Dhirubhai Ambani University

IT 584
Approximation Algorithm
Minimum Weight Triangulation

Prof. Rachit Chhaya

Prepared By – Group Members

Nirva Patel – 202201071

Vidhi Dhanani – 202201076

Kashvi Bhanderi – 202201149

Gunjan Saroliya – 202201225

Aastha Bhavsar – 202201259

Contents

1 Introduction

Triangulation is a fundamental problem in computational geometry with important applications in computer graphics, geographical information systems, mesh generation, and finite element analysis.
A notable variant of this problem is the Minimum Weight Triangulation (MWT), which aims to connect a set of n points in the plane with non-overlapping edges to form a valid triangulation, such that the total length of all edges used in the triangulation is minimized. Here, the weight of an individual edge is defined by its Euclidean length, and the triangulation must cover all points with a set of triangles whose combined edges have the least possible total length.

2 Mathematical Formulation

Given a set $P = \{p_1, p_2, \ldots, p_n\}$ of n distinct points in the Euclidean plane such that no three points are collinear. The goal is to construct a triangulation of the convex hull P such that we get the total weight of the triangulation is minimum.

Triangulation means connecting points in a set P into straight line so that entire shape get divided in triangle and no lines are crossing each other. These triangles use only the given points as corners, and the outer boundary forms the shape's convex hull. Our goal is to make as many triangular as possible with minimum weight and without crossing each lines.

The weight of a triangulation is typically defined as the sum of the Euclidean lengths of all the edges used in the triangulation:

$$\text{Weight}(T) = \sum_{e \in T} |e|$$

Objective:

P is set of n distinct points defined as:

$$P = \{p_1, p_2, \ldots, p_n\} \quad \text{where each } p_i = (x_i, y_i) \in \mathbb{R}^2$$

Triangulation T is a set of $(2n - h - 2)$ non-intersecting triangles:(where h is the size of convex hull)

$$T = \{\Delta_{i,j,k} \mid i \leq j < k \leq n \text{ and } \Delta_{i,j,k} \text{ is valid triangle.}\}$$

Euclidean distance is defined as:

$$d(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

The weight $w(\Delta_{i,j,k})$ of a triangle $\Delta_{i,j,k}$ is defined as:

$$w(\Delta_{i,j,k}) = d(v_i, v_j) + d(v_j, v_k) + d(v_k, v_i)$$

Then, the cost of $C(T)$ is:

$$c(T) = \sum_{\Delta_{i,j,k} \in T} w(\Delta_{i,j,k})$$

Then our objective function become:

$$\text{Minimize} \sum_{e \in T} |e|$$

Constraints:

- All vertices of the triangle must be points from P.
- No three points in a set P are collinear.
- The triangulation must cover the convex hull of P.
- No two edges in the triangle can intersect.

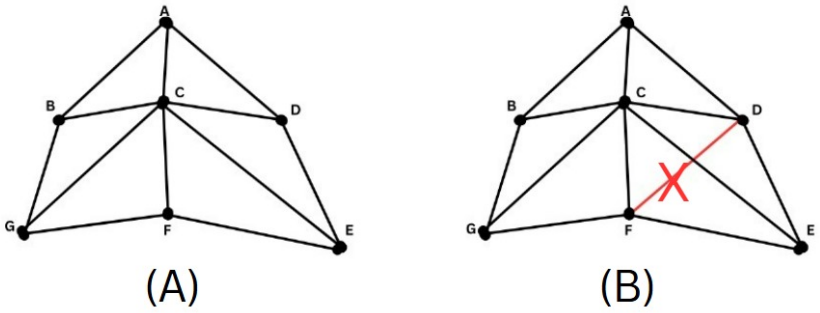


Figure 1: This figure shows the valid edge

3 Why NP-Hard?

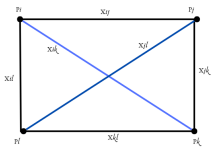
Let P be a set of points in the plane.

Goal: Triangulate P using non-intersecting diagonals.

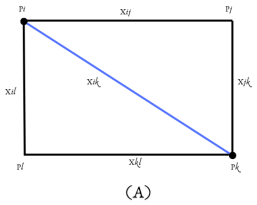
Objective: Reduce triangulation to 3-SAT (to show NP-completeness). We represent a valid triangulation using Boolean variables and enforce conditions of non-intersection as a 3-SAT problem.

Let x_{ij} be a variable representing an edge between every two possible vertices (P_i, P_j). Then:

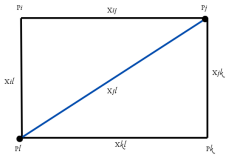
$$x_{ij} = \begin{cases} 1 & \text{if the edge } (P_i, P_j) \text{ is used in the triangulation} \\ 0 & \text{otherwise} \end{cases}$$



(a) internal edges for triangularization



(b) Possible Triangularization



(c) Possible Triangularization

Figure 2

For triangulation, we need to choose any one of the diagonals in the quadrilateral.

Now, enforce the condition of MWT (Minimum Weight Triangulation):

No diagonals (edges) should intersect, i.e., it means we get non-overlapping triangles.

Let us create a boolean expression that satisfies our constraints.

- We need to find all the possible assignments of the boolean expression such that it generates successful triangulations.
- For non-intersection: $\neg X_{ik} \vee \neg X_{jm}$

The truth value of this gives us a valid triangulation.

For $n = 4$

$$(\neg x_{ij} \wedge \neg x_{km}) \vee (\neg x_{ik} \wedge \neg x_{jm}) \vee (\neg x_{jm} \wedge \neg\ x_{ij})$$

Ensures that the truth values gives us valid triangulation.

Hence, it reduces to 3-SAT, a known NP-hard problem.

Thus, MWT is NP-Hard.

We extend this explanation as general n, with an additional constraint of metric rules which gives us all possible triangulations.

$$\Rightarrow \text{MWT is at least as hard as 3-SAT.}$$

4 Exponential Time Algorithm

4.1 Algorithm:

The brute-force algorithm for minimum weight triangulation follows these steps:

1. **Generate all triangulations:**

$$\mathcal{T}(P) = \{T_1, T_2, \ldots, T_k\} \quad \text{(all valid triangulations of } P\text{)}$$

2. **Compute total weight for each triangulation T_i:**

$$W(T_i) = \sum_{e \in T_i} |e|$$

3. **Select triangulation with minimum total weight:**

$$T^* = \arg\min_{1 \le i \le k} W(T_i)$$

4.2 Complexity Analysis:

The number of possible triangulations of n points of a polygon is given by C_{n-2}, which grows exponentially. For each triangulation, computing its weight takes $O(n)$ time. The number of triangulations grows exponentially for n a set of random points, and although the exact number depends on the arrangement of points, it is known to be asymptotically $O(4^n)$.

5 Approximation Algorithms

5.1 Greedy Algorithm

Algorithm: Greedy Triangulation of a Planar Point Set

Input: A set P of n points in the 2D plane.
Output: A triangulation T and the total length L of edges used.
Steps:

1. Compute all edges between every pair of points in P.
2. For each edge, calculate its Euclidean length.
3. Sort all edges in non-decreasing order of length.
4. Initialize:
   - An empty set E of edges.
   - An empty set T of triangles.
5. Compute the convex hull of P and let h be the number of convex hull points.
6. Set $t \leftarrow 0$ *// Triangle counter.*
7. For each edge e in the sorted edge list:
   (a) If e crosses any edge in E, skip this edge.
   (b) Else:
      i. Add e to E.
      ii. For each pair of existing edges in E that can form a triangle with e:
         A. If the triangle does not overlap existing triangles:
         B. Add the triangle to T.
         C. Set $t \leftarrow t + 1$.
         D. If $t = 2n - h - 2$, break all loops.
8. End For.
9. Compute the total length L of all edges used in T.
10. **Output:** The triangulation T and total length L.

5.1.1 Time Complexity Analysis

- Generating all edges: $O(n^2)$
- Sorting edges: $O(n^2 \log n^2)$
- For each of $O(n^2)$ edges, checking for intersection with up to $O(n^2)$ existing edges: $O(n^4)$ in worst case

Total Time Complexity:

$$O(n^2 \log n^2 + n^4) = O(n^4)$$

5.1.2 Implementation

[Code link]

5.1.3 Illustrative Example of Greedy Algorithm

Points:

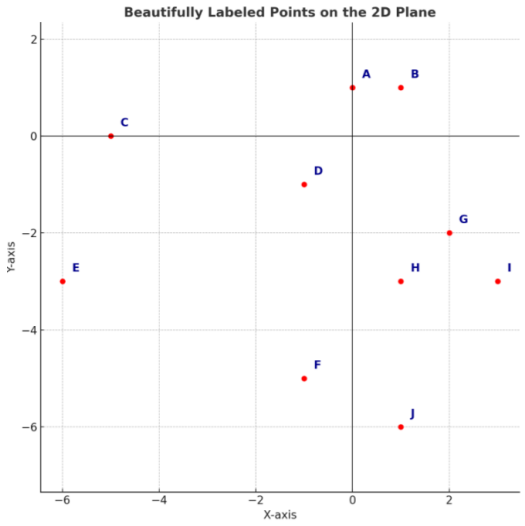
- A (0, 1), B (1, 1), C (-5, 0), D (-1, -1), E (-6, -3)
- F (-1, -5), G (2, -2), H (1, -3), I (3, -3), J (1, -6)

All Possible Diagonals and Their Euclidean Distances

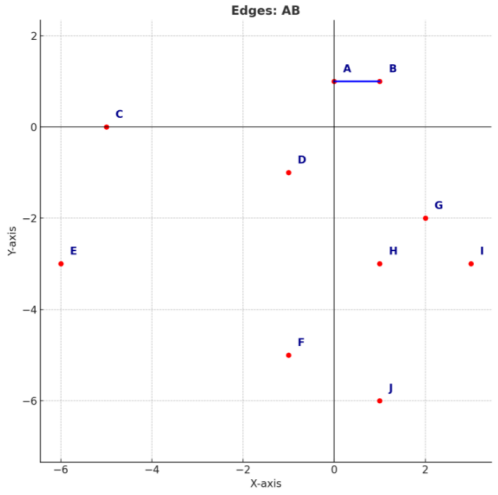
AB = 1	BC = 6.08	CE = 3.16	DH = 2.82	FH = 2.82
AC = 5.099	BD = 2.82	CF = 6.40	DI = 4.47	FI = 4.47
AD = 2.23	BE = 8.06	CG = 7.28	DJ = 5.38	FJ = 2.23
AE = 7.21	BF = 6.32	CH = 6.70	EF = 5.38	GH = 1.41
AF = 6.08	BG = 3.16	CI = 8.54	EG = 8.06	GI = 1.41
AG = 3.60	BH = 4	CJ = 8.48	EH = 7	GJ = 4.12
AH = 4.12	BI = 4.47	DE = 5.38	EI = 9	HI = 2
AI = 5	BJ = 7	DF = 4	EJ = 7.61	HJ = 3
AJ = 7.07	CD = 4.12	DG = 3.16	FG = 4.24	IJ = 3.60

Sorted Edges Based on Euclidean Distances

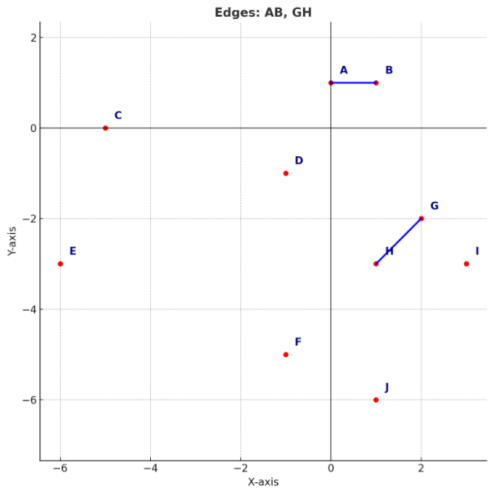
AB = 1	DH = 2.82	DF = 4	AI = 5	CF = 6.40	BE = 8.06
GH = 1.41	HJ = 3	CD = 4.12	AC = 5.099	CH = 6.70	EG = 8.06
GI = 1.41	BG = 3.16	GJ = 4.12	DE = 5.38	BJ = 7	CJ = 8.48
HI = 2	CE = 3.16	AH = 4.12	EF = 5.38	EH = 7	CI = 8.54
AD = 2.23	DG = 3.16	FG = 4.24	DJ = 5.38	AJ = 7.07	EI = 9
FJ = 2.23	AG = 3.60	BI = 4.47	AF = 6.08	AE = 7.21	
BD = 2.82	IJ = 3.60	DI = 4.47	BC = 6.08	CG = 7.28	
FH = 2.82	BH = 4	FI = 4.47	BF = 6.32	EJ = 7.61	



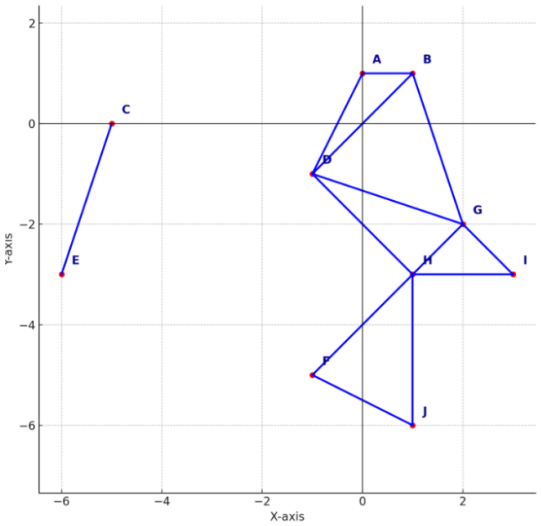
(a) Set of n points in 2D plane



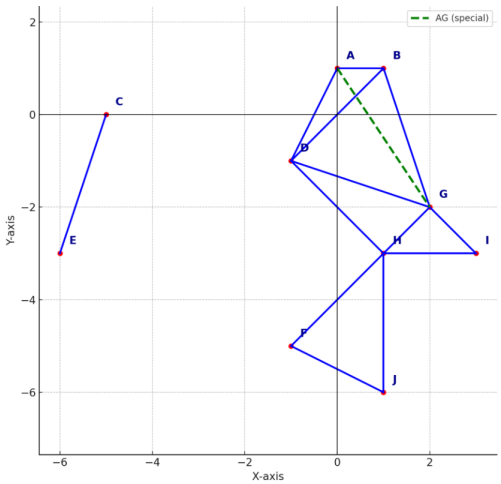
(b) 1st iteration connecting shortest edge AB



(c) 2nd iteration



(d) After Several Iteration



We cannot include the edge AG because it intersects with other existing edges. The final triangulation is obtained using a greedy method. The total number of triangles

is given by the formula:

$$2 \times n - h - 2,$$

where n is the total number of points and h is the number of points on the convex hull.

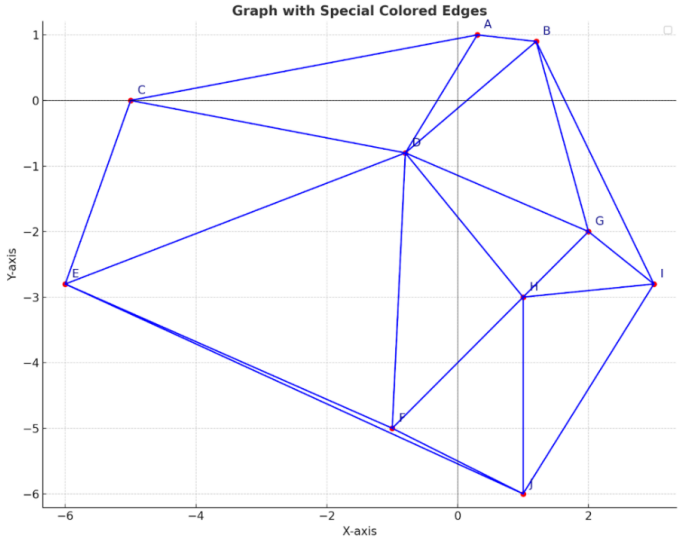


Figure 4: Final Tringularization.

Total weight is 70.9586

```
10
Enter the coordinates (x y) of each point:
0 1
1 1
-5 0
-1 -1
-6 -3
-1 -5
2 -2
1 -3
3-3
1 -6
Triangles formed:
(6, 7, 8)
(0, 1, 3)
(5, 7, 9)
(1, 3, 6)
(3, 6, 7)
(7, 8, 9)
(3, 5, 7)
(1, 6, 8)
(0, 2, 3)
(2, 3, 4)
(3, 4, 5)
(4, 5, 9)
Total weight: 70.9586
```

Approximation Guarantee of the Greedy Triangulation Algorithm

In this section, we analyze the worst-case approximation ratio of the greedy triangulation algorithm. Let S be a set of n points in general position (no three collinear), and let:

- GT(S): the triangulation produced by the greedy algorithm

- $\mathrm{MT}(S)$: a minimum weight triangulation (MWT) of S
- $|\cdot|$: denotes the total Euclidean length of all edges in a triangulation.

Our goal is to bound the approximation ratio:

$$\frac{|\mathrm{GT}(S)|}{|\mathrm{MT}(S)|}$$

The total length of the greedy triangulation is at most a linear factor worse than the minimum weight triangulation. Specifically, for any set S of n points in general position,

$$|\mathrm{GT}(S)| \leq c \cdot n \cdot |\mathrm{MT}(S)| \quad \text{for some constant } c > 1$$

5.1.4 Claim

An upper bound showing that greedy triangulation is at most $O(n)$ times worse than optimal:

$$|\mathrm{GT}(S)| \leq O(n) \cdot |\mathrm{MT}(S)|$$

Let:

- $\mathrm{GC}(S)$: a greedy convex partition, a subgraph of $\mathrm{GT}(S)$ formed by selecting up to 3 edges (called spokes) at each vertex to form a convex subdivision.
- $\mathrm{MC}(S)$: the minimum weight convex partition, i.e., the shortest subdivision of the convex hull into convex cells.

5.1.5 Lemma 1 (Convex Partition Bound)

$$\frac{|\mathrm{GC}(S)|}{|\mathrm{MC}(S)|} = O(n)$$

5.1.6 Lemma 2 (Greedy Triangulation via Partition)

There exists a constant $r > 0$ such that:

$$|\mathrm{GT}(S)| \leq O(r \cdot |\mathrm{GC}(S)| + r \cdot |\mathrm{MT}(S)|)$$

Explanation: The greedy triangulation can be viewed as a refinement of the convex partition, and its length is bounded by the sum of:

- the length of the convex partition, and
- the extra diagonals needed to triangulate each convex cell.

5.1.7 Combine Lemmas

From Lemma 1 and Lemma 2:

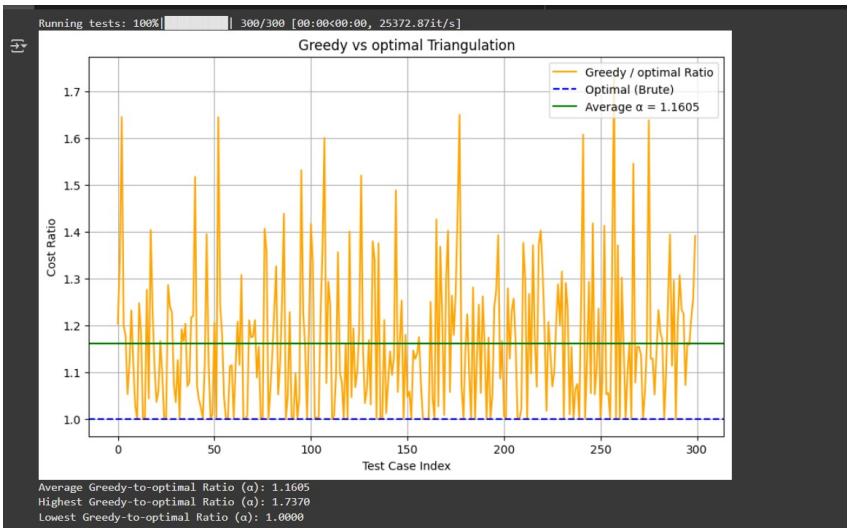
$$|\mathrm{GT}(S)| \leq O(n \cdot |\mathrm{MC}(S)|) + O(|\mathrm{MT}(S)|)$$

But $|\mathrm{MC}(S)| \leq |\mathrm{MT}(S)|$ since the minimum weight triangulation is also a convex partition (with more edges). So:

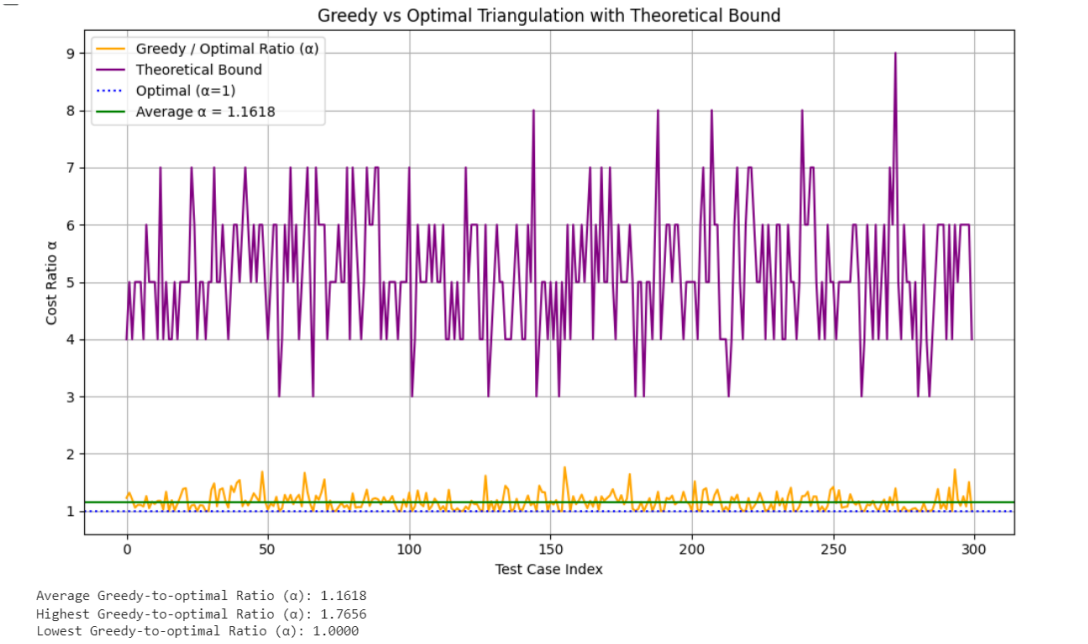
$$|\mathrm{GT}(S)| \leq O(n) \cdot |\mathrm{MT}(S)|$$

5.1.8 Conclusion:

So our claim is true. The approximation factor for the greedy algorithm of minimum weight triangulation is $O(n)$.



Theoretical vs. Practical Performance Comparison



The greedy triangulation algorithm has a known theoretical worst-case approximation factor of $O(n)$, which implies that for certain pathological point sets, the total weight of the triangulation can be linear times worse than the optimal. This makes the greedy method theoretically weak, especially when n becomes large, potentially leading to insufficient solution quality in critical applications.

However, in our practical experiments across diverse datasets, the observed approximation factor $\alpha = \frac{w(T_{\text{greedy}})}{w(T^*)}$ remains significantly better than the worst-case bound. Empirically, α is often close to 1.1–1.5, even for large values of n, indicating that the greedy algorithm typically produces solutions that are near-optimal in practice.

Thus, although the greedy approach lacks a strong worst-case theoretical guarantee, its simplicity, efficiency, and consistently good empirical performance make it a competitive heuristic in real-world scenarios. Nevertheless, for applications requiring provable guarantees, algorithms with tighter approximation bounds (e.g., $O(\log n)$) may be preferable.

5.2 Dynamic Programming

Algorithm: Triangulation with Convex Hull and Interior Point Insertion

Input: A list of points on a 2D plane
Output: List of all triangles formed and total sum of perimeters of these triangles
Steps:

1. Compute the Convex Hull using Graham's scan or Andrew's monotone chain.
2. Separate interior points (points not part of the convex hull).
3. Triangulate the Convex Hull:
   (a) Let n be the number of points on the convex hull.
   (b) Initialize $dp[n][n] \leftarrow \infty$ for all i, j.
   (c) For gaps g from 2 to $n-1$:
      i. For i from 0 to $n-g-1$:
         A. Set $j \leftarrow i+g$.
         B. For k from $i+1$ to $j-1$:
         C. Compute cost:

$$\text{cost} = dp[i][k] + dp[k][j] + \text{Perimeter}(i, k, j)$$

         D. If cost $< dp[i][j]$, set $dp[i][j] \leftarrow$ cost.
   (d) Minimal triangulation cost is $dp[0][n-1]$.
4. Insert Interior Points:
   (a) For each interior point:
      i. Find a triangle from the existing triangulation that contains this point.
      ii. Remove the containing triangle.
      iii. Create three new triangles by connecting the interior point to each pair of vertices of the removed triangle.
5. Compute the total sum of perimeters of all triangles formed after inserting all interior points.

Output:

- List of all triangles formed.
- Total sum of the perimeters of these triangles.

5.2.1 Time Complexity Analysis

- Convex Hull Computation: $O(n \log n)$
- Convex Hull Triangulation (Dynamic Programming): $O(n^3)$
- Inserting Interior Points (for each of h interior points): $O(h \cdot n^2)$
- Final Output Generation: $O(n)$

Total:

$$O(n \log n) + O(n^3) + O(h \cdot n^2) + O(n) \approx O(n^3) \quad \text{(in worst case)}$$

5.2.2 Implementation

Code link

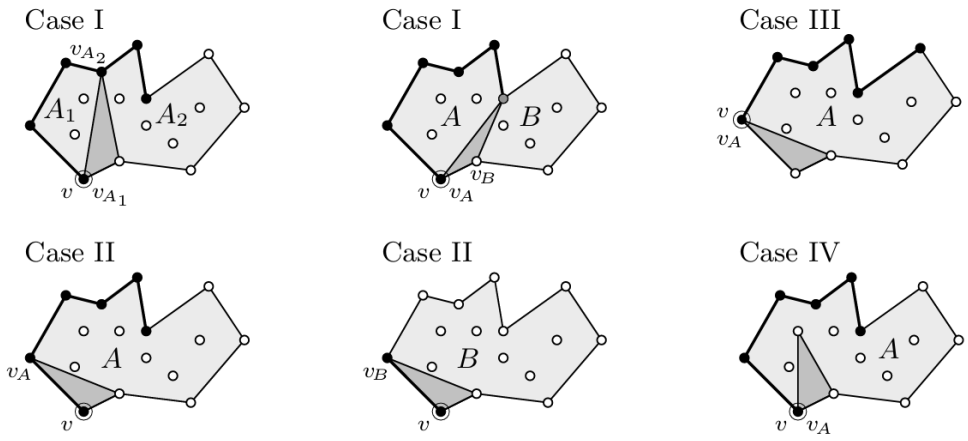


Figure 5: Dynamic Programming Cases for Minimum Weight Triangulation

In the Minimum Weight Triangulation problem, dynamic programming (DP) is used to divide a polygon into smaller parts and triangulate them optimally by minimizing the total weight. Below is a breakdown of the major cases:

- **Case I:** Split the polygon into two parts by connecting a vertex v to an intermediate vertex. Recursively triangulate both parts, then add the triangle formed by v, the first part's vertex and the second part's vertex.
- **Case II:** Split the polygon at an intermediate vertex v_A. Form a triangle (v, v_A, v_{A+1}) and recursively triangulate the remaining polygon.
- **Case III:** A small triangle is formed directly by three adjacent vertices (v, v_A, v_{A+1}), and the rest of the polygon is solved recursively.

- **Case IV:** A point v is inserted inside an existing triangle. The triangle is then replaced with three smaller triangles connecting v to each pair of the original triangle's vertices.

In all cases, the DP algorithm explores all possible ways to split the polygon and stores the optimal splits in a DP table. This table is later used to reconstruct the optimal triangulation that minimizes the total weight of edges.

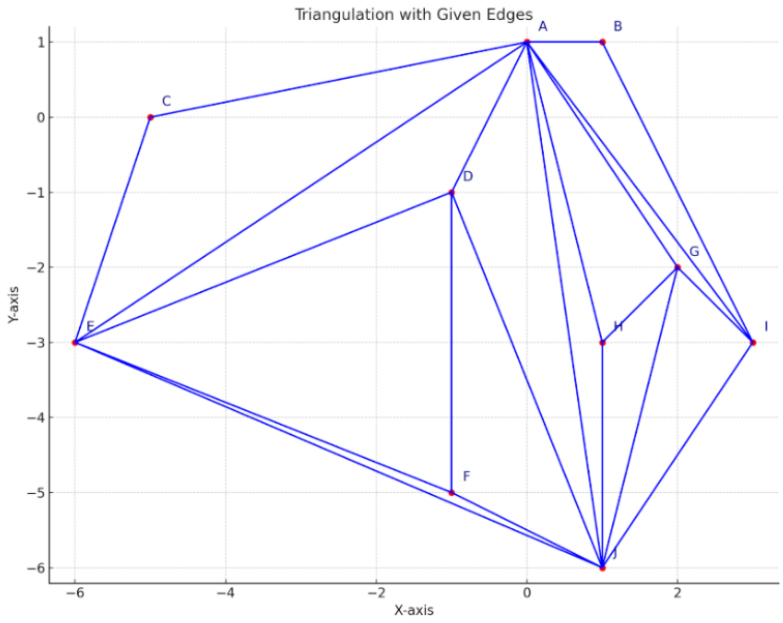
5.2.3 Illustrative Example of DP Algorithm

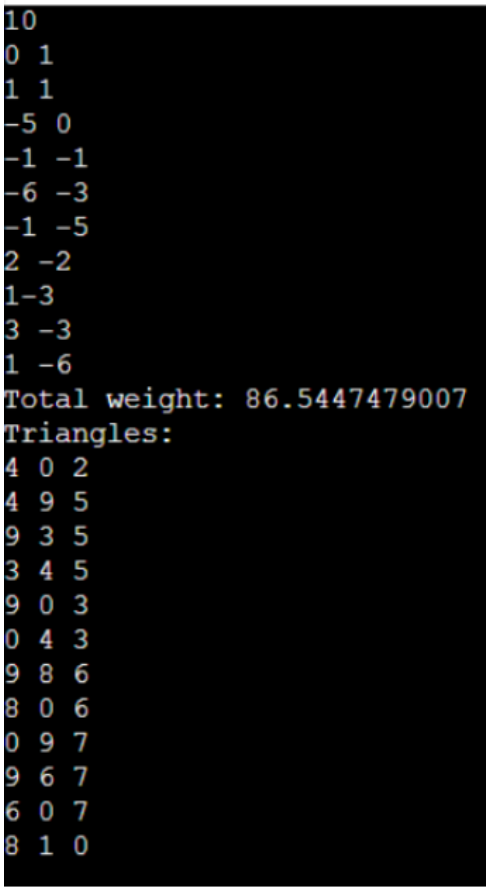
Given $n = 10$ points with 6 interior points:

- $A(0, 1)$, $B(1, 1)$, $C(-5, 0)$, $D(-1, -1)$, $E(-6, -3)$
- $F(-1, -5)$, $G(2, -2)$, $H(1, -3)$, $I(3, -3)$, $J(1, -6)$

After applying the dynamic programming-based triangulation algorithm:

- **Final Triangulation:** 12 triangles formed
- **Total Weight (Perimeter Sum):** 86.54





Approximation Factor Analysis

Let S be a set of n points in the plane. Let T^* denote the minimum weight triangulation (MWT) of S, and let T be the triangulation produced by our algorithm. Define $w(T)$ and $w(T^*)$ as the total edge weight (i.e., the sum of triangle edge lengths) of T and T^*, respectively.

5.2.4 Triangulation of the Convex Hull

Let $H \subseteq S$ be the set of convex hull vertices, and $I = S \setminus H$ be the set of interior points. Our algorithm first computes the optimal triangulation T_0 of the convex polygon formed by H using dynamic programming.

Since T^* includes a triangulation of the convex hull as all subset, we have:

$$w(T_0) \leq w(T^*)$$

5.2.5 Insertion of Interior Points via Triangle Splitting

Each interior point $p \in I$ is inserted by locating a triangle $\triangle ABC \in T_0$ that contains p, and replacing it with three new triangles: $\triangle ABp$, $\triangle BCp$, and $\triangle CAp$. From geometric bounds established in prior work **grantson2006**, the increase in weight from inserting one point is at most $\mathcal{O}(\log n)$ in the worst case.

Thus, inserting all $k = |I| \leq n$ interior points gives:

$$w(T) \leq w(T_0) + \mathcal{O}(k \log n)$$

Substituting from Step 1:

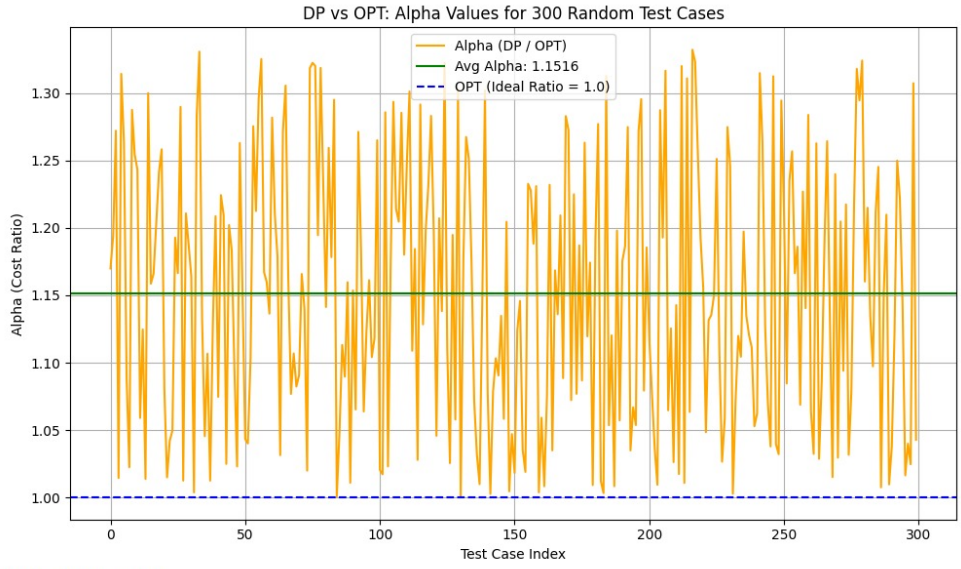
$$w(T) \leq w(T^*) + \mathcal{O}(n \log n)$$

Assuming $w(T^*) = \Omega(n)$ (i.e., the optimal triangulation weight grows at least linearly with n), we obtain the approximation ratio:

$$\frac{w(T)}{w(T^*)} \leq \frac{w(T^*) + O(n \log n)}{w(T^*)} = 1 + O\left(\frac{\log n}{1}\right) = O(\log n) \tag{1}$$

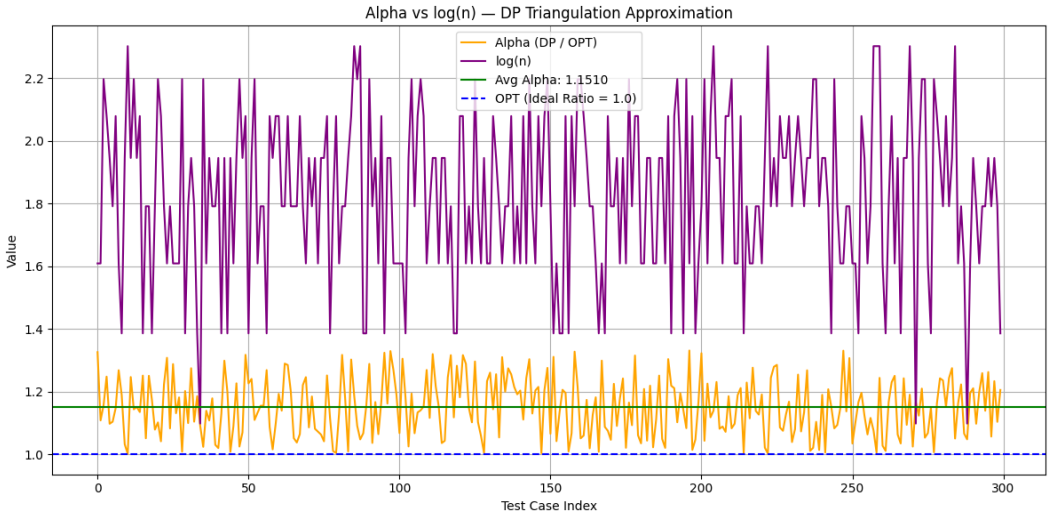
5.2.6 Conclusion

The algorithm achieves an $O(\log n)$ approximation factor for the Minimum Weight Triangulation problem.




```
Average Alpha: 1.1516
Max Alpha: 1.3323
Min Alpha: 1.0002
```

Theoretical vs. Practical Performance Comparison




```
Average Alpha: 1.1510
Max Alpha: 1.3316
Min Alpha: 1.0016
```

While the theoretical analysis provides an upper bound of $O(\log n)$ on the approximation factor, our experimental results demonstrate that the practical performance of the algorithm is often significantly better. In practice, the observed approximation factor $\alpha = \frac{w(T)}{w(T^*)}$ tends to remain much lower than the worst-case theoretical bound.

This suggests that although our algorithm guarantees an approximation factor of at most $O(\log n)$ in the worst case, it often achieves near-optimal triangulations for typical point sets. Thus, our method offers both robust theoretical guarantees and strong practical performance, making it a reliable choice for approximating minimum weight triangulations.

5.3 Local Search Algorithm

Algorithm: Local Search for Minimum Weight Triangulation

Input: A set $P = \{p_1, p_2, \ldots, p_n\}$ of n points in the 2D plane.
Output: A triangulation T of P with a locally minimal total perimeter W_T.
Steps:

1. Initialize by constructing a triangulation of n points.
2. Compute the total perimeter W_T of all triangles in T.
3. Repeat:
   (a) Set $flip\_performed \leftarrow$ false.
   (b) For each pair of adjacent triangles $\triangle ABC$ and $\triangle ABD$ in T sharing an edgeAB:
      i. If the quadrilateral $ACBD$ is convex:
         A. Propose an edge flip: remove edge AB, insert edge CD.
         B. Form new triangles $\triangle ACD$ and $\triangle BCD$.
         C. Compute change in total perimeter:

$$\Delta W = \text{Perimeter}(\triangle ACD) + \text{Perimeter}(\triangle BCD) - (\text{Perimeter}(\triangle ABC) + \text{Perimeter}(\triangle ABD))$$

         D. If $\Delta W < 0$ and edge CD does not intersect existing edges:
         E. Perform the edge flip.
         F. Update triangulation T.
         G. Update perimeter: $W_T \leftarrow W_T + \Delta W$.
         H. Set $flip\_performed \leftarrow$ true.
4. Until no flip is performed.
5. **Output:** The triangulation T and total perimeter W_T.

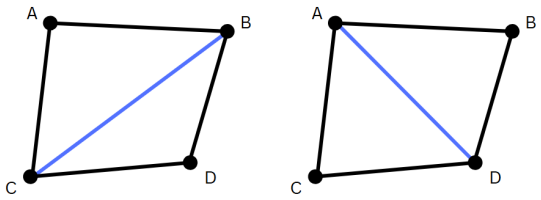


Figure 6: Depicting valid edge flip

5.3.1 Illustrative Example of Local Search Algorithm

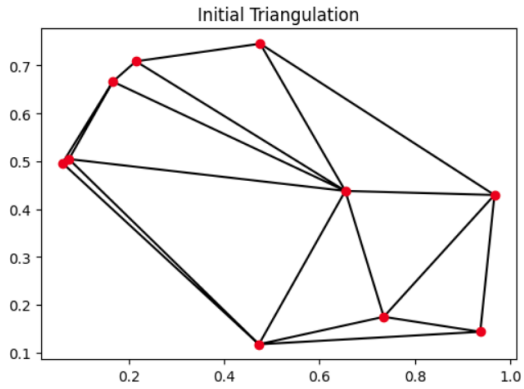


Figure 7: Initial Configuration

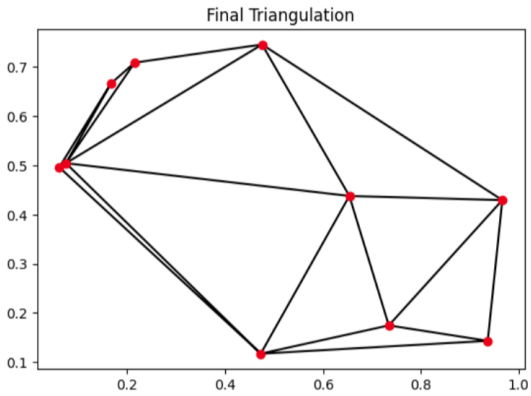


Figure 8: Final Configuration

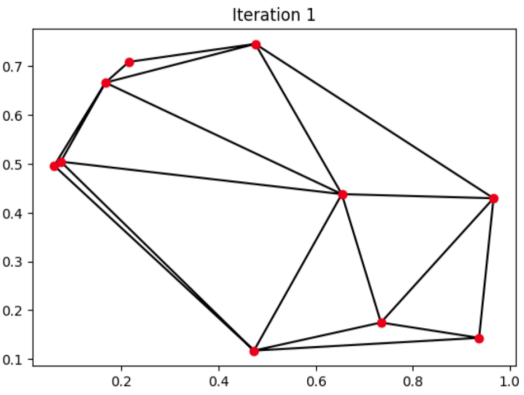


Figure 9: First Iteration

Triangle	Perimeter
[9, 2, 4]	1.1308
[5, 0, 7]	1.2523
[0, 2, 9]	1.5064
[6, 1, 7]	0.8367
[0, 6, 7]	0.9324
[1, 6, 9]	0.9369
[6, 0, 9]	0.9112
[0, 8, 2]	1.3071
[2, 8, 4]	0.4026
[5, 8, 0]	1.2126
[5, 8, 3]	0.6462

Total Perimeter: 11.0751

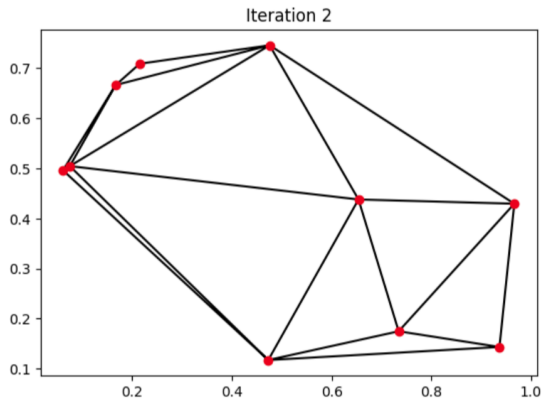


Figure 10: Second Iteration

Triangle	Perimeter
[9, 2, 4]	1.1308
[5, 0, 7]	1.2523
[0, 2, 9]	1.5064
[6, 1, 7]	0.8367
[0, 6, 7]	0.9324
[1, 6, 9]	0.9369
[6, 0, 9]	0.9112
[2, 8, 4]	0.4026
[5, 8, 3]	0.6462
[2, 5, 0]	1.4064
[2, 5, 8]	0.9728

Total Perimeter: 10.9347

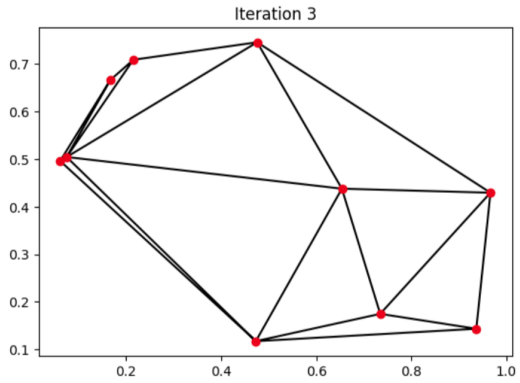


Figure 11: Third Iteration

Triangle	Perimeter
[9, 2, 4]	1.1308
[5, 0, 7]	1.2523
[0, 2, 9]	1.5064
[6, 1, 7]	0.8367
[0, 6, 7]	0.9324
[1, 6, 9]	0.9369
[6, 0, 9]	0.9112
[2, 8, 4]	0.4026
[2, 5, 0]	1.4064
[3, 2, 5]	0.9788
[3, 2, 8]	0.4985

Total Perimeter: 10.7929

5.3.2 Implementation

Code link

5.3.3 Time Complexity Analysis

- Initial triangulation: $\mathcal{O}(n \log n)$ (Optimized time complexity)
- For each iteration, checking all adjacent triangle pairs (up to $\mathcal{O}(n)$ pairs): $\mathcal{O}(n)$
- For each flip candidate, checking for edge intersection with up to $\mathcal{O}(n)$ edges: $\mathcal{O}(n)$
- So, per iteration: $\mathcal{O}(n^2)$
- Let the number of iterations be I (typically polynomial in practice)

Total Time Complexity:

$$\mathcal{O}(n^2 \cdot I)$$

5.3.4 Theoretical Correctness of the Local Search Algorithm

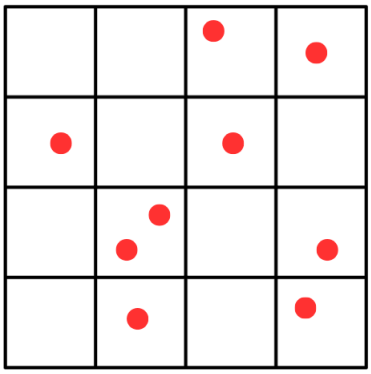


Figure 12: Uniform sampling of points in the plane: each dart has the equal probability to fall in the grid

To analyze the theoretical correctness of our local search algorithm, consider a conceptual model where we discretize the solution space into a uniform grid. We then simulate the local search process as randomly "throwing darts" at this grid. Under the assumption of **uniform distribution**, each dart (i.e., each solution attempt) has an equal probability of landing in any cell of the grid. This model allows us to approximate the behavior of the algorithm in exploring the solution space.

We employ an ϵ-δ **approximation** to describe the quality of the solution obtained by the algorithm in probabilistic terms. In this context:

- $\boldsymbol{\epsilon}$ represents the *tolerance*—how close the obtained solution is to the optimal solution (OPT).
- $\boldsymbol{\delta}$ represents the *error*—the maximum probability that the algorithm fails to find an ϵ-close solution.

Formally, we express this approximation as:

$$\mathbb{P}\left[\text{Solution} \leq (1+\epsilon) \cdot \text{OPT}\right] \geq 1 - \delta$$

where $\boldsymbol{\epsilon} \in [0.1, 0.2]$ and $\boldsymbol{\delta} \leq 0.5$, for $n \geq 20$.

This implies that the probability of the algorithm returning a solution within a factor of $(1+\epsilon)$ of the optimal solution is at least $(1-\delta)$.

These results indicate that the local search algorithm, even when guided by randomized sampling or heuristic evaluation, converges to a solution that is provably close to optimal under practical assumptions.

Acknowledgement
The values of δ, are taken from results of experiments performed in domain of Terrain Modeling, image processing, computer graphics, etc.
Performed on specific values of n, generally very large values n (typically $n = 10^4$ to 10^5), such values and search gives almost Optimal results (taken from their observation).

5.4 Efficient Quadtree Grid based algorithm for minimum weight triangularization

High-Level-Idea To reduce computational complexity, we constructed grids on the plane of points to eliminate longer edges and prioritize shorter ones. This approach helps avoid selecting distant edges that do not contribute efficiently to the solution.

Algorithm: Efficient Quadtree Grid based algorithm for minimum weight triangularization

Require: A set $P = \{p_1, p_2, \ldots, p_n\}$ of n points in the 2D plane
Ensure: A triangulation T of P with minimal total perimeter
Steps:

1. Compute the axis-aligned bounding square that covers all the points. Let it be of size $L \times L$, where L is the larger of the width or height of the bounding box.
2. Recursively partition the square into four equal-sized subgrids (like a quadtree).
3. For each subgrid:
   (a) If it contains more than 4 points:
      i. Repeat the partitioning.
   (b) Else:
      i. Proceed to local triangulation.
4. For each leaf grid with ≤ 4 points:
   (a) Generate all possible valid triangles that:
      - Do not overlap each other.
      - Satisfy the triangle inequality.
      - Have the minimum total perimeter.
5. Recursively merge neighboring grids' triangulations, ensuring:
   - No overlapping edges are introduced.
   - Only boundary points are considered across subgrids.
   - The triangle inequality is preserved.
   - The triangulation is updated with minimal additional perimeter.
6. **Merging Strategy:**
   - Take boundary points of local triangulations (child grids) and connect them with adjacent (top-bottom-left-right) boundary points of neighboring local triangulations.
   - Perform constrained Delaunay triangulation to connect the boundary points and build a unified triangulation.
7. **Output** triangulation T and its total perimeter.
8. **return**

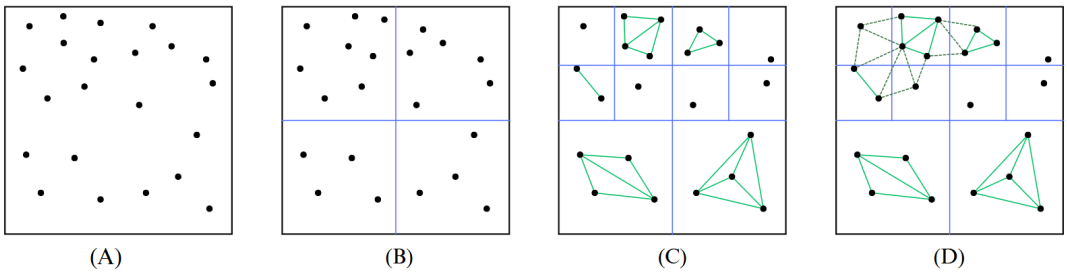


Figure 13: It depicts how an entire set of points is divided into small quadrants, reducing the computation weights of edges of farther points. Merging the adjacent points without violating the conditions for triangularization

5.4.1 Compexity Analysis

- Bounding box: $O(n)$
- Recursive partitioning (quadtree): $O(n \log n)$
- Local triangulation (4 points): $O(n)$
- Merging triangulations (boundary points): $O(n \log n)$
- Output triangulation: $O(n)$

Thus, overall time complexity is:

$$O(n \log n)$$

6 Preliminaries

This section provides an overview of key terminologies used in the Minimum Weight Triangulation problem.

6.1 Convex Polygon

A **convex polygon** is a polygon in which all interior angles are less than 180°, and every line segment between any two points lies entirely inside the polygon.

6.2 Convex Hull

The **convex hull** of a set of points P in the plane is the smallest convex polygon that can be drawn using the outermost points, such that it contains all the points in P. Geometrically, it can be visualized as the shape formed by a rubber band stretched around the outermost points in the set. It plays a crucial role in triangulation, as any valid triangulation must lie within the convex hull of the point set.

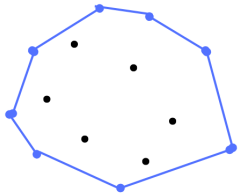


Figure 14: Convex Hull

6.3 Monotone Chain

Graham's Scan and Andrew's Monotone Chain algorithms both compute the Convex Hull in $O(n \log n)$ time. They involve sorting the points and iteratively constructing the upper and lower hulls using cross-product checks to determine the orientation of turns (left or right).

6.4 Delaunay Triangulation

The **Delaunay Triangulation** of a set of points in the plane is a triangulation such that no point lies inside the circumcircle of any triangle in the triangulation. It maximizes the minimum angle of all the triangle angles, avoiding skinny triangles and ensuring numerical stability.

6.5 Quad-Tree

A **Quad-tree** is a hierarchical data structure used to divide a two-dimensional space into four quadrants recursively. Each node represents a square region and can have up to four children, each corresponding to a subregion.

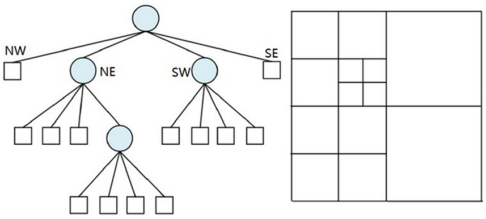


Figure 15: Quad-tree

7 Contribution

ID	Name	Contribution
202201071	Nirva Patel	NP-Hard Proof, Local-search Algorithm with Theoretical Analysis, code, and Efficient Quadtree Grid based algorithm.
202201076	Vidhi Dhanani	Greedy Algorithm, Dynamic Programming with proofs, code, and experiments.
202201149	Kashvi Bhanderi	Greedy Algorithm, Dynamic Programming with proofs, code, and experiments.
202201225	Gunjan Saroliya	Local-search Algorithm with Theoretical Analysis, and Efficient Quadtree Grid based algorithm, Latex file.
202201259	Aastha bhavsar	Local-search Algorithm with Theoretical Analysis, and Efficient Quadtree Grid based algorithm, Latex file.

References

[1] C. Levcopoulos and D. Krznaric, "Quasi-Greedy Triangulations Approximating the Minimum Weight Triangulation," *Journal of Algorithms*, vol. 27, pp. 303–338, 1998.

[2] M. Grantson, C. Borgelt, and C. Levcopoulos, "Fixed Parameter Algorithms for the Minimum Weight Triangulation Problem," *Technical Report LU-CS-TR:2005-238*, Lund University, Sweden, Feb. 2006. Available: `https://www.researchgate.net/publication/220669660_Fixed_Parameter_Algorithms_for_the_Minimum_Weight_Triangulation_Problem`

[3] Refik Samet and Emrah Ozsavas. *Optimization of Quadtree Triangulation for Terrain Models*. In: Khosla, R., Howlett, R.J., Jain, L.C. (eds) Knowledge-Based Intelligent Information and Engineering Systems. KES 2007. Lecture Notes in Computer Science, vol 4692. Springer, Berlin, Heidelberg, 2007. Available at: https://link.springer.com/chapter/10.1007/978-3-540-74607-2$_5$

[4] Wolfgang Mulzer and Günter Rote. *Minimum-Weight Triangulation is NP-hard*. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 158–167. IEEE. Available at: https://doi.org/10.1109/FOCS.2006.24

[5] Sharath Raghvendra and Mariëtte C. Wessels. *A Grid-Based Approximation Algorithm for the Minimum Weight Triangulation Problem*. In: Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018), pp. 101–120. Available at: https://arxiv.org/abs/1706.03263

[6] Faisal N. Abu-Khzam, Christine Markarian, Friedhelm Meyer auf der Heide, and Michael Schubert. *Approximation and Heuristic Algorithms for Computing Backbones in Asymmetric Ad-Hoc Networks*. Available at: https://link.springer.com/article/10.1007/s00453-010-9442-z