CSC279 HW3

Hanzhang Yin

Sep/30/2023

Collaborator

Chenxi Xu, Yekai Pan, Yiling Zou, Boyi Zhang

Question 10

(PART A)

Parametrizing the Segment pq:
Let $p = (x_0, y_0)$ and $q = (x_1, y_1)$ be the endpoints of the segment pq. Any point $x(t)$ on pq can be expressed as:

$$x(t) = (1-t)p + tq = ((1-t)x_0 + tx_1, (1-t)y_0 + ty_1),\ t \in [0,]$$

Dual Lines Corresponding to Points on pq:
The dual line $\hat{x}(t)$ Corresponding to $x(t)$ is:

$$y = A(t)x - B(t)$$

where:

$$A(t) = (1-t)x_0 + tx_1$$
$$B(t) = (1-t)y_0 + ty_1$$

Now we can express rewrite the equation of $\hat{x}(t)$ as:

$$y = [x_0 + t(x_1 - x_0)]x - [y_0 + t(y_1 - y_0)]$$
$$= x_0x - y_0 + t[(x_1 - x_0)x - (y_1 - y_0)]$$

Let:

$$D = (x_1 - x_0)x - (y_1 - y_0)$$

Then:

$$y = x_0x - y_0 + tD$$

For a fixed x, y varies linearly with t from $y = x_0x - y_0$ (when $t = 0$) to $y = x_1x - y_1$ (when $t = 1$). The Union of all dual lines $\hat{x}(t)$ for $t \in [0, 1]$ is the set of all points (x, y) in the plane satisfying:

$$\min(y_0, y_1) \leq y - x \cdot \min(x_0, x_1) \leq \max(y_0, y_1)$$

Equivilantly, by eliminating parameter t, we can get similar inequality:

$$(y - x_0x + y_0)(y - x_1x + y_1) \leq 0$$

This inequality describes all points (x, y) that lie between $y = x_0x - y_0$, $y = x_1x - y_1$. which forms up a region called "double wedge".

(PART B)

```
Input:
    S = { s_i | i = 1 to n}

output:
    l_max # Line have max. intersections points with other segements

Algorithm FindMaxIntersectingLine(S):
    Initialize an empty list L_dual_lines.

    For each segement s_i in S do:
        Let p_i = (x_{0i}, y_{0i}) and q_i = (x_{1i}, y_{1i})
        Compute the dual lines:
            L_{p_i}: y = x_{0i} x - y_{0i}
            L_{q_i}: y = x_{1i} x - y_{01}
        Add L_{p_i} and L_{q_i} to L_dual_lines.

    Construct the arrangement A of the lines in L_dual_lines:
        # TIME COMPLEXITY: O(n^2logn)
        Use line sweep algorithm to compute the A.
        Store the faces, edges, and vertices of the A.

    Initialize count c_f = 0 for a starting PLANE f_0 at INFINITY

    Traverse the arrangement A to label each face with the number of double wedges
    covering it:
        # TIME COMPLEXITY: O(n^2)
        For each edge e in A:
            Determine which double wedges have boundaries along e
            For each face f adjacent to e:
                *c_{f'} is the count of the adjacent face before crossing e
                If crossing e enters a double wedge W_i, then c_f = c_{f'} + 1
                If crossing e exits a double wedge W_i, then c_f = c_{f'} - 1


    Keep track of the face f_max with the maximum count c_max during traversal

    Let (a_max, b_max) be a point inside face f_max.

    Compute the line l_max in the primal plane corresponding to (a_max, b_max):
        l_max: y = a_max x - b_max

    Return l_max
```

Question 11

Question 12

```
Input:
    RedPoints = [ (x1, y1), (x2, y2), ..., (xn, yn) ]
    BluePoints = [ (x1', y1'), (x2', y2'), ..., (xn', yn') ]

Output:
    Coefficients (a, b, c) defining the parabola y = a x^2 + b x + c
    or report "No solution exists" if impossible

Algorithm FindSeparatingParabola(RedPoints, BluePoints):
    Initialize an empty list Constraints = []

    # TIME COMPLEXITY: O(n)
    For each red point (x_i, y_i) in RedPoints do:
        Make inequality:
            a x_i^2 + b x_i + c - y_i < 0
        Convert to standard LP form (inequalities with <=):
            a x_i^2 + b x_i + c - y_i <= -e
        Add this to Constraints

    # TIME COMPLEXITY: O(n)
    For each blue point (x_j, y_j) in RedPoints do:
        Make inequality:
            a x_j^2 + b x_j + c - y_j > 0
        Convert to standard LP form (inequalities with <=):
            -a x_j^2 - b x_j - c + y_j <= -e
        Add this to Constraints

    ObjectiveFunction: Minimize 0

    # TIME COMPLEXITY: O(n)
    Solution = LinearProgramming(Constraints, ObjectiveFunction)

    If Solution is feasible then:
       Output "Parabola found with coefficients:"
       Output "a =", Solution.a
       Output "b =", Solution.b
       Output "c =", Solution.c

    Output "No solution exists"
```