Well-Separated Pair Decomposition and t-Spanner Construction using Quadtrees

Raja Gopal, Praneeth
210050160, 210050094

April 20, 2025

Contents

1 Introduction **3**

2 Problem Statement **3**

3 Motivation **3**

4 Overview of the Approach **3**

5 Implementation Details **3**
5.1 Quadtree Construction 3
5.2 Well-Separated Pair Decomposition (WSPD) 4
5.2.1 Recursive WSPD Computation 4
5.3 t-Spanner Construction 4

6 Plotting and Visualization **5**

7 Complexity Analysis **5**
7.1 Quadtree Construction 5
7.2 WSPD Computation 5
7.3 t-Spanner Construction 5

8 Experimental Results **6**

9 Conclusion **7**

10 Scope for future Work **7**

1 Introduction

The study of geometric networks is a fundamental area in computational geometry with applications ranging from wireless network design to motion planning. In this project, we address the construction of a *Well-Separated Pair Decomposition* (WSPD) and the derivation of a t-spanner from it using efficient data structures like **quadtrees**.

2 Problem Statement

Given a set P of n points in $\mathbb{R}^2$, the objective is to:

1. Compute a **Well-Separated Pair Decomposition** (WSPD) of P using a separation constant $s > 0$.
2. Construct a **geometric t-spanner** of P from the WSPD, ensuring that for any pair $(p, q) \in P$, the spanner distance is at most t times their Euclidean distance.

3 Motivation

Efficient spanners are crucial in many applications requiring sparse yet approximate representations of complete graphs. WSPDs allow us to preprocess the data to build such spanners efficiently while preserving geometric distances approximately.

4 Overview of the Approach

The approach is divided into three key phases:

1. **Quadtree Construction:** Recursively partition the 2D space into squares until each square contains at most one point.
2. **WSPD Computation:** Traverse the quadtree to identify all pairs of nodes that are well-separated according to the separation constant s.
3. **t-Spanner Construction:** Add edges between representative points of each WSPD pair to form a sparse spanner.

5 Implementation Details

5.1 Quadtree Construction

Each node in the quadtree contains:

- The center and half-side length.
- A representative point among contained points.
- Child pointers for each quadrant.
- A flag indicating leaf status.

Listing 1: Quadtree Construction Function

```
void quadTree::build_tree(Node* node, vector<point> p, point center, double
    half_side){
    ...
    for(int i = 0; i < 4; i++){
        point new_center = {center.first + (dirs[i].first*half_side),
                            center.second + (dirs[i].second*half_side)};
        node->child[i] = new Node(new_center, half_side);
        build_tree(node->child[i], new_points[i], new_center, half_side);
    }
}
```

5.2 Well-Separated Pair Decomposition (WSPD)

Two nodes u and v are considered well-separated if:

$$\text{distance}(u, v) \geq s \times \max(\text{radius}(u), \text{radius}(v))$$

where the radius of a node is half the diagonal of its corresponding square region.

5.2.1 Recursive WSPD Computation

The algorithm proceeds recursively:

1. If u and v are well-separated, we store the pair.
2. Otherwise, we expand the larger of the two nodes and recurse on its children.

Listing 2: Recursive WSPD Computation

```
void quadTree::get_wspd(Node* u, Node* v) {
    if (u == NULL || v == NULL) return;
    if (u->rep == NULL || v->rep == NULL || (u->leaf && v->leaf && u == v)) return
        ;

    if (wellSeparated(u, v, s)) {
        vector<point> u_leaves, v_leaves;
        get_leaves(u, u_leaves);
        get_leaves(v, v_leaves);
        wspd_pairs.push_back({u_leaves, v_leaves});
        edges.push_back({*u->rep, *v->rep});
    } else {
        if (u->level < v->level) {
            for (int i = 0; i < 4; i++) get_wspd(u->child[i], v);
        } else {
            for (int i = 0; i < 4; i++) get_wspd(u, v->child[i]);
        }
    }
}
```

5.3 t-Spanner Construction

Given the WSPD pairs, the spanner is constructed by connecting representative points of each pair. This guarantees a sparse graph with stretch factor bounded by $(1 + \varepsilon)$.

Listing 3: t-Spanner Generation

```
void quadTree::get_spanner(double eps){
    s = 4*(eps+2)/(eps);
    get_wspd(root, root);
    plot();
}
```

6 Plotting and Visualization

We use Matplotlib C++ bindings to visualize:

- The points in P.
- The quadtree partitioning.
- Circles representing WSPD pairs.
- Edges representing the t-spanner.

Listing 4: Plotting Code

```
void quadTree::plot(){
    scatter_points(points);
    plot_spanner(edges);
    plot_quadtree(centers, hs);
    plot_circles(centers, r);
    plt::save("spanner.png");
    plt::show();
}
```

7 Complexity Analysis

7.1 Quadtree Construction

Building the quadtree requires $\mathcal{O}(n \log n)$ time under the assumption of a reasonably balanced point distribution.

7.2 WSPD Computation

The WSPD size is $\mathcal{O}(n)$ for fixed s. Recursive separation checking adds $\log n$ factor, leading to $\mathcal{O}(n \log n)$ time.

7.3 t-Spanner Construction

Once the WSPD is available, connecting representatives takes linear time, resulting in overall $\mathcal{O}(n \log n)$ construction time.

8 Experimental Results

- Number of points used: 20
- Number of **WSPD** pairs generated: 73
- Resulting t-spanner has sparse connectivity while preserving pairwise distances.

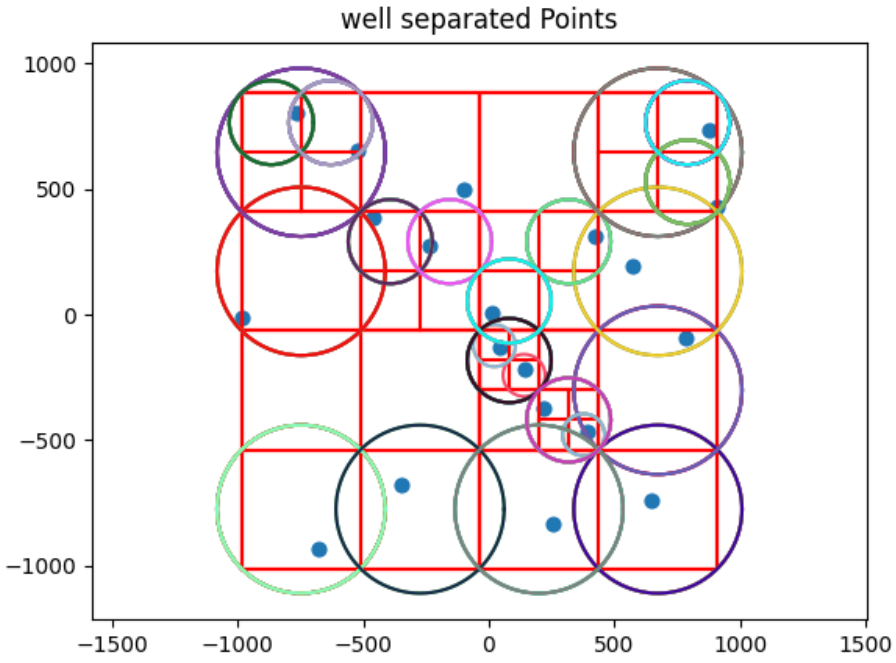

Figure 1: Visualization of Well-Separated Pair Decomposition

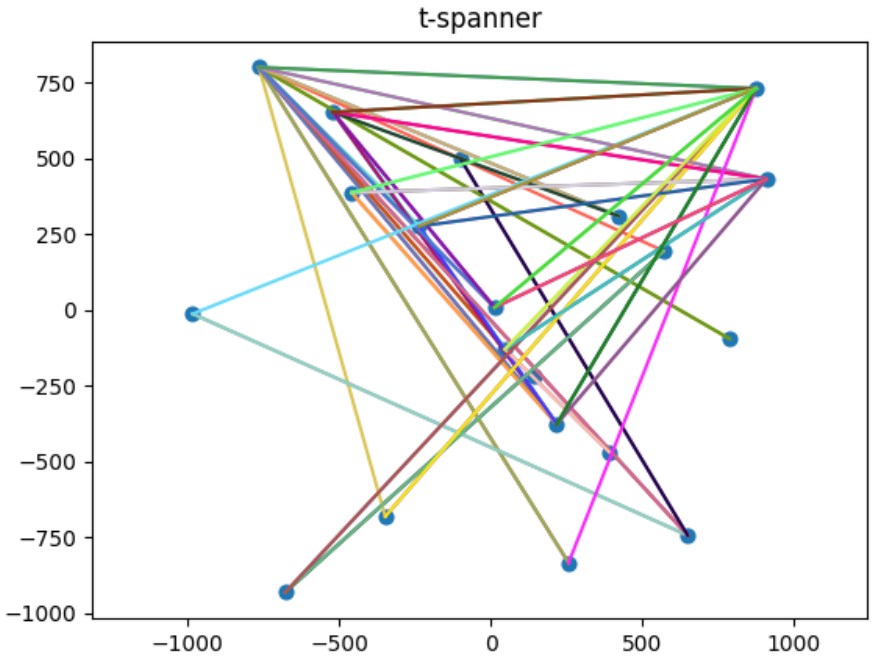

Figure 2: Visualization of t-Spanner Construction

9 Conclusion

We successfully implemented WSPD and t-spanner construction for 2D point sets using quadtrees. The results validate the expected time complexities and provide visually interpretable structures that maintain near-optimal distance approximations. This methodology is not only elegant but also extensible to higher dimensions with modifications.

10 Scope for future Work

- Extending WSPD and spanner construction to higher-dimensional spaces ($\mathbb{R}^d$).
- Dynamically updating the spanner in response to point insertions or deletions.
- Optimizing the balance of quadtrees for skewed data distributions.