**Solutions to Homework 3**

**Solution 1:**

(a) To compute the largest square contained in a polygon, let's model the square as a 3-element vector $(c_x, c_y, r)$, where $c = (c_x, c_y)$ is the square's center and $r$ is half the side length. The four corners are $q_1 = (c_x - r, c_y - r)$, $q_2 = (c_x - r, c_y + r)$, $q_3 = (c_x + r, c_y - r)$, $q_4 = (c_x + r, c_y + r)$. The square lies within the polygon if and only if all four points lie within the polygon's bounding halfplanes. Since the polygon has $n$ vertices and hence is bounded by $n$ halfplanes, this yields $4n$ constraints to be satisfied. (With some care this can be reduced to just $n$, since if we know the slope of a side of $P$, we can determine which of the four vertices to test.) The objective is to maximize $r$ (or more precisely, to maximize the dot product $(0, 0, +1) \cdot (c_x, c_y, r)$.)

Here are the messy details (which are not required for full credit.) Let's assume that the vertices are enumerated in counterclockwise order. Thus, each corner $q_j$ of the square must lie to the left of the directed edge $\overrightarrow{p_i p_{i+1}}$ (assuming indices wrap around). As we know, this can be reduced to an orientation test, namely, $\text{orient}(q_j, p_i, p_{i+1}) \geq 0$.

The orientation is the sign of a determinant. We claim that this condition can be expressed as a linear inequality of the parameters $c_x$, $c_y$, and $r$. To see this, let's consider the concrete case of $q_1 = (c_x - r, c_y - r)$ and the edge $\overrightarrow{p_1 p_2}$. Letting $|\cdot|$ denote matrix determinant, we have the constraint

$$\begin{aligned} 0 \;\leq\; & \begin{vmatrix} 1 & c_x - r & c_y - r \\ 1 & p_{1,x} & p_{1,y} \\ 1 & p_{2,x} & p_{2,y} \end{vmatrix} \\ =\; & \begin{vmatrix} p_{1,x} & p_{1,y} \\ p_{2,x} & p_{2,y} \end{vmatrix} - (c_x - r) \begin{vmatrix} 1 & p_{1,y} \\ 1 & p_{2,y} \end{vmatrix} + (c_y - r) \begin{vmatrix} 1 & p_{1,x} \\ 1 & p_{2,x} \end{vmatrix} \\ =\; & \Delta_1 - (c_x - r)\Delta_2 + (c_y - r)\Delta_3, \end{aligned}$$

where $\Delta_1$, $\Delta_2$, and $\Delta_3$ are the three given $2 \times 2$ determinants. This is equivalent to

$$\begin{pmatrix} \Delta_2, -\Delta_3, \Delta_3 - \Delta_2 \end{pmatrix} \begin{pmatrix} c_x \\ c_y \\ r \end{pmatrix} \;\leq\; \Delta_1,$$

which (since the $\Delta$'s can be computed from the inputs) is a linear inequality in the variables $(c_x, c_y, r)$. (Note that we cannot use general position to rule out an orientation being zero, since in general, some vertices of the maximum square will lie on the boundary of some edges of $P$.)

Observe that the LP must return a feasible, bounded result, since the polygon is bounded and nonempty. Given the value of $(c_x, c_y, r)$ we can construct the corners of the final square as described above.

(b) While it appears at first that three variables are needed, two for the cue-balls starting coordinates and one for the direction, this can actually be solved as a two-dimensional LP. All we really need is the slope $a$ and the $y$-intercept $b$ of the line of the initial shot. Given these, we can select the starting point $(s_x, s_y)$ to be any point on the line that lies to the left all the points. So, we actually have an LP in $\mathbb{R}^2$.

There are two approaches to dealing with the fact that there are four segments to the cue ball's trajectory. The first is to determine the equations of the four separate line motions, and apply them to the points. If we model the initial strike by the line $L_0 : y = ax + b$, the other three lines are $L_1$ (reflected about the lower wall), $L_2$ (reflected about the lower and right walls), and $L_3$ (reflected about the lower, right, and top walls). Formally:

$$L_1 : y = -ax - b, \quad L_2 : y = ax - (2\ell a + b), \quad \text{and} \quad L_3 : y = -ax + (2\ell a + b + 2w).$$

From these we obtain $4n$ constraints, declaring that the points must be above $L_0$ and $L_1$ and below $L_2$ and $L_3$. For example, given a point $p_i$ and the line $L_2$, we have the constraint

$$p_{i,y} \;\geq\; a \cdot p_{i,x} + (2\ell a + b) \;\text{ or equivalently }\; a \cdot (p_{i,x} + 2\ell) + b \;\leq\; p_{i,y},$$

which is clearly linear in $a$ and $b$.

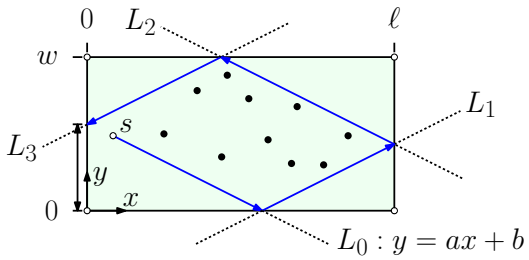


Figure 1: Billiards shot.

In addition, we need to check that it hits the proper sides. We can do this by checking that the table's lower right corner $(\ell, 0)$ is below $L_1$ and $L_2$, its upper right corner $(\ell, w)$ is above $L_1$ and $L_2$, its upper left corner $(0, w)$ is above $L_3$, and its lower left corner $(0, 0)$ is below $L_3$. For example, the test that the upper right corner is above $L_2$, we have the constraint

$$w \;\geq\; a\ell - (2\ell a + b) \;\text{ or equivalently }\; a \cdot (-\ell) - b \;\leq\; w,$$

which is clearly linear in $a$ and $b$. To ensure that the line can start to the left of the leftmost point, let $x_{\min}$ be the smallest $x$-coordinate among all the points. We add the two constraints that $L_0$ pass above $(x_{\min}, 0)$ and below $(x_{\min}, w)$. Altogether, this generates 10 additional constraints. (Some of these are redundant, and I believe that the number can be reduced, but it does not affect the asymptotic running time.)

An alternative (but equivalent) approach is based on using one line but making four copies of the points. First, we flip the billiards table about its bottom edge, and then we flip this table about its right edge, and finally we flip this table about its bottom edge (which was the

top edge of the original table). The original set of $n$ points is mapped to a set of $4n$ point, call them $P = P_0$, $P_1$, $P_2$, and $P_3$. (Formally, the transformation $T_j$ that maps $P_0$ to $P_j$ is:

$$T_1(x, y) = (x, -y), \quad T_2(x, y) = (2\ell - x, -y), \quad \text{and} \quad T_3(x, y) = (2\ell - x, y - 2w).$$

Now the problem is whether there exists a line $L : y = ax + b$ such that $P_0$ and $P_2$ lie on or above $L$ and $P_1$ and $P_3$ lie on or below $L$. As in the previous version, there are additional constraints to ensure that the line crosses the proper vertical segments.

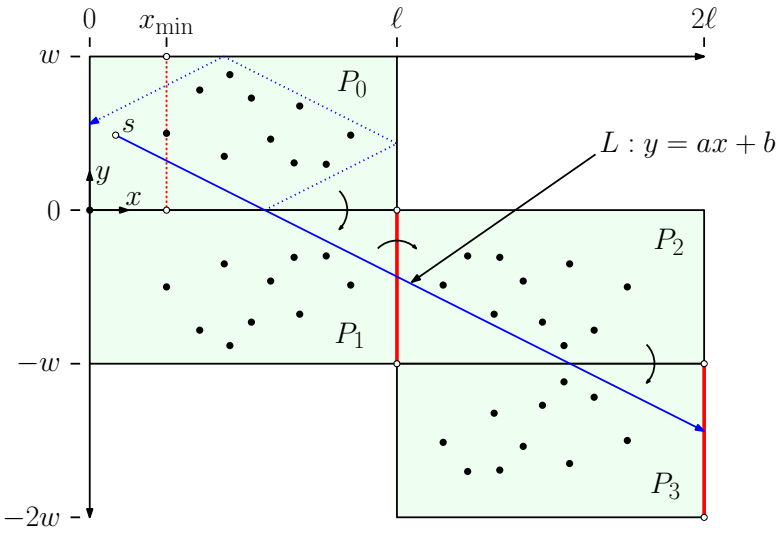


Figure 2: Billiards shot.

(c) The constraints are the same as in (a). The only change is to modify the objective function. The quantity to maximize is the height of the final piece of the shot, which is the $y$-intercept of $L_3$, $2\ell a + b + 2w$. Since $2w$ is a fixed constant, this is equivalent to maximizing $2\ell a + b$, or equivalently to maximize using the objective vector $(2\ell, 1)$.

(d) The objective function is a bit tricky to get right because the absolute value is not a linear function. From part (b), we know that the height along the left side where the shot hits is $y = 2\ell a + b + 2w$. Since the middle is at $y = w/2$, the objective is to minimize $|2\ell a + b + 2w - w/2| = |2\ell a + b + 3w/2|$. Here are two ways to handle this complication.

Our first approach is based on making two calls to an LP in $\mathbb{R}^2$, one designed to catch the best solution below the midpoint and one to catch the best solution above the midpoint:

- First, we set the objective function to maximize left-wall height just as in part (c), subject to the constraint that the final line must hit on or below the midpoint, that is, $2\ell a + b + 2w \leq w/2$. Note that the LP cannot be unbounded due to constraint (b). If it is feasible, we return this answer as the final result, since this is as close as we can get to $w/2$ from below. If infeasible, we go to the second step.
- Second, we try again, but this time we minimize the left-wall height, subject to the constraint that the final line must hit on or above the midpoint, that is, $2\ell a + b + 2w \geq$

$w/2$. As before, the LP cannot be unbounded. If it is feasible, we return this answer as the final result, since this is as close as we can get to $w/2$ from below. Otherwise, we conclude that the entire problem is infeasible.

Clearly, if there is a shot that reaches the point $w/2$ exactly, the first LP will discover this solution. Otherwise, the optimum either lies above $w/2$ and the first LP finds it. Otherwise, the optimum lies below $w/2$, and the second LP will find it. The LPs can be set up in $O(n)$ time, and they each run in $O(n)$ expected time.

Our second approach involves solving this with a single LP in $\mathbb{R}^3$. To do this, we create a new "slack variable" $t$ which intuitively represents how close we get to the desired target of $w/2$. We will run just one LP in $\mathbb{R}^3$, with the variables $(a, b, t)$. Our final value $y$-intercept must lie within $t$ units of $w/2$, that is,

$$2\ell a + b + 2w \;\leq\; \frac{w}{2} + t \quad \text{and} \quad 2\ell a + b + 2w \;\geq\; \frac{w}{2} - t.$$

These are both linear constraints in $a$, $b$, and $t$. (We think of $t$ as being nonnegative, but we do not need to add the constraint $t \geq 0$, since this is implied by these two constraints.) To get the answer that is closest to $w/2$, we run a single LP, but with the objective of minimizing $t$. If infeasible, there is no valid shot. The LP cannot be unbounded (since $t \geq 0$). If feasible, we output the solution implied by $c$ and $d$. As before the running time in expectation is $O(n)$.

**Solution 2:**

(a) Computing the leftmost and rightmost points takes $O(n)$ time. Since points are stored in the bucket according to which hull edge they overlap, when a point is inserted, we can determine the edge that lies immediately above or below it in constant time. If we are below the edge, we may ignore this point since it cannot contribute to the final upper hull.

If we are above this edge, we start walking to the right and left (in Graham-scan style) until finding the left and right tangents. The time for this operation is $O(k + 1)$, where $k$ is the number of points deleted from the hull. Thus, total time spent in this walking process over all $n$ insertions is $\sum_{i=1}^{n}(k_i + 1)$. Since each point can be deleted at most once, it follows that $\sum_{i=1}^{n} k_i \leq n$, and therefore the total sum is $O(n)$.

The other steps of the algorithm (e.g., determining whether a vertex is a point of tangency and inserting a new vertex into a given position int he hull) take only $O(1)$ time each. Thus, the total time spent in adding and removing points is $O(n)$.

(b) To analyze the total rebucketing time, fix an arbitrary point $q \in P$. For $1 \leq i \leq n$, let $p_i$ denote the $i$th point to be inserted, and let $b_i(q)$ denote the probability that $q$ has been is rebucketed after $p_i$'s insertion. We will use a backwards analysis to show that $b_i(q) \leq 2/i$.

Let $P_i = \{p_1, \ldots, p_n\}$ denote the points that have been inserted so far. This is a random subset of $P$, but since our analysis will not depend on the contents of this set, we may take it to be an arbitrary $i$-element subset of $P$ (subject to the condition that it contains the points $v_1$ and $v_n$).

After point $p_i$ is added to the hull, there are two cases. If $p_i$ does not contribute to the hull, no points are rebucketed. Otherwise, $p_i$ does contribute to the hull. In this case the

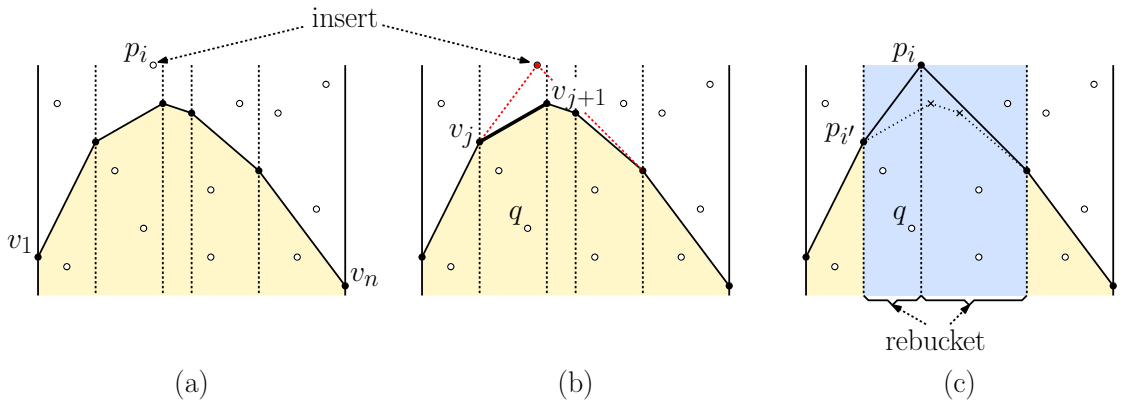


Figure 3: Incremental construction of the upper hull.

points observe that the points that have been rebucketed all lie in the buckets associated with the two edges that lie immediately to the left and right of $p_i$ (see the red shaded regions of Fig. 3(c)).

Now, consider edge of the hull $\overline{p_{i'}p_i}$ under which $q$ lies after the $i$th insertion. If either of the points $p_{i'}$ or $p_i$ were the last point to be added, then $q$ would have been rebucketed as a result of this insertion. If any of the other points of $P_i$ were to last to be added, $q$ would not be rebucketed. Ignoring the special points $v_1$ and $v_n$, every one of the $i$ points of $P_i$ are equally likely to be the last to be inserted. It follows that $b_i(q) \leq 2/i$, as desired.

Given this, the total number of times that $q$ has been rebucketed over the entire algorithm in expectation is at most

$$\sum_{i=1}^{n} b_i(q) \;\leq\; \sum_{i=1}^{n} \frac{2}{i} \;\approx\; 2\ln n \;=\; O(\log n).$$

Since $q$ is an arbitrary point of $P$, the total number of rebucketings over all the points of $P$ is $O(n \log n)$.

Note that each rebucketing can be done in $O(1)$ time per point (since it just involves determining whether $q$ lies to the left or right of $p_i$). Therefore, the total time spent rebucketing over the algorithm's entire execution is $O(n \log n)$.

**Solution 3:**

(a) To prove that polar duality is order reversing, we observe that a point $p = (c, d)$ lies inside/on/outside of line $\ell : ax + by = 1$ if and only if $ac + bd <$/$=$/$> 1$ if and only if the point $\ell^* = (a, b)$ lies inside/on/outside the line $p^* : cx + dy = 1$ (respectively).

(b) Let $\ell_1 : a_1x + b_1y = 1$ and $\ell_2 : a_2x + b_2y = 1$. The point $p = (c, d)$ lies at their common intersection if and only if

$$a_1c + b_1d = 1 \qquad \text{and} \qquad a_2c + b_2d = 1$$

which is equivalent to saying that the points $\ell_1^* = (a_1, b_1)$ and $\ell_2^* = (a_2, b_2)$ lie on the line $p^* : cx + dy = 1$. Therefore, polar duality is incidence preserving.

(c) Given a set of lines $L = \{\ell_1, \ldots, \ell_n\}$ let $K = \bigcap_{i=1}^{n} \ell_{i,\leq}$ denote the convex polygon defined by the intersection of their inner halfplanes (see Fig. 4(a)). Let $L^* = \{\ell_1^*, \ldots, \ell_n^*\}$ and define $K^*$ to be the convex hull of these dual points, that is, $K^* = \text{conv}(L^*)$ (see Fig. 4(b)).

To establish the relationship between $K$ and $K^*$, consider any point $p \in K$. The fact that $p$ lies within $K$ is equivalent to saying that $p$ lies within all the inner halfplanes of $L$, that is $p \in \ell_{i,\leq}$, for all $i$. By (a), this is dually equivalent to saying that all the points of $L^*$ lie within $p^*$'s inner halfplane, that is, $\ell_i^* \in p^*_{\leq}$, for all $i$. If a halfplane contains all the points of a set, then it contains their convex hull. We conclude that $p \in K$ if and only if $K^* \subseteq p^*_{\leq}$.

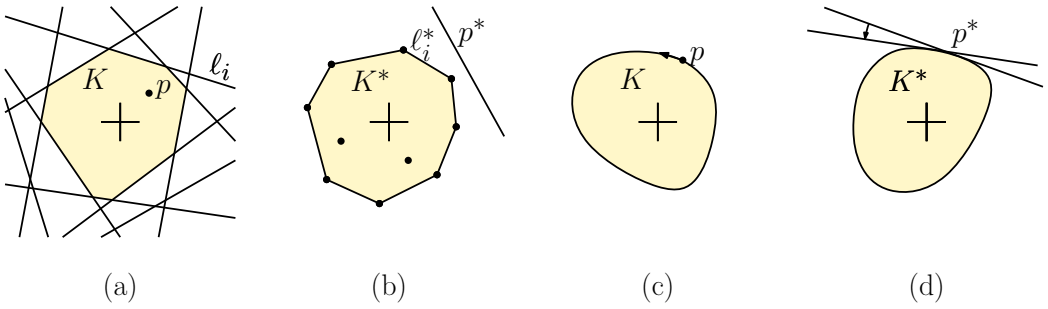


Figure 4: Support pair under the polar dual.

This implies that when a point $p = (a, b)$ travels along the boundary of $K$ in a counterclockwise order (see Fig. 4(c)), its dual $p^* : ax + by = 1$ is a support line of $K^*$ that rotates around the boundary of $K^*$ (see Fig. 4(d)). We assert that the support line also rotates in a counterclockwise direction. To see why, observe that as a point $p$ travels around the boundary of a convex body in a counterclockwise direction, its associated support line rotates in counterclockwise manner. (This is a bit easier to visualize if you imagine that $K$ is a smooth convex body, rather than a convex polygon.) It an easy exercise in geometry to see that the dual line $p^* : ax + by = 1$ is perpendicular to the vector $p = (a, b)$.[1] Thus, as a point $p = (a, b)$ travels in a counterclockwise direction around the boundary of $K$, the associated support line $p^*$ for $K^*$ rotates in a counterclockwise direction, which implies that its contact point with the boundary of $K^*$ also rotates in a counterclockwise direction (see Fig. 4(d)).

**Solution 4:** To solve the tennis problem, we will consider a solution space defined by the vector $(b, c)^\mathsf{T} \in \mathbb{R}^2$, corresponding to the coefficients of the parabola formula $y = -gx^2 + bx + c$. There constraints are

- Passes above the net along the $y$-axis: $-g \cdot 0 + b \cdot 0 + c \geq h \Longrightarrow c \geq h$.
- Passes above all the opposing player points in $P$: $-gp_{i,x}^2 + bp_{i,x} + c \geq p_{i,y}$, for $1 \leq i \leq m$. This can be expressed as a linear inequality involving $b$ and $c$ as $-p_{i,x}b - c \leq gp_{i,x}^2 - p_{i,y}$.
- Passes below all the lighting fixture points in $Q$: $-gq_{j,x}^2 + bq_{j,x} + c \leq q_{j,y}$, for $1 \leq j \leq n$. This can be expressed as a linear inequality involving $b$ and $c$ as $q_{i,x}b + c \leq -gq_{i,x}^2 + q_{i,y}$.

[1] This follows from the fact that the line passing through the vector has slope $b/a$ and $\ell$ has slope $-a/b$. The slopes of perpendicular lines are negative reciprocals of each other.

- Lands on the ground in the court. This is equivalent to checking that the $y$-coordinate of the parabola is negative at the back end of the court, that is, $-g\ell^2 + b\ell + c \leq 0$. This can be expressed as a linear inequality involving $b$ and $c$ as $b\ell + c \leq g\ell^2$.

  Note that an alternative approach would be to compute the $x$-coordinate where the parabola hits the floor (by the quadratic formula) and adding a constraint that this be less than $\ell$. Unfortunately, this yields a nonlinear inequality in $b$ and $c$, and so it would not fit within the LP structure.

The objective is to maximize clearance above the net. Since the net is at $x = 0$, this is equivalent to maximizing the $y$-intercept $c$, which is represented by the objective function associated with the vector $(0, 1)$. If the LP is infeasible, then there is no parabolic trajectory. If the LP has a feasible answer $(b, c)$, this is returned as the final answer.

It is possible for our LP formulation to be unbounded, but the associated shot descends from the heavens like a nearly vertical asteroid. The existence of such a solution suggests that the problem formulation was not very good. We could rule out such an absurd solution by requiring that the shot originate along the $x$-axis somewhere within the left side of the court, by adding a constraint that the leftward extension of the shot to the left baseline ($x = -\ell$) should be on or below the ground, that is, $-g\ell^2 - b\ell + c \leq 0$.

**Solution to the Challenge Problem:** There is a very cute solution to this problem. Starting in the upper left corner, trace a path through the box of mirrors until the beam leaves the box. Do the same, starting at the lower right corner. If these two beams never intersect, then there is no single mirror that can be added to create a successful path. Otherwise, if they do intersect, find any such intersection point and insert a mirror there of the appropriate angle. If the beams enter from the top and left or bottom and right, then the mirror angle is $45°$ and otherwise it is $-45°$ (see Fig. 5).

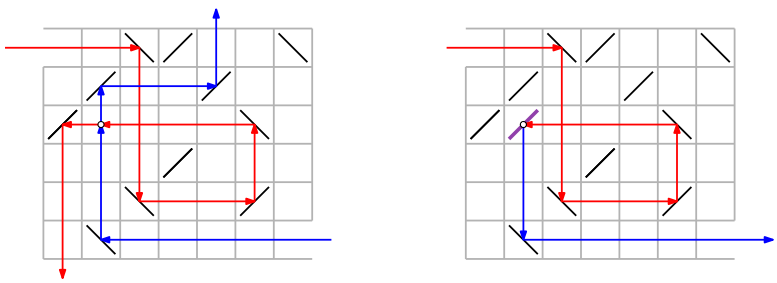


Figure 5: Laser beam solution.

The final beam consists of the following the upper-left beam in the forward direction and the lower-right beam in the reverse direction. The running time of the algorithm is dominated by the time to trace the two beams, which is $O(n \log n)$, since the beam can hit every mirror at most twice, and each ray-shooting query can be done in $O(\log n)$ time, by the data structure that was provided. Computing the intersection of the two paths can be handled by the line-segment intersection algorithm, presented in class. The algorithm can be stopped as soon as the first intersection is detected, and hence it runs in time $O(n \log n)$.

To show correctness, observe that if it is possible to insert a mirror to connect the entry and exit points with a single path, then the path can be split at this mirror into two subpaths. If we reverse the portion of the path that comes after the mirror, this is exactly the same as the path entering the box from the lower-right corner. Therefore, the two paths must intersection.

Conversely, if the two paths intersect, then first observe that the only possible intersection is one where one path is vertical and the other is horizontal. (If they were traveling in the same direction, then they would need to be oppositely directed, and this would imply that the path entering at the upper-left would eventually exit at the lower-right, which we assumed does not happen.) We can insert a mirror at this point that (subject to reversing the path entering from the lower-right corner) connects the two corners of the box.