Mathematical Proof and Analysis: Algorithm 4 vs AlgoPro

1. Preliminary Definitions

Let G = (V, E, w) be a weighted undirected graph where: - V is the set of vertices - E V × V is the set of edges - w: E → + is the weight function - |V| = n vertices - |E| = m edges

2. Algorithm 4 Analysis

2.1 Correctness Proof

Theorem 1: Algorithm 4 correctly computes the minimax path values for all pairs of vertices.

Proof: 1. Let P(u,v) be the minimax path between vertices u and v 2. Let M(u,v) be the maximum edge weight in P(u,v) 3. For any vertices u,v V: `M(u,v) = min{max{w(e) | e  P} | P is a path from u to v}`

4. MST Property:
   - Let T be the MST of G
   - For any u,v V, the path in T between u and v has maximum edge weight M(u,v)
5. Edge Removal Process:

   ```
   For edge e = (x,y) in T (in descending weight order):
   - Remove e from T
   - Components C , C  formed
   -  u  C , v  C : M(u,v) = w(e)
   ```

Lemma 1: The process maintains the invariant that M(u,v) actual minimax distance.

2.2 Complexity Analysis

Time complexity breakdown:

1. MST Construction:

   ```
   T_MST = O(n² log n)  // Using NetworkX implementation
   ```

2. Edge Sorting:

   ```
   T_sort = O(n² log n)  // For all edges
   ```

3. DFS Operations:

   ```
   T_DFS = O(n + m) per iteration
   Number of iterations = O(n)
   ```

```
Total T_DFS = O(n³)
```

Overall Complexity:

```
T_total = T_MST + T_sort + T_DFS
T_total = O(n² log n) + O(n² log n) + O(n³)
T_total = O(n³)
```

3. AlgoPro Analysis

3.1 Correctness Proof

Theorem 2: AlgoPro computes correct minimax paths while maintaining lower complexity.

Proof: 1. Kruskal's MST Construction with Union-Find: `Let S , S , ..., S be disjoint sets For each edge e = (u,v) in ascending weight order: If Find(u) Find(v): Union(u,v) Add e to MST`

2. Component Tracking:

```
For each edge e = (x,y) removed from MST:
  C  = QuickFind(x)  // O(1) with path compression
  C  = QuickFind(y)  // O(1) with path compression
  Update M(u,v) = w(e) for u  C , v  C
```

Lemma 2: Path compression in Union-Find ensures O((n)) amortized time per operation, where is the inverse Ackermann function.

3.2 Improved Complexity Analysis

1. Kruskal's MST with Union-Find:

```
T_kruskal = O(m log n)  // where m = O(n²)
          = O(n² log n)
```

2. Edge Sorting:

```
T_sort = O(n log n)  // Only MST edges
```

3. Optimized DFS:

```
T_DFS = O(n) per component
Total components = O(n)
Total T_DFS = O(n²)
```

Overall Complexity:

```
T_total = T_kruskal + T_sort + T_DFS
T_total = O(n² log n) + O(n log n) + O(n²)
T_total = O(n² log n)
```

4. Performance Comparison

4.1 Key Improvements

1. Union-Find Optimization:

   ```
   Classic DFS: O(n) per query
   Union-Find: O( (n))   O(1) per query
   Improvement factor: O(n)
   ```

2. Component Management:

   ```
   Algorithm 4: O(n) DFS per edge
   AlgoPro: O(1) lookup per edge
   Improvement factor: O(n)
   ```

3. MST Construction:

   ```
   NetworkX: O(n^2 log n) with overhead
   Custom Kruskal: O(n^2 log n) optimized
   Improvement: Constant factor + reduced memory
   ```

4.2 Theoretical Speed-up

For large graphs ($n \to \infty$):

```
Speed-up ratio = O(n^3)/O(n^2 log n)
               = O(n/log n)
```

5. Why AlgoPro Wins

1. **Asymptotic Improvement:**
   - Reduces cubic complexity to near-quadratic
   - Eliminates redundant DFS traversals
   - Achieves optimal component tracking

2. **Data Structure Optimization:**

   ```
   Union-Find operations: O( (n))   O(1)
   Path compression benefit: O(n) → O(1)
   Component lookup: O(n) → O(1)
   ```

3. **Memory Efficiency:**

   ```
   Algorithm 4: O(n^2) + library overhead
   AlgoPro: O(n^2) optimized structures
   ```

4. **Operation Count Reduction:**

   ```
   Algorithm 4: O(n^3) operations
   AlgoPro: O(n^2 log n) operations
   Reduction ratio: O(n/log n)
   ```