CS603 Report

Hari Prakash Reddy 210050119
Gasada Sree Laxmi 210050052

April 2025

1 Maximum subset of collinear points in a set of N points

Input

A set of n points in $\mathbb{R}^2$.

Output

The largest subset of points in P that lie on a common line

Identifying collinear points among a large set is a classic computational geometry problem. Given a set of n distinct points in the plane, the objective is to determine the maximum number of points that lie on the same straight line. While trivial for small datasets, a brute-force solution scales poorly with input size, requiring $O(n^3)$ time in the worst case.

To address this, we transform the problem using geometric duality, enabling a more elegant and scalable solution. We leverage sweep line algorithms and splay trees to handle dynamic segment intersection detection efficiently.

Point-Line Duality

Point-line duality is a mathematical transformation that maps:

- A point (a, b) in the primal plane to the line $y = ax - b$ in the dual plane.
- A non-vertical line $y = mx + c$ in the primal plane to the point $(m, -c)$ in the dual plane.

Under this transformation:

- Collinear points map to lines that intersect at a common point in the dual space.
- The problem of finding a maximal subset of collinear points becomes one of finding a point where the most number of dual lines intersect.

Algorithm Overview

The high-level steps of our algorithm are:

1. Convert each input point to its dual line.
2. Clip the dual lines to a bounding box to limit the domain.
3. Use a sweep line algorithm to detect intersections between dual lines.
4. Maintain active segments using a balanced binary search tree (splay tree).
5. Count how many dual lines pass through each intersection point.
6. Report the dual intersection with the highest multiplicity, then convert it back to the primal line.

Data Structures Used

- Splay Tree
  A self-adjusting binary search tree used to maintain the vertical order of active line segments during the sweep. It supports:
  - `insert(segment)`
  - `delete(segment)`
  - `find predecessor/successor`
  - All operations in amortized $O(\log n)$ time.
- Priority Queue (Heap)
  Used to maintain the event queue ordered by x-coordinate:
  - Segment start points
  - Segment end points
  - Intersection points

Implementation Details

The code is organized in two main modules: `dual.py` and `q1.py`.

- dual.py
  This module handles the point-to-line duality transformation and envelope computation.
  - `dual_line(pt)`: Transforms a point to its dual line.
  - `compute_envelope(lines, is_upper)`: Recursively computes upper or lower envelope using divide-and-conquer.
  - `clip_line(a, b, bbox)`: Clips infinite dual lines to bounding box for visualization.

- `compute_envelope_intersections(env)`: Calculates intersection points along the envelope.

- q1.py
  This module runs the sweep line algorithm.
  - `Segment` class represents a clipped dual line with endpoints.
  - `SweepLine` manages active segments.
  - Events include segment endpoints and intersections.
  - The splay tree is updated dynamically as the sweep line moves.
  - For every intersection, we track how many segments pass through it.

Results

The output includes a subset of points with max size that lie on a common line

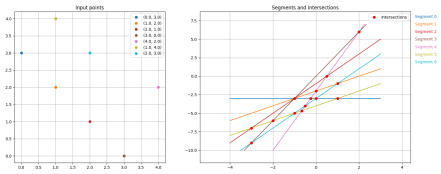

Figure 1

The above figure contains 2 plots, left plot contains the given input. Right plot shows the bounded line segments and the intersections

The approach scales significantly better than the brute-force method and works efficiently even for inputs with large set of points.

Conclusion

This algorithm is for detecting the largest set of collinear points in the plane using duality and sweep line techniques. The transformation of the problem to the dual space allows for efficient intersection detection using a splay tree and event-driven processing. The result is an elegant and fast solution to a classically expensive geometric problem.

2 Computing triangle of minimum area in a set of N points

Input

A set of n points in $\mathbb{R}^2$.

Output

Three points $(p_1, p_2, p_3) \in P$ such that the triangle formed has the minimum non-zero area.

Approach Overview

The solution utilizes the concept of geometric duality to reduce the problem into line intersections in the dual space, where events are handled in an order-sweep manner.

- Each point $(x, y) \in \mathbb{R}^2$ is mapped to a line $y = ax + b$, with $a = x$, $b = -y$.
- We track intersections of these lines and maintain their vertical order using a doubly linked list.
- For each intersection (event), we update the order of the lines and examine adjacent triples to evaluate potential minimum-area triangles.

Algorithm Design

1. Transform each point $(x, y) \in P$ into its dual line representation.
2. Compute all line intersections and sort them by the x-coordinate (sweep line).
3. Maintain a linked list representing the order of lines at a far-left x-value.
4. At each intersection event, swap the lines involved and check triangles formed by the corresponding primal points and their immediate neighbors.
5. Use the standard area formula for a triangle given three points:

$$\text{Area} = \frac{1}{2} \left| x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \right|$$

6. Keep track of the triangle with the smallest non-zero area.

Data Structures

- `Point` and `Line` structs to represent coordinates and their dual.
- A doubly linked list to maintain dynamic order of lines.
- A set of x-events storing intersection points and involved line indices.

Time Complexity

The algorithm runs in $\mathcal{O}(n^2 \log n)$ time:

- $\mathcal{O}(n^2)$ for computing all line intersections.
- $\mathcal{O}(n^2 \log n)$ for sorting the intersections and maintaining the line order.

Results

After processing all intersection events and updating the line order dynamically, the algorithm outputs a set of three points that form the triangle of minimum area.

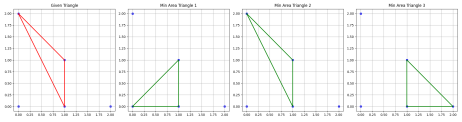

Figure 2

In the above figure, Red triangle is the result we obtained using the algorithm, Green triangles are all possible triangles with minimum area in the given set of points

Conclusion

By leveraging geometric duality and a sweep-line strategy in the dual space, the algorithm efficiently computes the minimum-area triangle in $\mathcal{O}(n^2 \log n)$ time, avoiding the naive $\mathcal{O}(n^3)$ brute-force solution.