them in groups.