CLOSEST PAIR OF POINTS

applied to geographical information systems

CS315

Dr.Renad Alsweed

INTRODUCTION

The Closest Pair of Points is a key problem in computational geometry used in GIS to find the nearest two locations. Its importance comes from the need to compute distances efficiently and use optimized algorithms and data structures, showing how theory improves real-world performance with large datasets.

This project compares two algorithms for finding the closest pair of points: the simple Brute Force method to show the basic idea, and the faster Plane Sweep algorithm for an optimized solution.

RELEVANCE TO REAL-WORLD APPLICATIONS

Applying the Closest Pair of Points algorithm in a GIS environment opens the door to multiple real-life applications:

- Emergency response optimization
- Delivery and transportation services
- Urban planning and infrastructure design
- Sensor networks and environmental monitoring

EXPERIMENTAL EXAMPLE

Input: A set of coordinates representing different locations.
Example: (2, 3), (5, 4), (3, 1), (12, 30), (40, 50)

Process: The algorithm computes the Euclidean distance between all pairs of points to identify the smallest one.

Output: The two points that are closest to each other and their distance.

Example result:
Closest pair → (2, 3) and (3, 1), Minimum distance → 2.23

ALGORITHMS

NAIVE ALGORITHM (BURT FORCE)

How It Works :

- Compares every point with all others using a double loop.
- Computes Euclidean distances and keeps the smallest value.
- Always finds the correct closest pair.
- Simple and requires no advanced data structures.

NAIVE ALGORITHM PSEUDOCODE

```
Algorithm NaiveClosestPair(P)
if N = 0 or N = 1 then
 return (∞, (null, null))
endif

d ← ∞
closestPair ← (null, null)
for i ← 1 to N - 1 do

 for j ← i + 1 to N do

 dx ← P[i].x - P[j].x
 dy ← P[i].y - P[j].y
 dist ← sqrt(dx*dx + dy*dy)
 if dist < d then
 d ← dist
 closestPair ← (P[i], P[j])
 endif

 endfor
endfor
return (d, closestPair)
```

OPTIMIZED ALGORITHM (PLANE SWEEP)

How It Works :

- Sort points by x-coordinate.
- Sweep a vertical line across the plane.
- Keep only points within distance d in an active set.
- Check nearby candidates in a narrow strip.
- Update the minimum distance when a closer pair appears.

OPTIMIZED ALGORITHM PSEUDOCODE

```
Algorithm PlaneSweepClosestPair(xP)
if N ⩽ 1 then
 return (∞, null)
endif
d ← ∞
closest ← null
Y ← empty balanced binary search tree (ordered by y)
leftIndex ← 1
for rightIndex from 1 to N do
 p ← xP[rightIndex]
 while (leftIndex < rightIndex) AND
 (xP[leftIndex].x ⩽ p.x - d) do
 remove xP[leftIndex] from Y
 leftIndex ← leftIndex + 1
 endwhile
 for each q in Y such that
 p.y - d < q.y < p.y + d do
 dist ← distance(p, q)
 if dist < d then
 d ← dist
 closest ← (p, q)
 endif
 endfor
 insert p into Y
endfor
return (d, closest)
```

IMPLEMENTATION

IMPLEMENTATION OF NAIVE ALGORITHM

Description: Compares every point with every other point to find the closest pair.
Purpose: Simple approach and correctness check.
Inputs: Set of points P
Output: Closest pair and distance d_{min}

Steps:

- Double loop over all point pairs (i, j)
- Compute Euclidean distance for each pair
- Update d_{min} if distance is smaller
- Return the pair with d_{min}

IMPLEMENTATION OF OPTIMIZED ALGORITHM

Description: Sorts points and sweeps a vertical line with an active window to reduce comparisons.
Purpose: Faster, scalable solution for large datasets.
Inputs: Set of points P
Output: Closest pair and distance d_{min}

Steps:

- Sort points by X
- Sweep a vertical line left to right, maintaining an active set of points within current distance d
- Keep active set sorted by Y (BST or sorted list)
- For each new point, check only nearby points in active set (usually $\leqslant 6$)
- Update d_{min} and closest pair if smaller distance found

THEORETICAL ANALYSIS

TIME & SPACE COMPLEXITY

Naive algorithm

Time O(n^2)
Space O(1)

Optimized algorithm

Time O(n log n)
Space O(n)

NAIVE ALGORITHM

ADVANTAGES

- Very easy to understand and implement.
- Works well for small datasets.
- Useful as a baseline for comparing faster algorithms.

DISADVANTAGES

- Slow for large datasets due to $O(n^2)$ complexity.
- Not scalable for big GIS data.
- Inefficient when the number of points becomes large.

OPTIMIZED ALGORITHM

ADVANTAGES

- Much faster than brute force.
- O(n log n) time.
- Scales well for large GIS datasets.
- Minimizes unnecessary comparisons.

DISADVANTAGES

- More complex to code.
- Requires extra data structures.
- Harder to debug if the active set is mishandled.

EMPIRICAL ANALYSIS

TEST ENVIRONMENT :

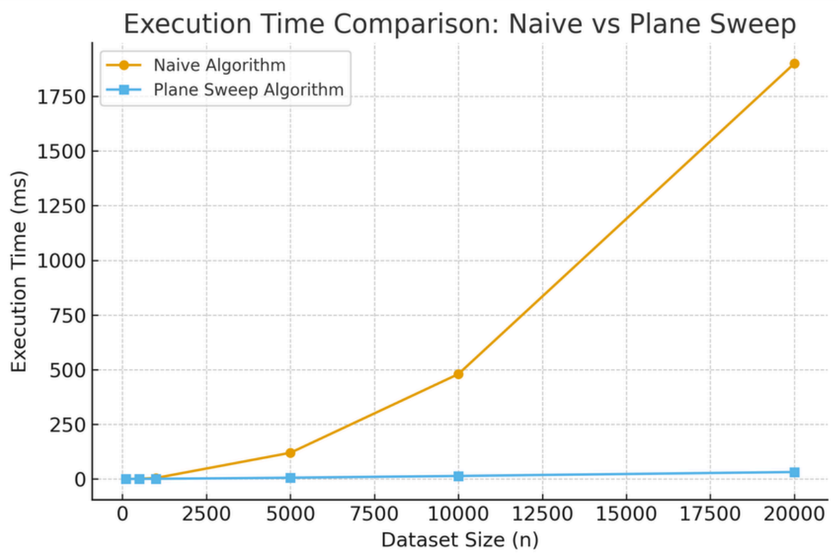
Python was used to implement both the Naive and the Optimized (Plane Sweep) algorithms.
Their performance was measured using multiple point datasets (e.g., 100, 500, and 1,000 points).

TEST SCENARIO :

Each algorithm was evaluated by:

- Measuring execution time across different dataset sizes.
- Examining how accurately and efficiently each algorithm detects the closest pair of points.
- Observing memory usage during execution, especially for large datasets.

EXECUTION TIME :



Number of points (n)	Naive Time (ms)	Plane Sweep Time (ms)
100	0.05	0.20
500	1.20	0.50
1,000	4.80	0.90
5,000	120	6.00
10,000	480	14.00
20,000	1,900	32.00

	NAIVE ALGORITHIM	OPTIMIZED ALGORITHIM
Approach :	Compares every point with every other point directly Uses a double nested loop	Sorts the points and sweeps a vertical line, checking only nearby point using balanced BST
Accuracy :	Exact 100% Always finds the true closest pair because it checks all possible pairs.	Exact 100% Same as naive algorithm but faster
Use case :	Small database	Medium to large database

FINAL THOUGHTS

In conclusion, this project demonstrated how selecting the appropriate algorithm depends heavily on the dataset size and real-world requirements.

The Naive algorithm is simple and fully accurate for small inputs, but becomes slow as the number of points increases.
In contrast, the Optimized Plane Sweep algorithm scales efficiently and maintains high performance with large datasets, making it ideal for real-world geometric applications.

THANK YOU