Homework 3

Question 1 part a

According to the given definition of solvent-accessibility, a protein is solvent-accessible if there is an placement of a water molecule that touches the protein and does not overlap other proteins. Distance between the protein and the water molecules is at least <= to r + r’.

Given the above definition, in a protein’s Voronoi cell, there are 2 position types of the water molecule that satisfies the solvent-accessibility:

* Type 1: Any place except the cluster center.
* Type 2: The center of the Voronoi cell.

In type 1, there are undefined number of water molecules that satisfies the solvent-accessibility. Hence, it doesn’t guarantee that there exists at least one placement of a water molecule to make the protein solvent-accessible.

In type 2, if the center is the water molecule satisfying the solvent-accessibility,

* Proteins which are closet (touching and surrounding the center) to the center solvent-accessible according to the above definition. The water-molecule centers of other cells do not overlap with proteins in the current cell according to the Voronoi properties.
* Proteins (not touching the center or on the edge) are solvent-accessible since the distance between these proteins to the center is larger than r + r’. According to Voronoi properties, the water-molecule centers of other cells do not overlap with these proteins.

Hence, a protein is solvent-accessible if and only if there exists a water molecule that is the center of the protein’s Voronoi cell.

Question 1 part b

Based on the part a, any protein is solvent-accessible if and only if we can identify that the water-molecule placement at the center of the protein’s Voronoi cell satisfies the solvent-accessibility. In other words, proteins are solvent-accessible if they are closer to a site (water-molecule) than other sites. This problem can be treated as nearest-neighbor-searching using kd-tree that constructs the kd-tree to partition solvent-accessible proteins into Voronoi cells, it takes O(nlogn) time that n is the number of proteins.

Question 2 part a

Say that every top segment endpoint is ai\* and every bottom segment endpoint is bi\*.

By duality transformation, we want to find a dual point l\* that lies above every dual line ai\*and lies below every dual line bi\*. Or the dual point l\* lays in every double wedge si\* shaped by each point si\* (duality transformed from each segment s) and every corresponding pair of dual lines ai\* and bi\*. Specifically, the dual point l\* lays in the union double wedge s-union of all double wedges si\*. Please see the image below.

This problem can be treated as the maximum stabbing line since the line l intersects all given segments. Hence, the dual point l\* should be the intersection of every pair of dual lines ai\* and bi\*. Or the dual point l\* lies right at point m\* (in the image below).

Diagram

Description automatically generated

Image reference: [here](https://cw.fel.cvut.cz/b181/_media/courses/cg/lectures/12-duality.pdf)

Question 2 part b

As mentioned earlier, we can treat this problem as the maximum stabbing line since we seek for a non-vertical line that intersects all line segments.

By applying the dual transformation (part a), the maximum stabbing line is the dual point l\* laying right at the intersection point m\* of all dual lines ai\* and bi\*. Therefore, by duality, we can solve the maximum stabbing line by identifying the intersection point m\*. Hence, we can apply the plane-sweep algorithm for the line segment intersection to find m\*.

Say, we pick two segment endpoints for dual lines ai\* and bi\* which are different from the intersection point m\*. Hence, we have a set of line segments laying on dual lines ai\* and bi\*. Note that there are mi\* that are intersection points of other line segments. Note that all line segments in the set share a single intersection point m\*.

First, we have to build the line segments by the constructing a line arrangement in O(n^2).

Instead of naively applying the original plane-sweep algorithm, the algorithm is modified such that:

* Our algorithm shares the same data structures with the original plane-sweep algorithm.
* The coordinate status f the common intersection point m\* is stored along the run. Initially, the coordinates are empty.
* Pseudo-code:
  + Insert all of the endpoints of the line segments of S into the event queue. The initial sweep-line status is empty.
  + While the event queue is nonempty, extract the next event in the queue. There are three cases, depending on the type of event:
    - Left endpoint: all actions for left endpoints in our algorithm are shared with the original algorithm.
    - Right endpoint: all actions for right endpoints in our algorithm are shared with the original algorithm.
    - Intersection:
      * Let s+ and s− be the two segments involved (with s+ above just prior to the intersection). Report this intersection. If the coordinates of this intersection are different from the coordinate status, replace the coordinate status with the new one. If the coordinate status is empty, assign the intersection.
      * Let s++ and s−− be the segments immediately above and below the intersection. Remove any event involving the pair (s+, s++) and the pair (s−, s−−).
      * Swap s+ and s− in the sweep-line status (they must be adjacent to each other).
      * Test for an intersection between s− and s++ to the right of the sweep line, and if so, add it to the event queue. Do the same for s+ and s−−.
  + At the end, when the event queue is empty, the last updated coordinate status (a, b) is the intersection point m\* that represents the dual point l\* or the maximum stabbing line y = ax – b.

Since our algorithm is closely similar to the plane-sweep algorithm except tracking the coordinate status of the intersection point m\*. As stated, the (topological) plane-sweep algorithm given the constructed line arrangement takes O(n^2). Hence, the total running time is O(2n^2) = O(n^2)