NORTHWESTERN UNIVERSITY

# An $O(n \log n)$ Edge-Based Algorithm for Obstacle-Avoiding Rectilinear Steiner Tree Construction

Jieyi Long, Hai Zhou, and Seda Ogrenci Memik

**Dept. of EECS, Northwestern Univ.**

# Outline

1. Introduction
2. Problem Formulation
3. Edge-Based OARST Algorithm
4. Experimental Results
5. Conclusions

# Outline

1. Introduction
2. Problem Formulation
3. Edge-Based OARST Algorithm
4. Experimental Results
5. Conclusions

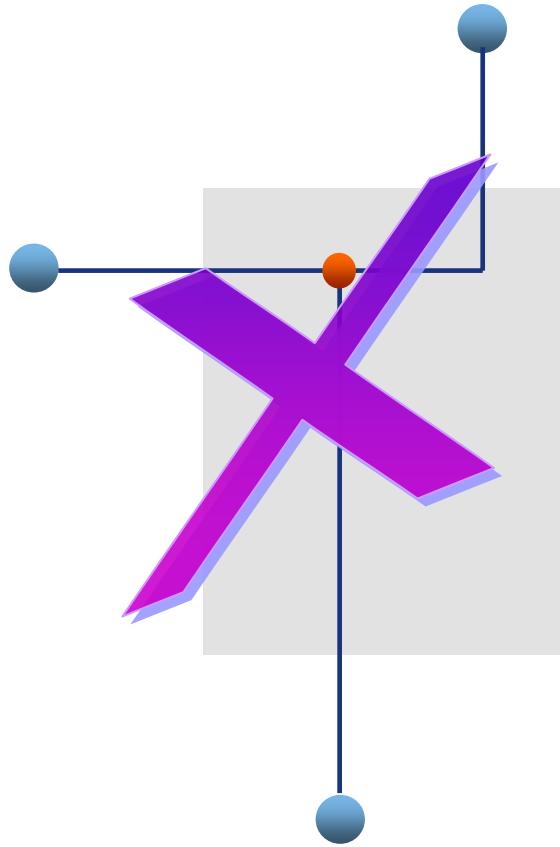
# Introduction

**Steiner Routing**

**Existing Work:** Mostly assumes obstacle-free routing plane

**Modern VLSI Chip**

**Obstacles:** IP cores, macro blocks, pre-route nets



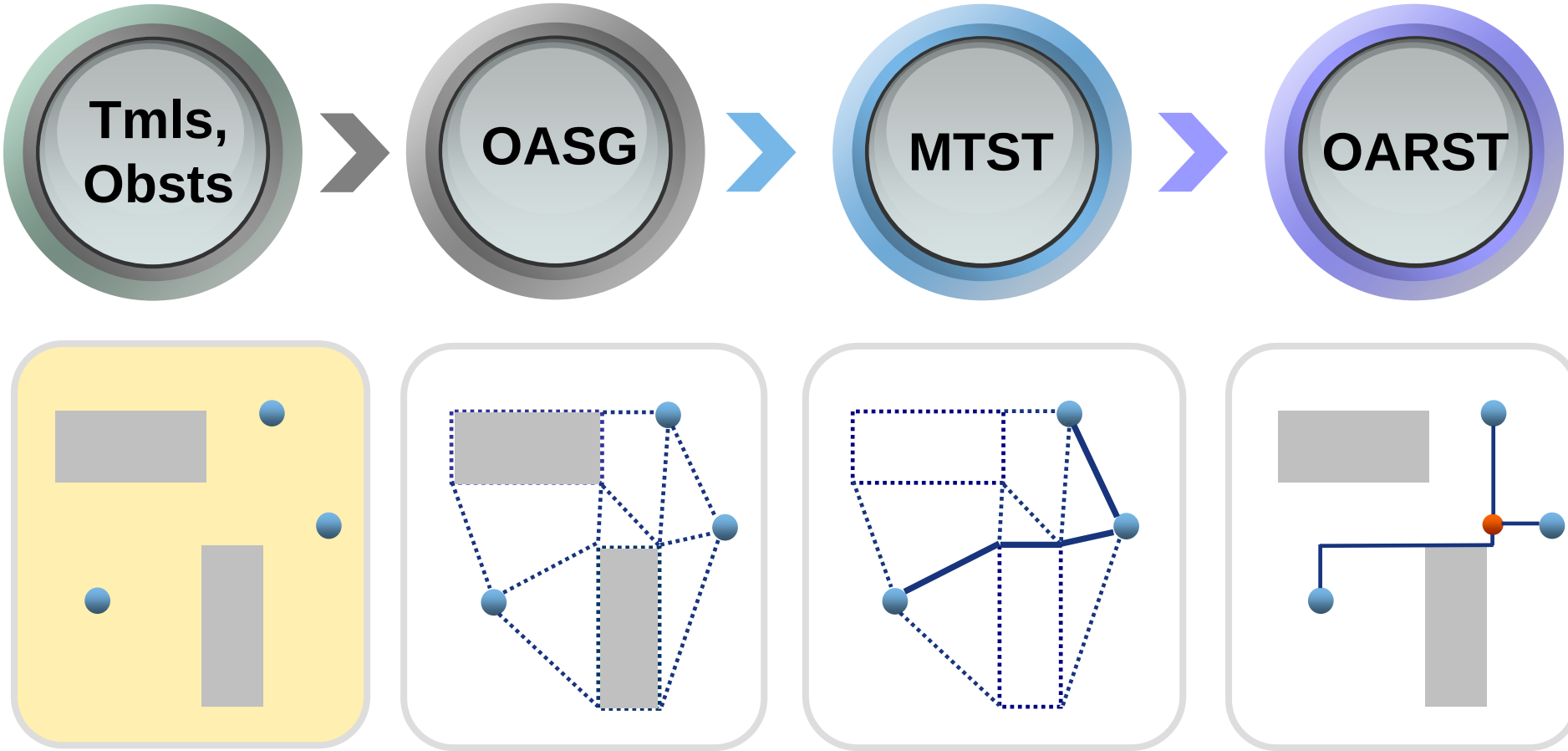
# Contents

# Problem Formulation

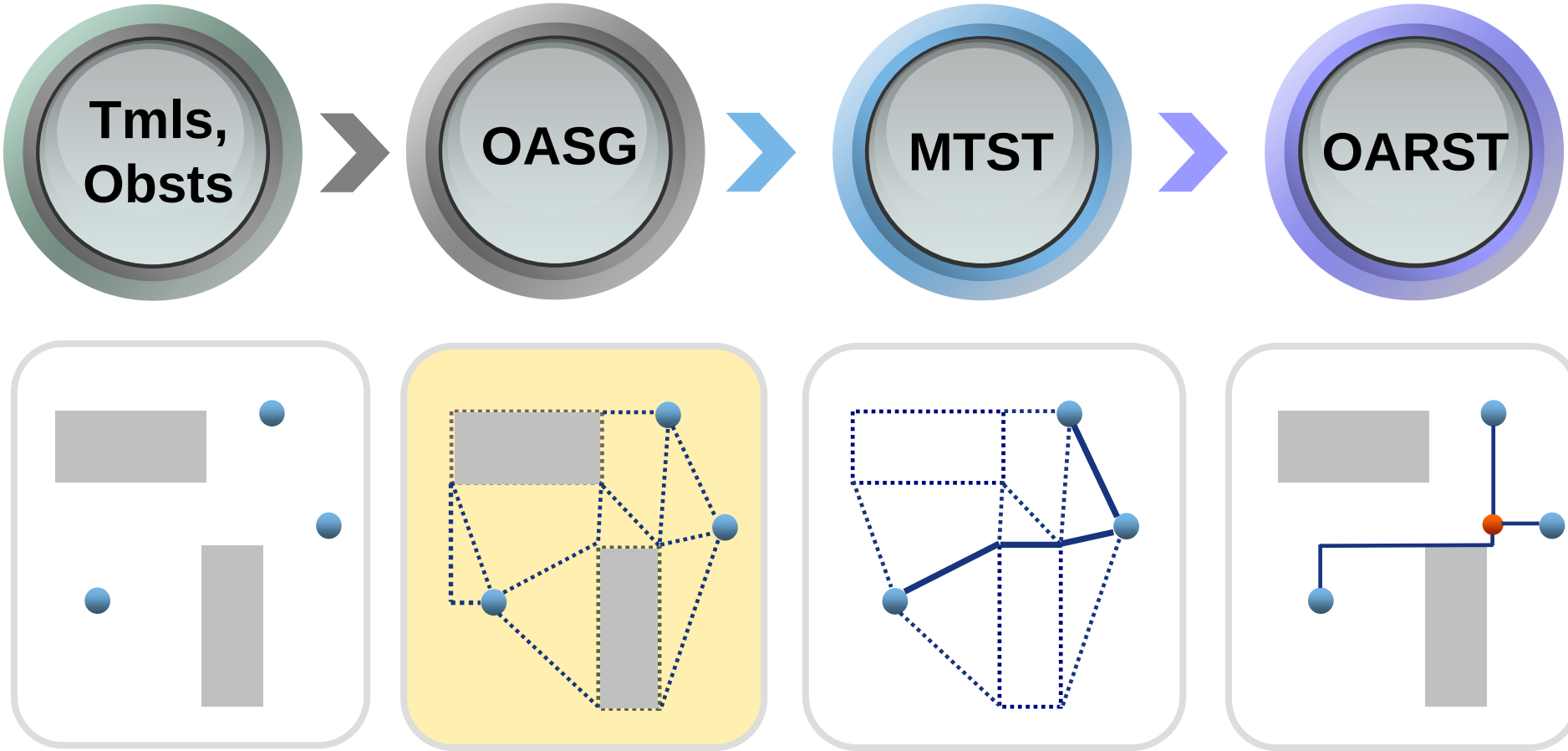
**Obstacle-Avoiding Rectilinear Steiner Minimal Tree**

# Contents

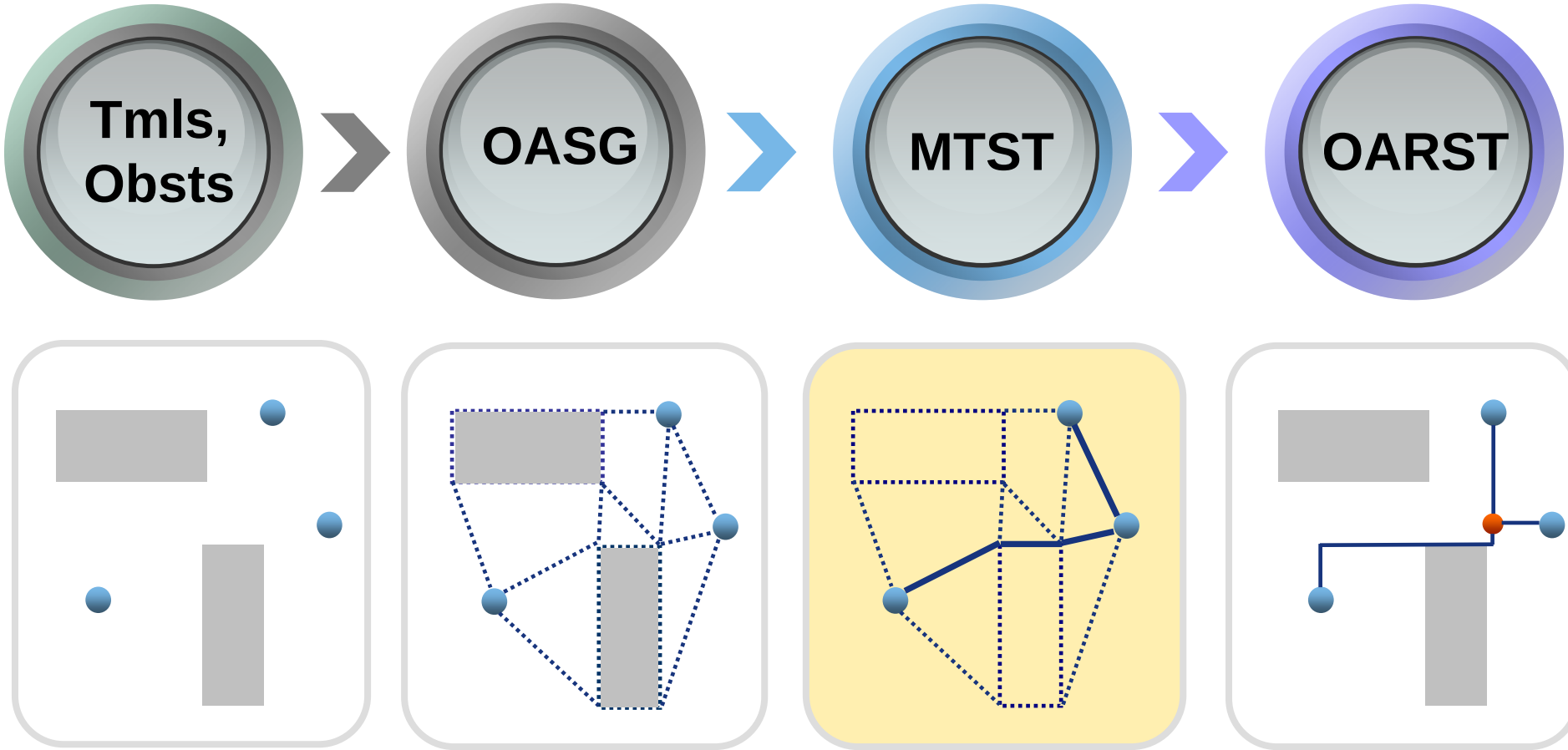
# Algorithm Overview



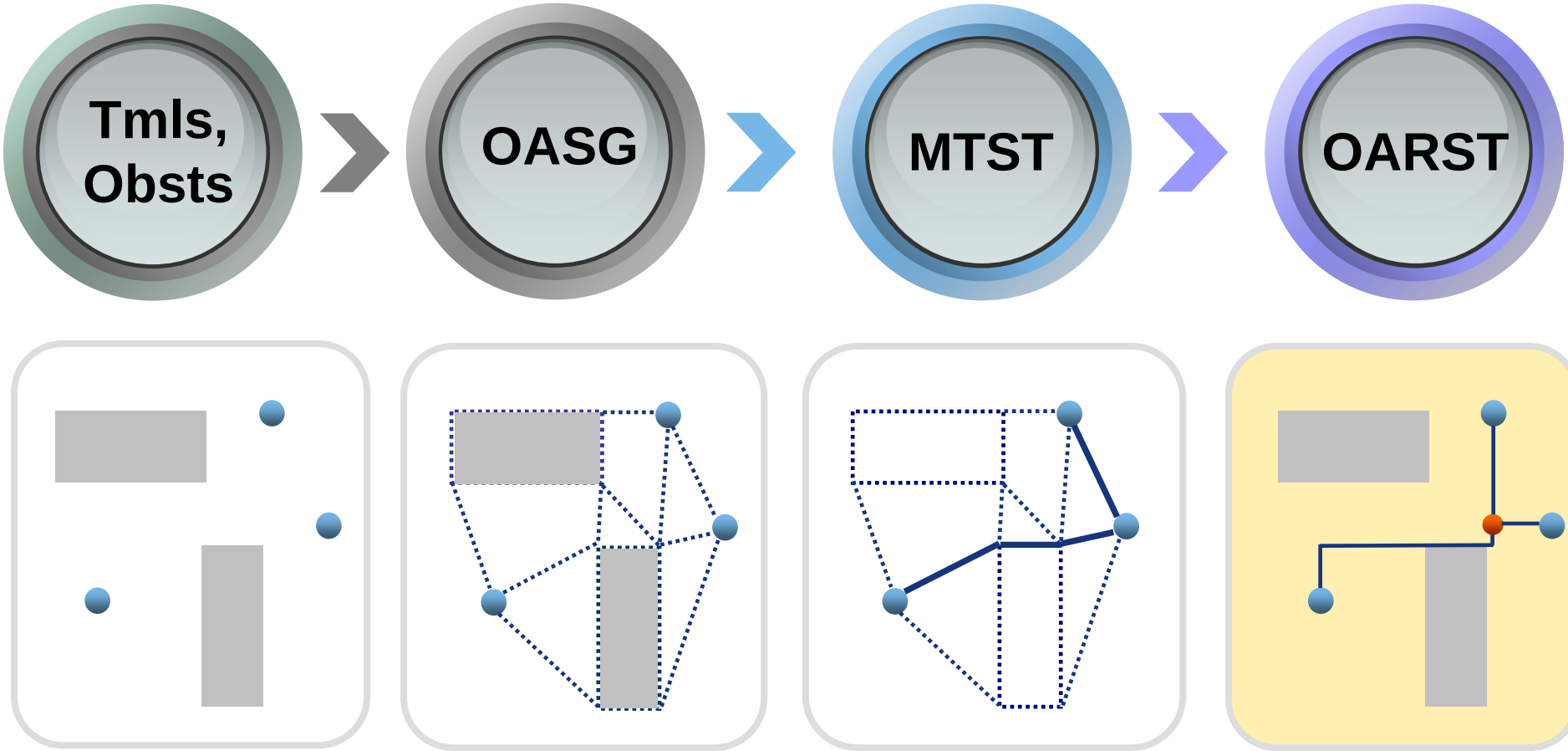
# Algorithm Overview



# Algorithm Overview

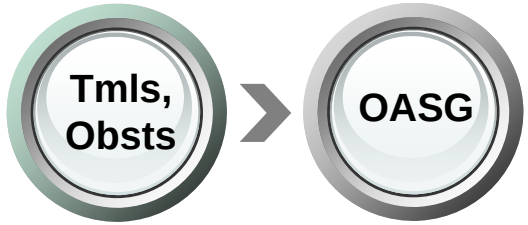


# Algorithm Overview



# Step 1: OASG Generation

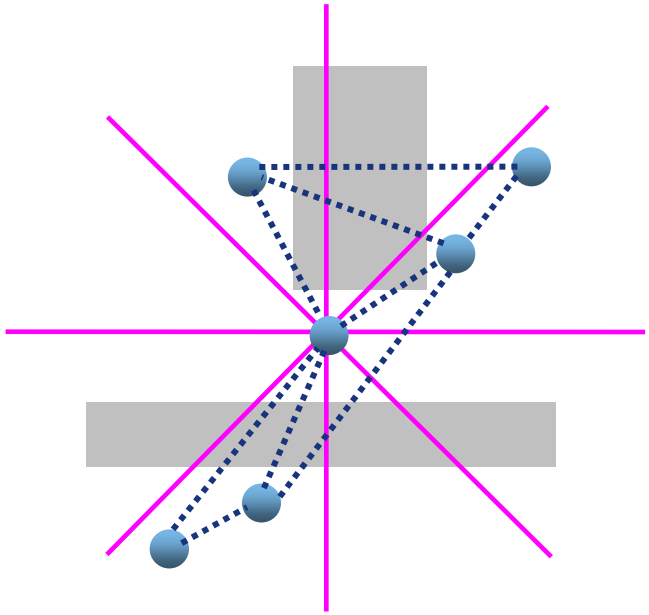
## Obstacle-Avoiding Spanning Graph (OASG)




**On a obstacle-free routing plane [ Zhou et al.]**

**Spanning Graph**

- Octal partition
- Minimum Spanning Tree (MST) embedded



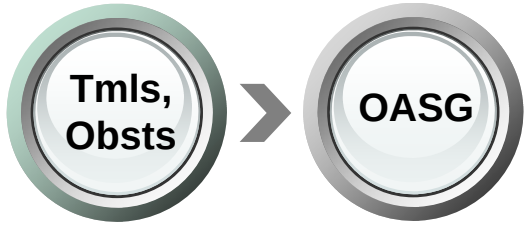
**What if there are obstacles…**

**Obstacle-Avoiding Spanning Graph**

- How to define SG when there are obstacles?
- How to construct OASG efficiently?
- Is MST still embedded?

# Step 1: OASG Generation

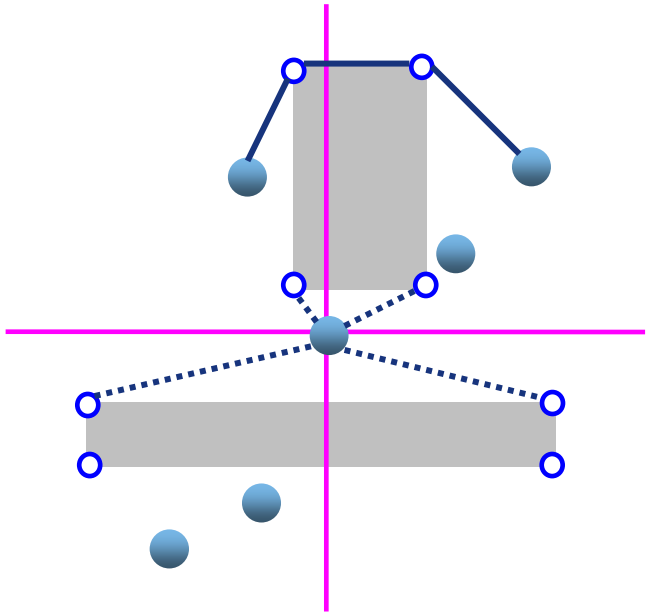
## Obstacle-Avoiding Spanning Graph (OASG)




**Q1:** How to Define SG when there are obstacles?

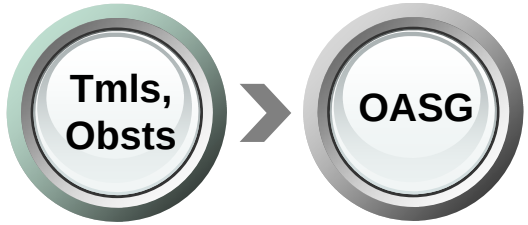
**A:** **1. Connect the terminals and corners**
- Intuition: when blocked, shortest paths going through corners

**2. Quadrant partition only**
- easier to handle, does not degrade the solution quality much



# Step 1: OASG Generation

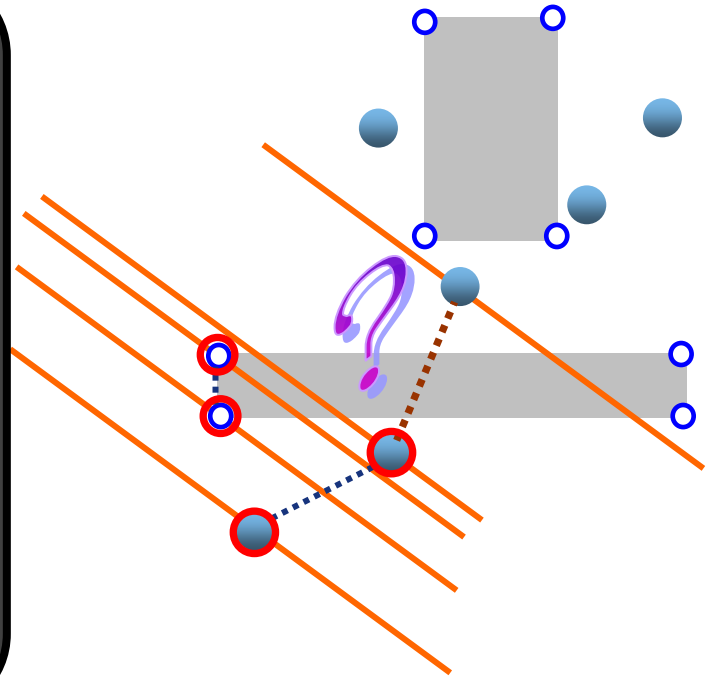
## Obstacle-Avoiding Spanning Graph (OASG)




**Q2:** How to construct OASG efficiently?

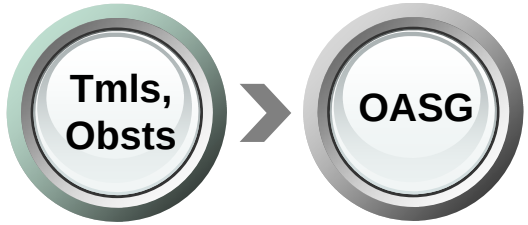
**A: Sweeping Line Algorithm (Quad1)**

- + order the vertices by x+y;
- + sweep from the first vertex in the order;
- + Active Set (vertices in red circles): contain all the swept vertices whose closest vertex is to be discovered;
- + blockage checking…



# Step 1: OASG Generation

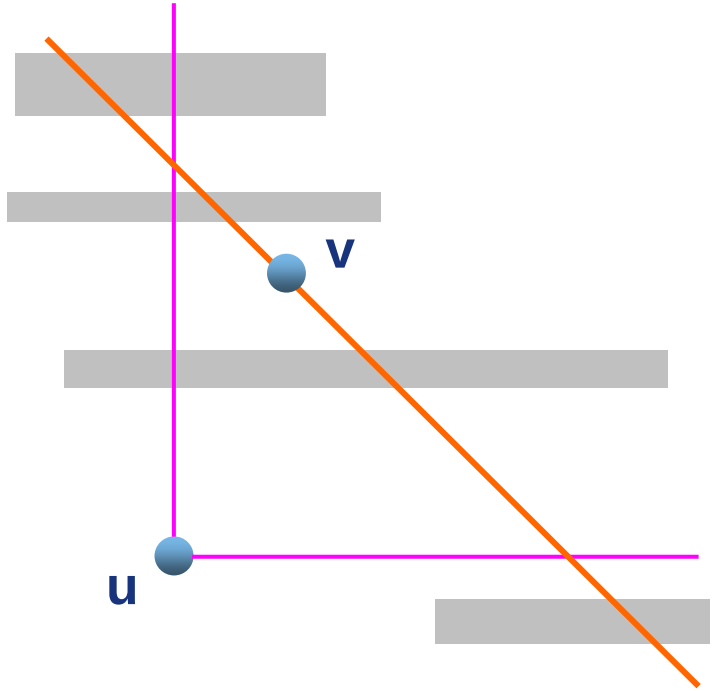
## Obstacle-Avoiding Spanning Graph (OASG)




### Blockage Checking

**Lemma 1**: (u, v) intersects with a horizontal edge if and only if:

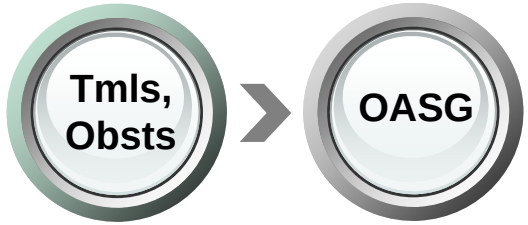
1) The edge is intersecting with the sweeping line
2) The y-coordinate of the edge is between those of u and v




Use balanced BST to store y-coordinate of the edges, ensuring $O(\log n)$ query time

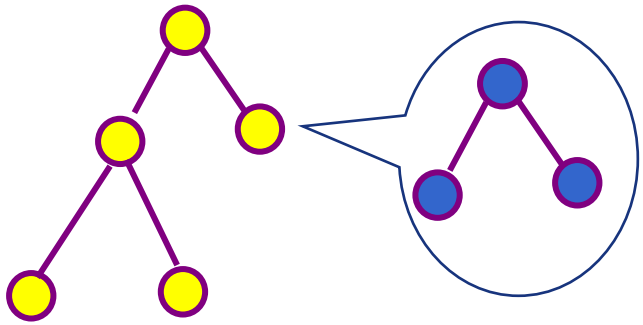
# Step 1: OASG Generation

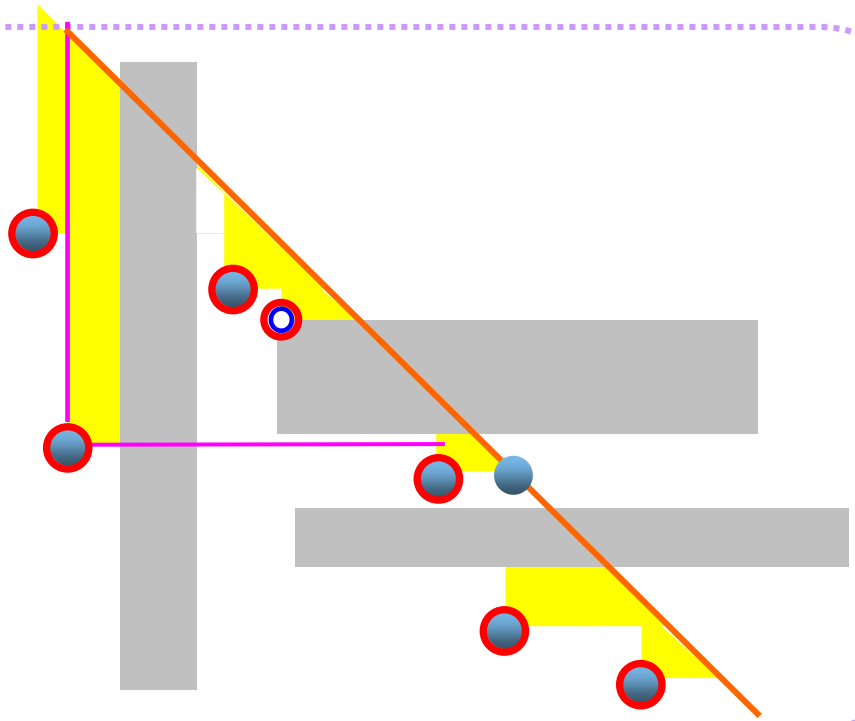
## Obstacle-Avoiding Spanning Graph (OASG)




### Data Structure for the Active Set

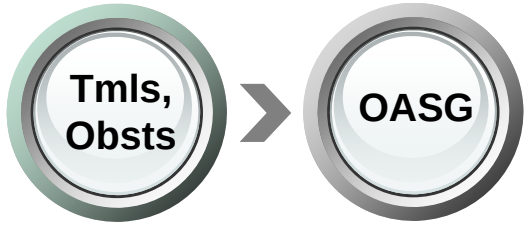
**Hierarchical Balanced Binary Search Tree**





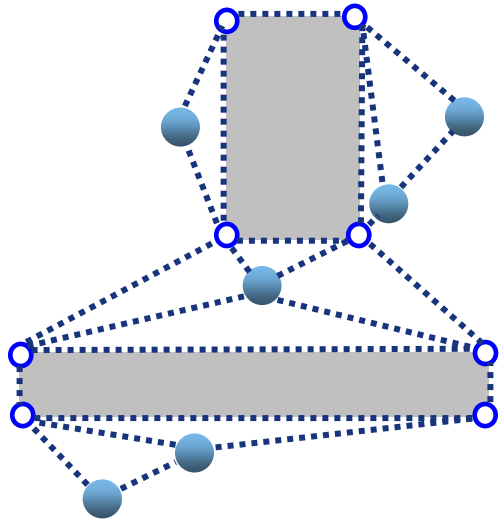
# Step 1: OASG Generation

## Obstacle-Avoiding Spanning Graph (OASG)



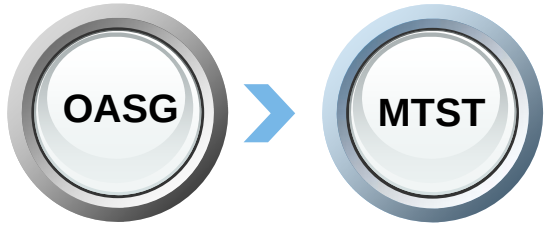

**Q3:** Does the OASG contain the MST connecting all the terminals?

**A: Unfortunately no, but we can extract a structure called MTST from OASG as the starting point for refinement**



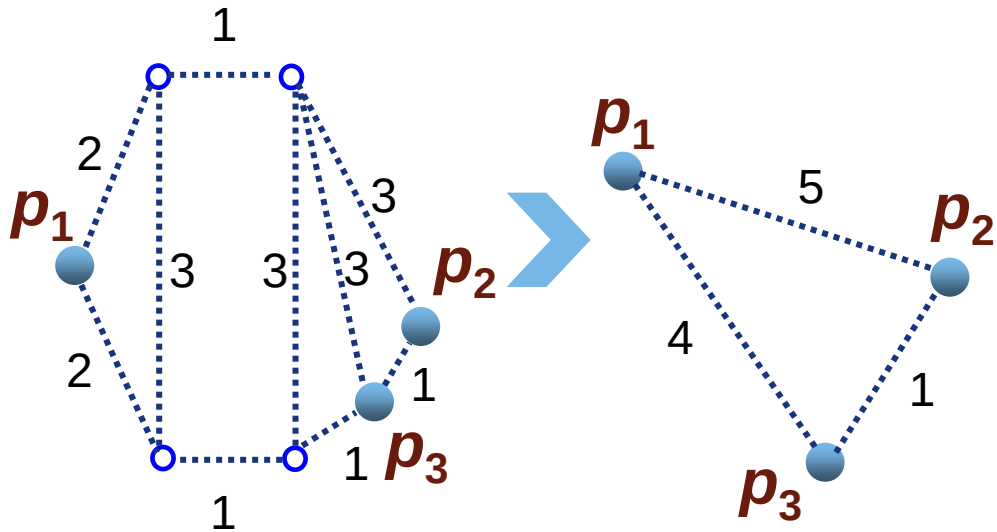
# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)



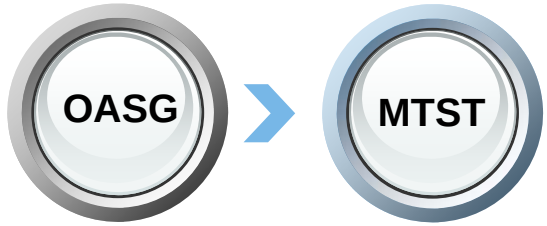

**MTST**

- Map an OASG to a Complete Graph CG
- + a vertex for each terminal
- + edge length equals the shortest path length between the two end vertices on the OASG



# Step 2: MTST Construction

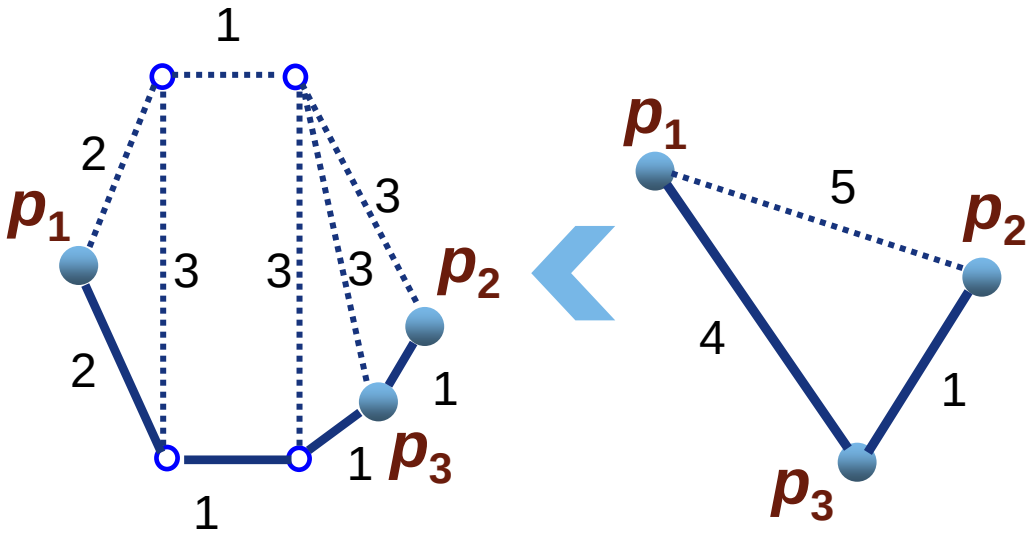
## Minimum Terminal Spanning Tree (MTST)




**MTST**

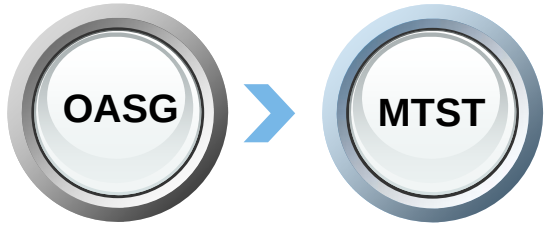
- Generate MST(CG)
- Map MST(CG) to MTST(OASG)

The definition gives a trivial algorithm to generate MTST. However, its complexity is at least $O(n^2 \log n)$



# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)




**Can we do better than $O(n^2 \log n)$?**

### Extreme Cases

**Only two terminals**: MTST degenerated to shortest path between these two terminals, which can be found by Dijkstra's algorithm in $O(n \log n)$ time

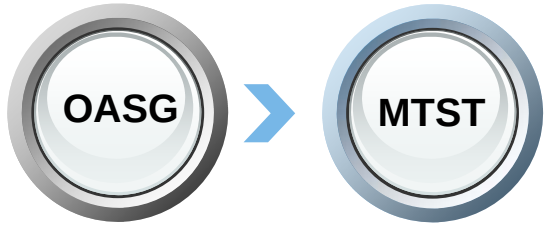
**All vertices are terminals**: MTST degenerated to MST, which can be found using Kruskal's algorithm in $O(n \log n)$ time

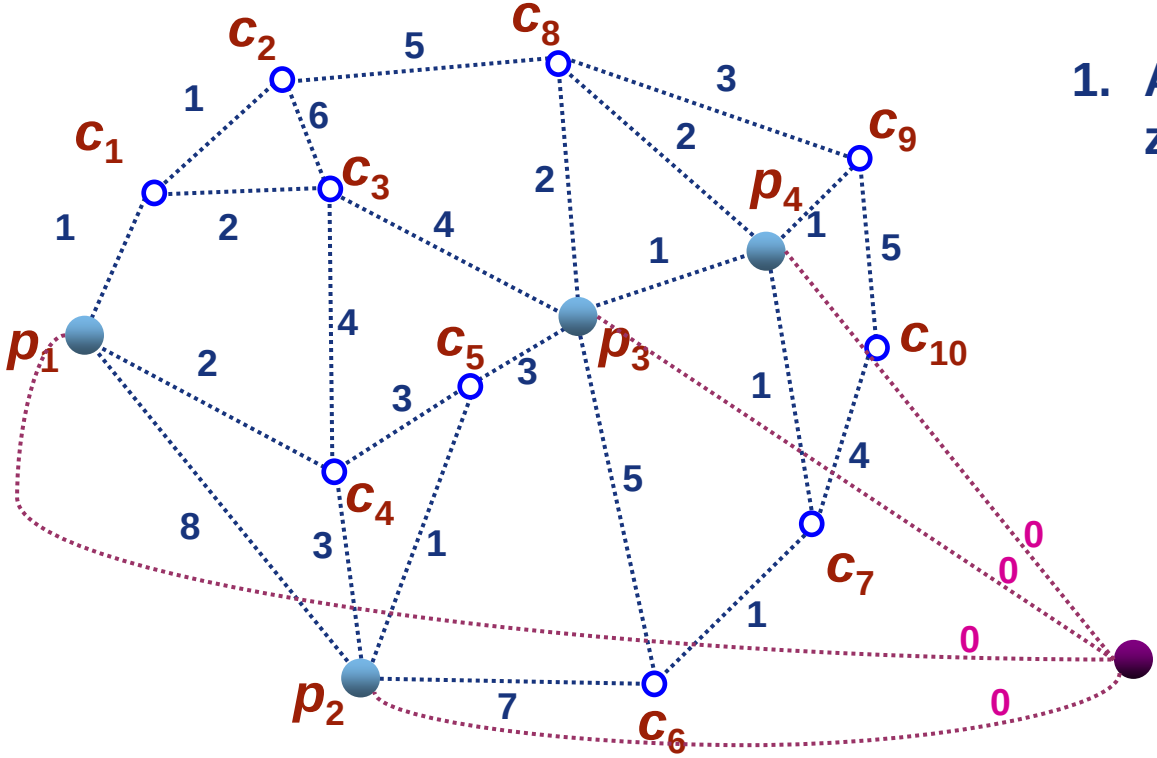
**It is reasonable to assume that certain combination of Dijkstra's and Kruskal's alg. can generate MTST in $O(n \log n)$ time**

# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)

**Can we do better than $O(n^2 \log n)$?**

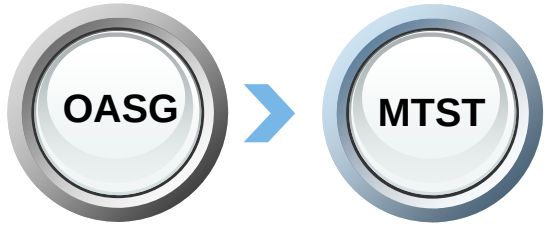


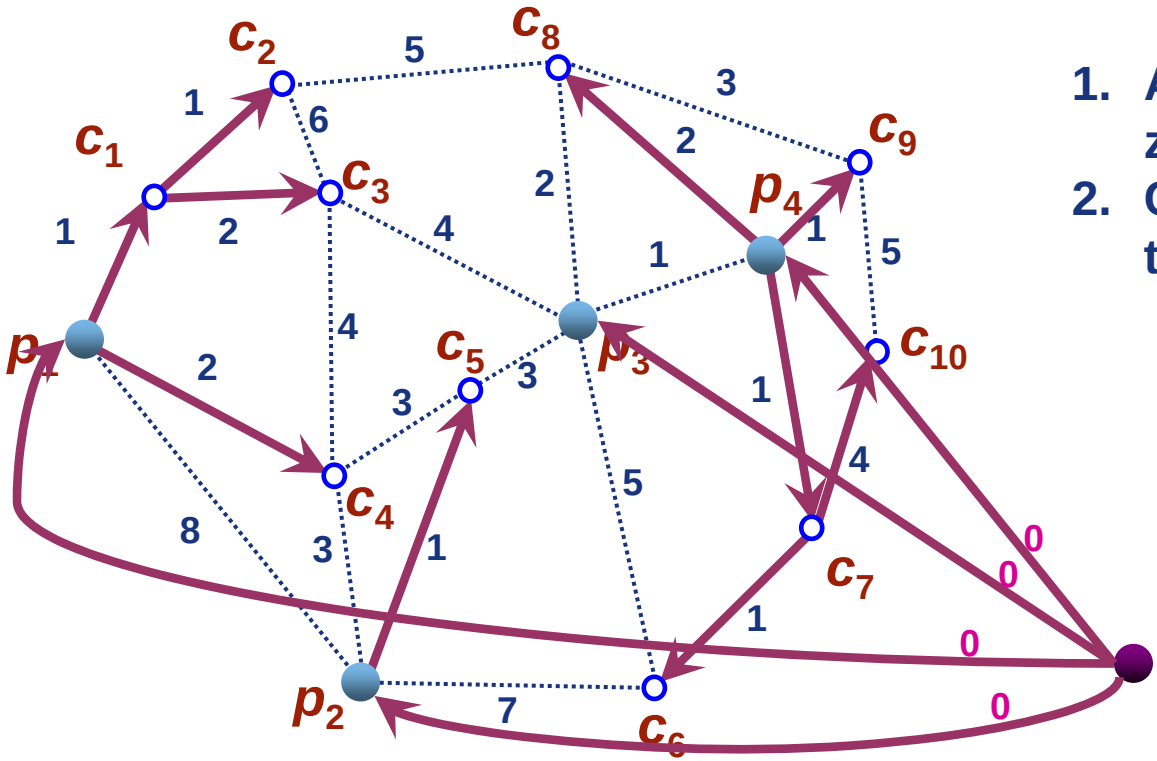

1. Add a dummy node and zero-weighted edges

# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)




**Can we do better than $O(n^2 \log n)$?**

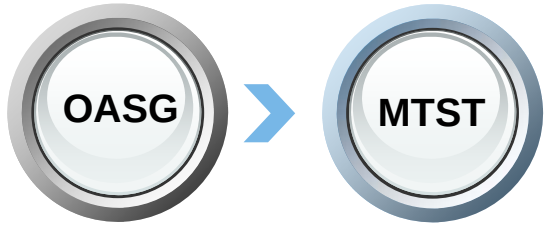


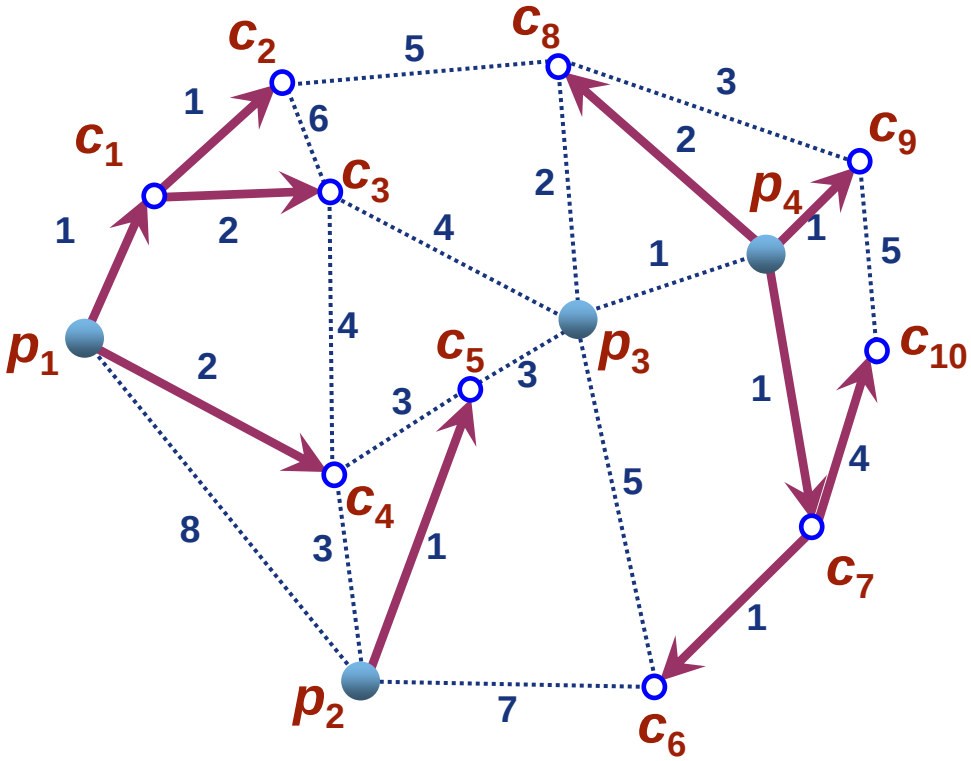

1. Add a dummy node and zero-weighted edges
2. Construct shortest path tree using Dijkstra's Alg.

# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)

**Can we do better than $O(n^2 \log n)$?**



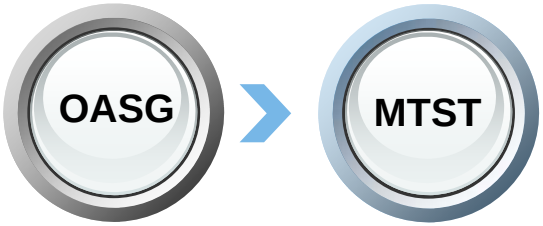



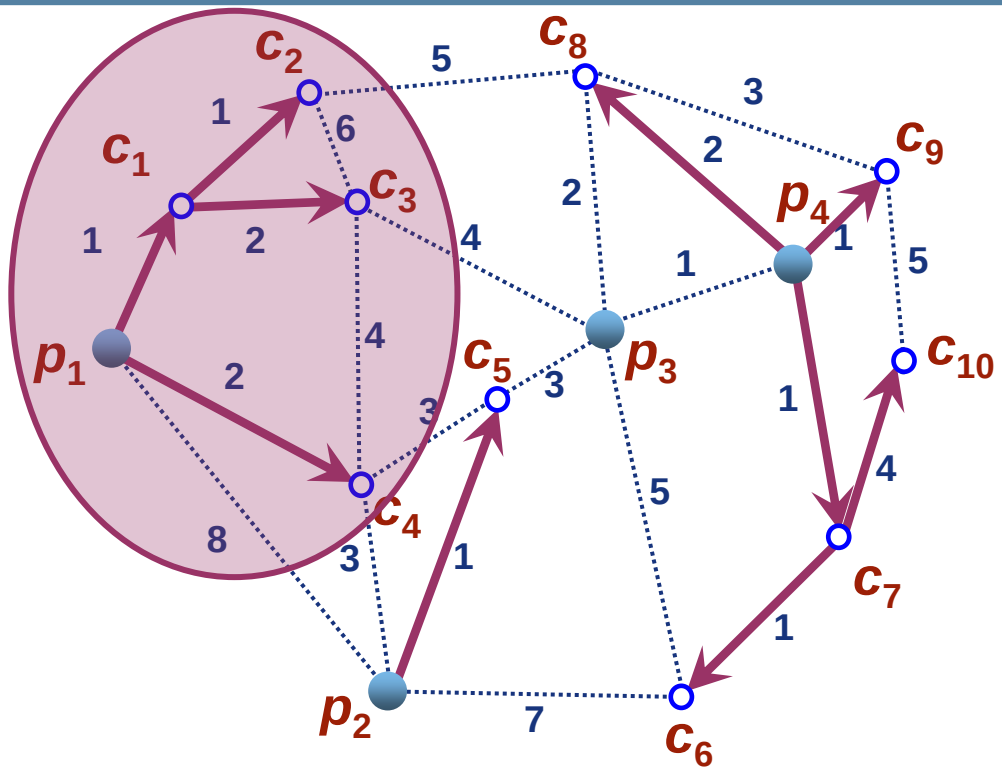

1. Add a dummy node and zero-weighted edges
2. Construct shortest path tree using Dijkstra's Alg.
3. Remove the dummy node and edges

# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)

**Can we do better than $O(n^2 \log n)$?**



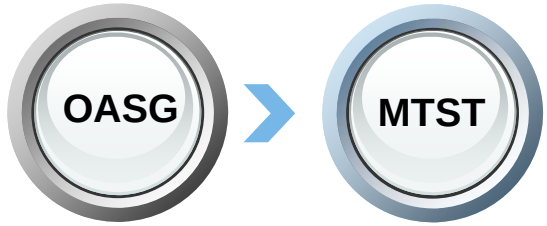



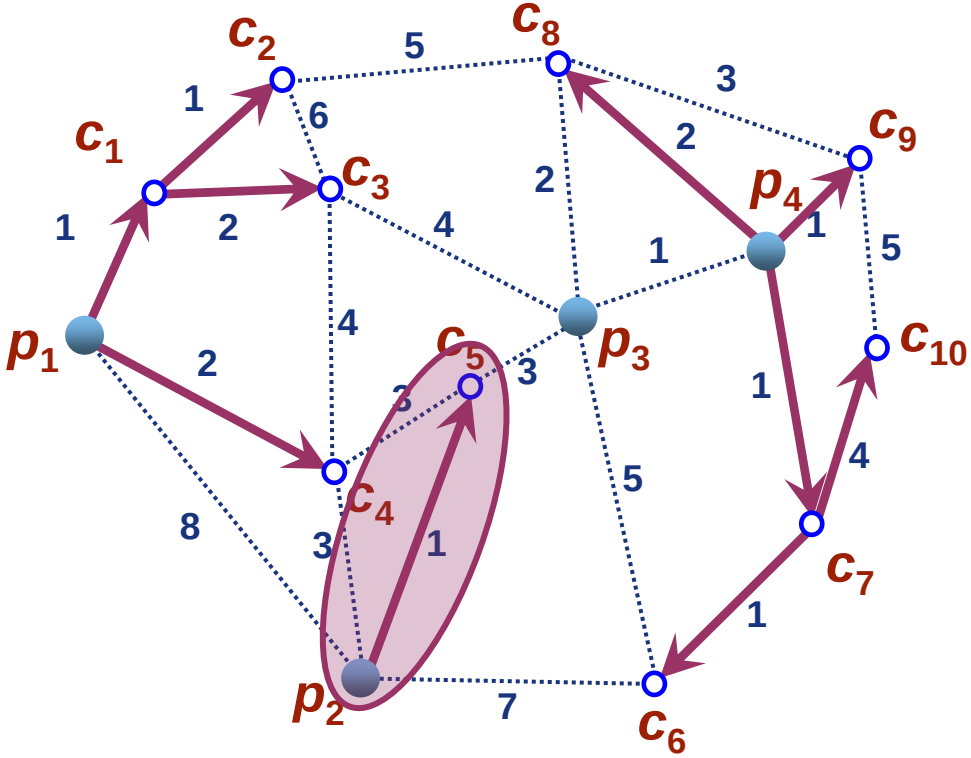

1. Add a dummy node and zero-weighted edges
2. Construct shortest path tree using Dijkstra's Alg.
3. Remove the dummy node and edges
4. Treat each tree as a super node and apply Kruskal's Alg.

# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)

**Can we do better than $O(n^2 \log n)$?**



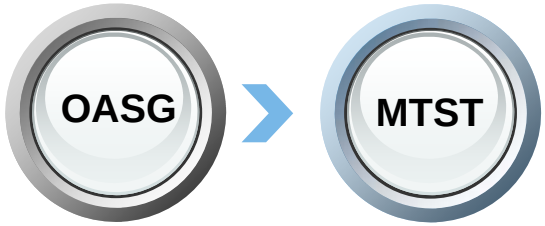



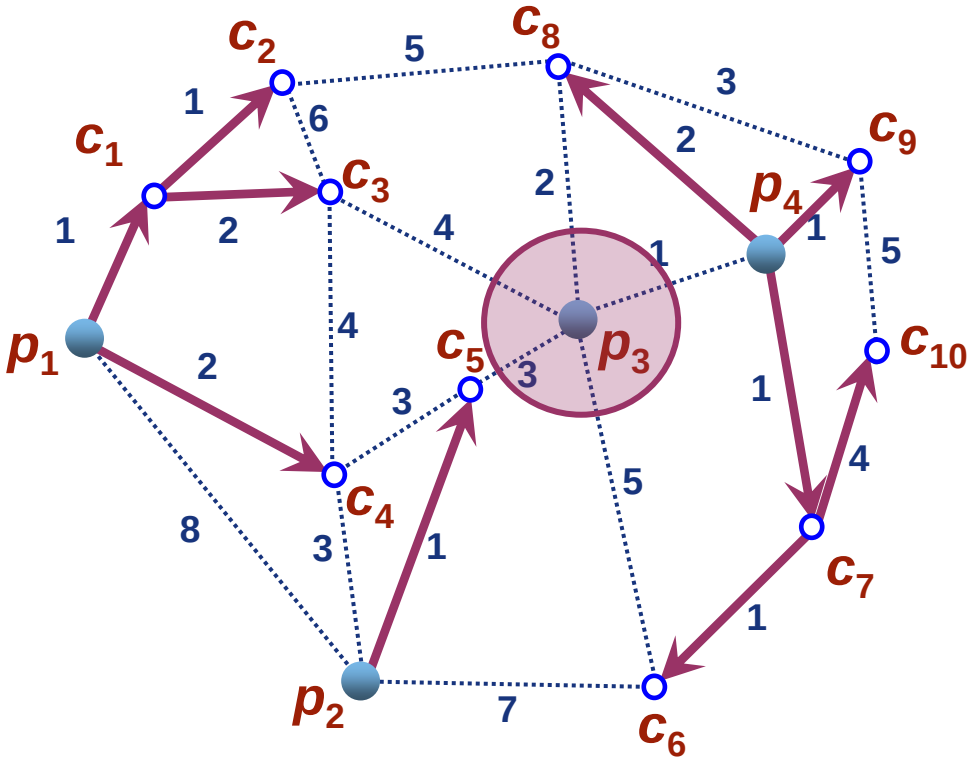

1. Add a dummy node and zero-weighted edges
2. Construct shortest path tree using Dijkstra's Alg.
3. Remove the dummy node and edges
4. Treat each tree as a super node and apply Kruskal's Alg.

# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)

**Can we do better than $O(n^2 \log n)$?**



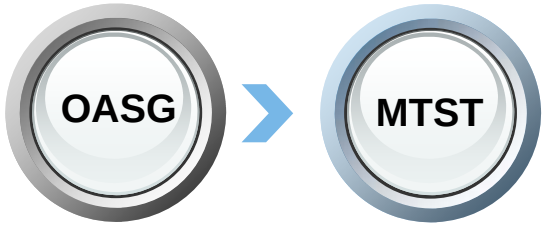



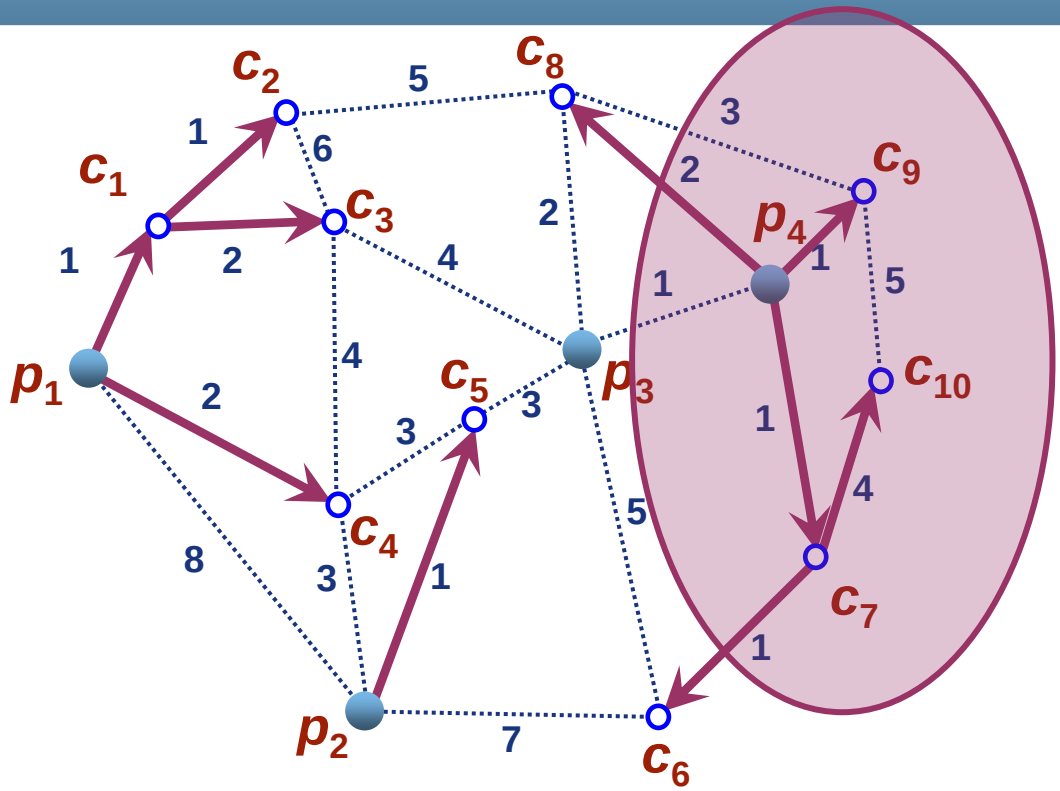

1. Add a dummy node and zero-weighted edges
2. Construct shortest path tree using Dijkstra's Alg.
3. Remove the dummy node and edges
4. Treat each tree as a super node and apply Kruskal's Alg.

# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)

**Can we do better than $O(n^2 \log n)$?**

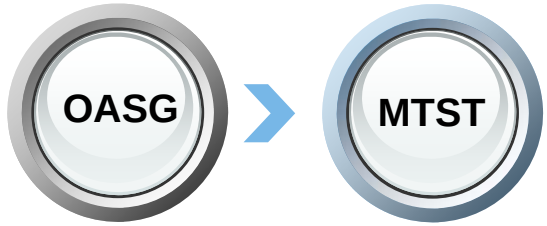


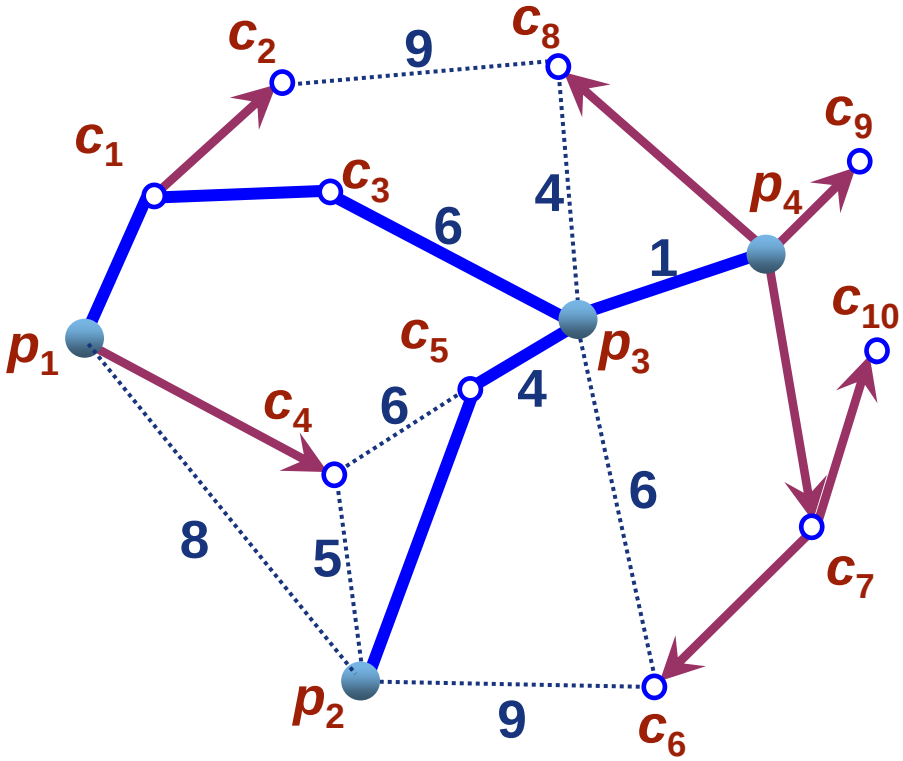

1. Add a dummy node and zero-weighted edges
2. Construct shortest path tree using Dijkstra's Alg.
3. Remove the dummy node and edges
4. Treat each tree as a super node and apply Kruskal's Alg.

# Step 2: MTST Construction

## Minimum Terminal Spanning Tree (MTST)




**Can we do better than $O(n^2 \log n)$?**

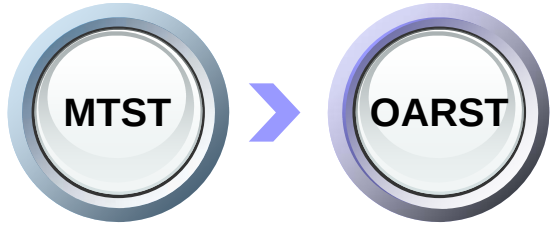



**Time Complexity: $O(n \log n)$**

- **Dijkstra's and Kruskal's Alg. on a sparse graph**

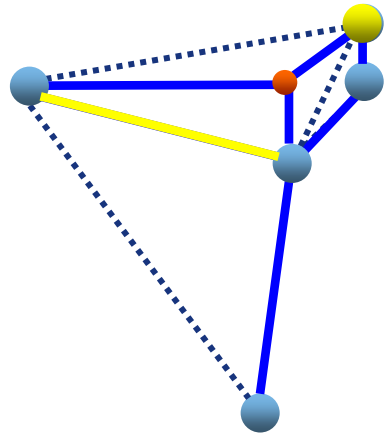
# Step 3: Edge-Based Refinement

**Obstacle-Avoiding Rectilinear Steiner Tree (OARST)**



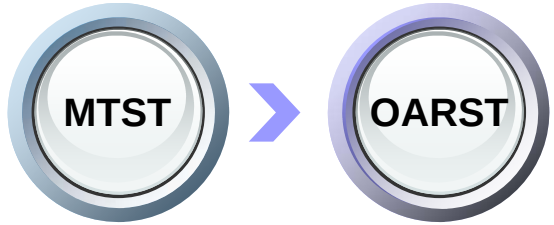

## From MTST to OARST

- **Obstacle-Free:**
  - consider vertex-edge pairs for tree length reduction ($O(n^2)$ pairs) [**Borah et al.**]

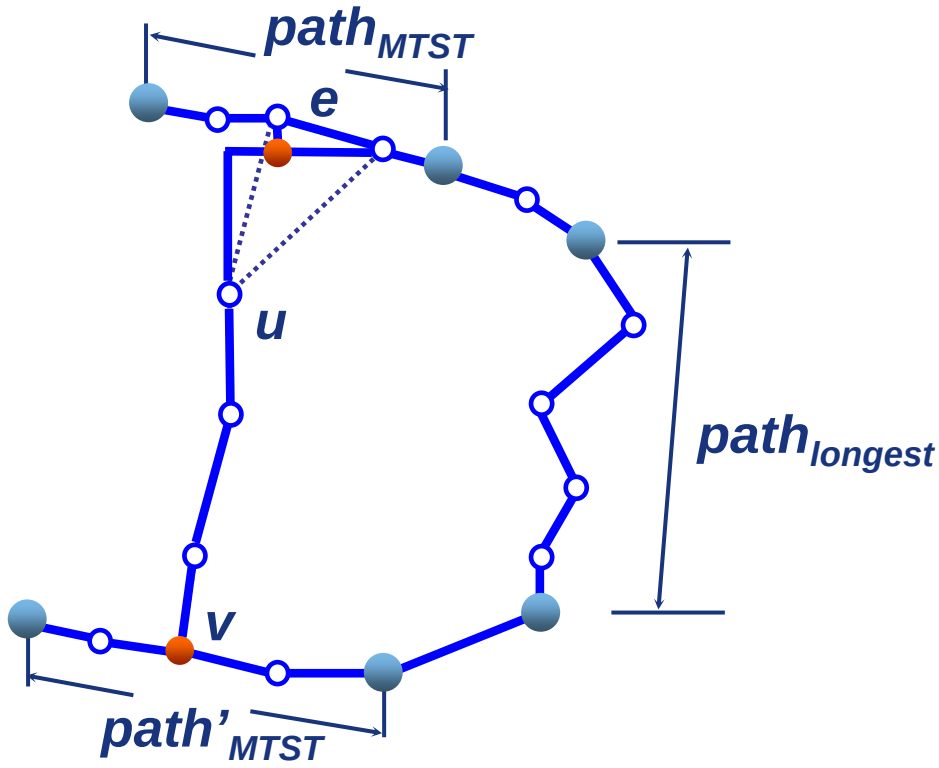


# Step 3: Edge-Based Refinement

## Obstacle-Avoiding Rectilinear Steiner Tree (OARST)



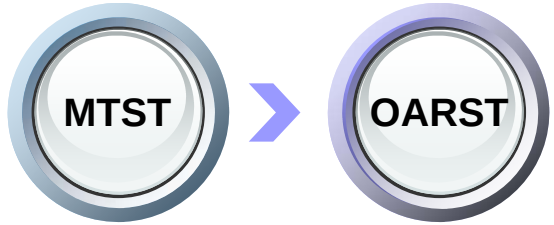

### Handling Obstacles




- Consider each neighboring vertex-edge pair (*u*, *e*) where *e* is on the MTST
- Find out *v*, the closest on-MTST vertex to u (Extended-Dijkstra in Step 2 can be reused here to identify all such <*u*, *v*> pairs)
- Add Steiner points, connect path <*u*, *v*>
- Remove *e* and $path_{longest}$ ($path_{longest}$ for all (*u*, *e*) pair can be identified by Tarjan's offline least common ancestor alg. )

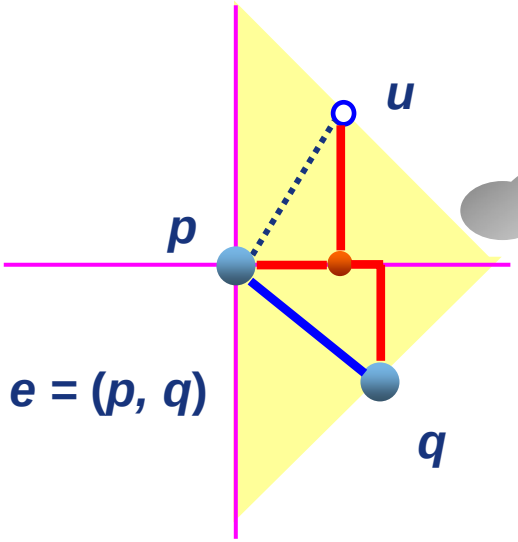
# Step 3: Edge-Based Refinement

## Obstacle-Avoiding Rectilinear Steiner Tree (OARST)




### Handling Obstacles

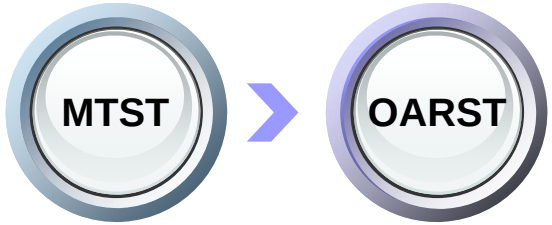
**Q: Do the newly added edges intersect with the obstacles?**




**No. Because there should not be any vertex within the yellow area**

# Complexity of the Algorithm

## Obstacle-Avoiding Rectilinear Steiner Tree (OARST)




### Time Complexity Analysis

**OASG Generation:** Sweeping line alg. with efficient active set implementation ($O(n\log n)$)

**MTST Construction:** Dijkstra's and Kruskal's Alg. on sparse graphs ($O(n\log n)$)

**Edge-Based Refinement:** Consider only O($n$) vertex-edge pairs ($O(n\log n)$)

**Step 1: $O(n\log n)$**

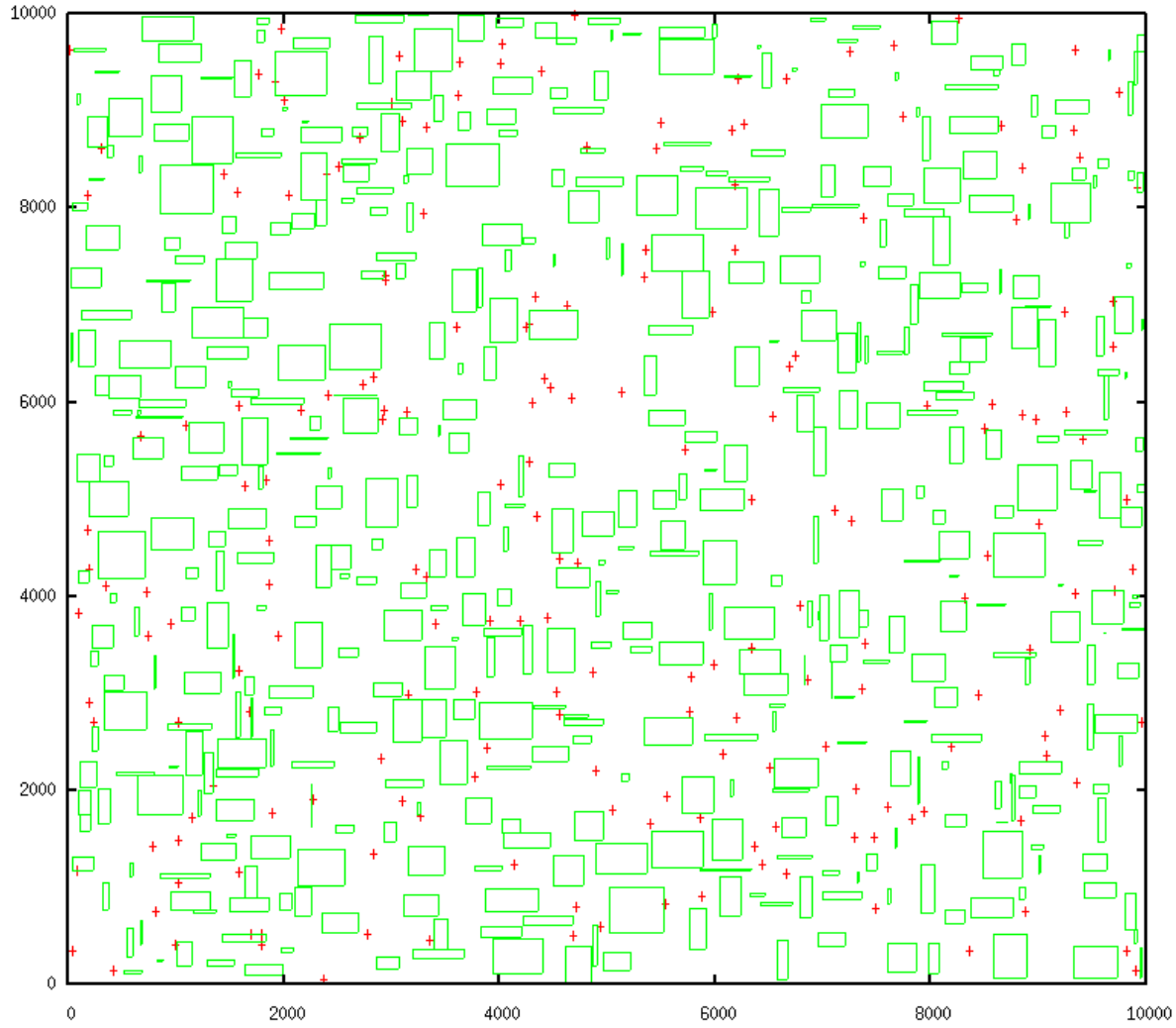
**Step 2: $O(n\log n)$**

**Step 3: $O(n\log n)$**

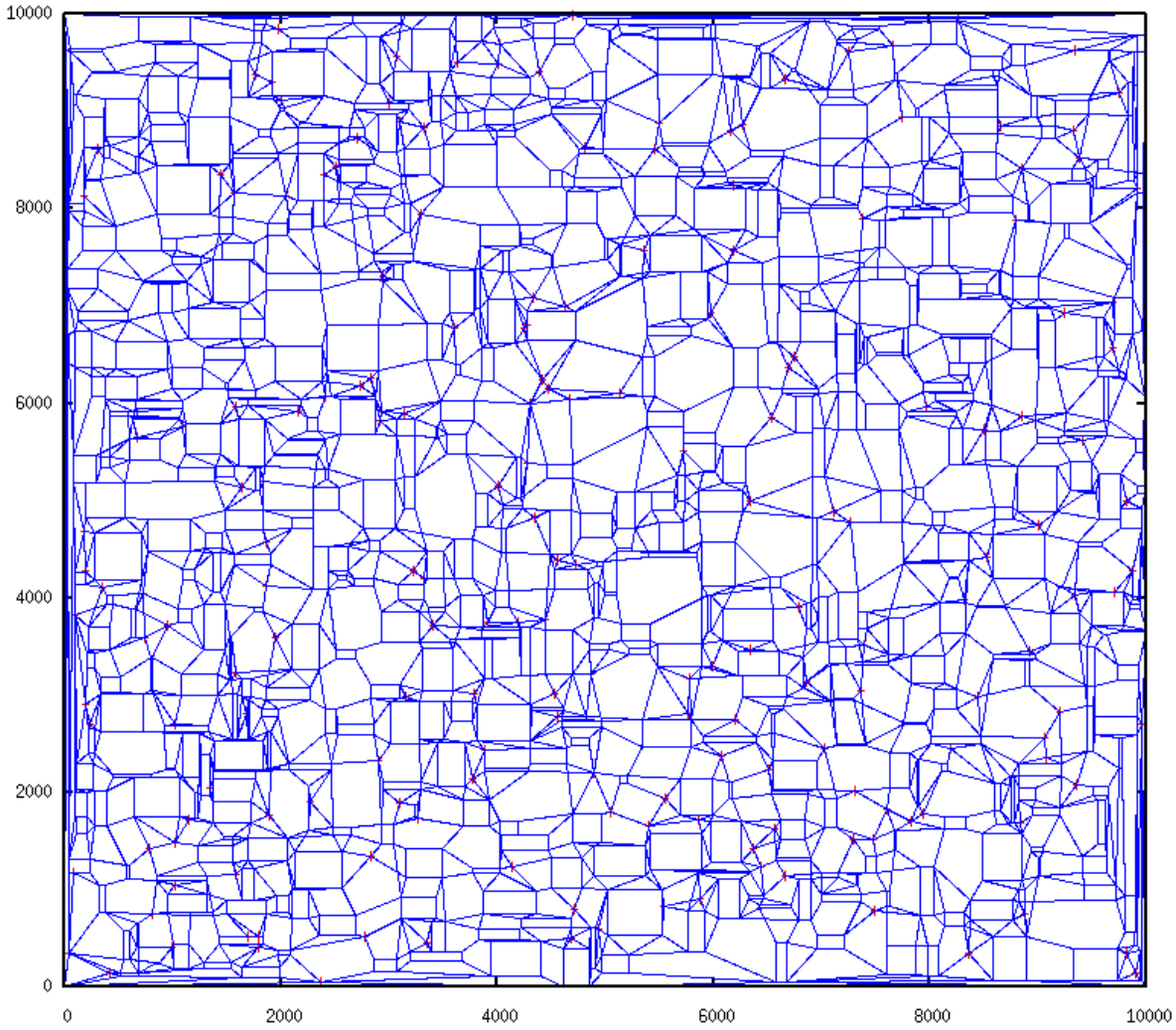
Entire Alg.: $O(n\log n)$

# Contents

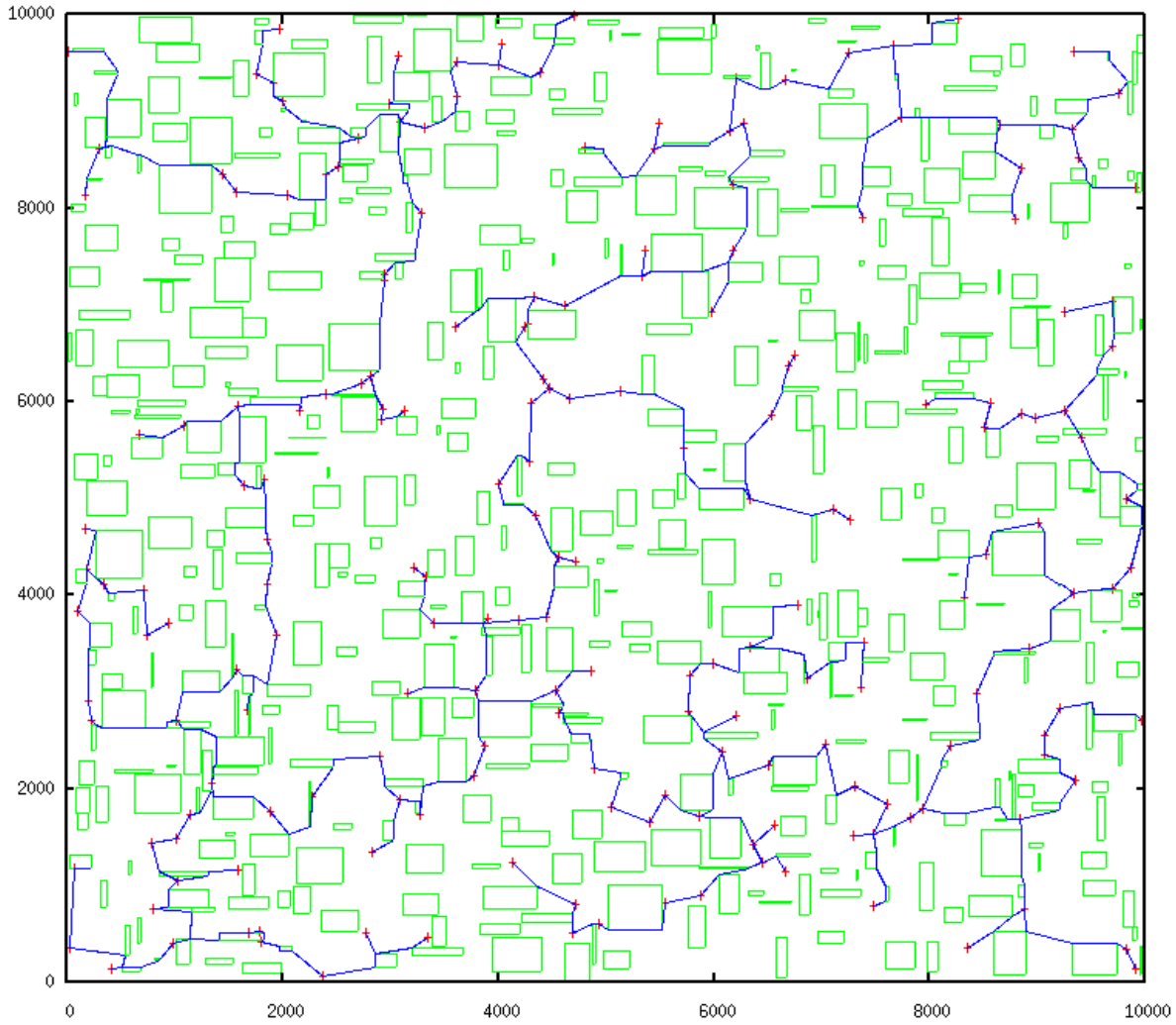
# Experimental Results



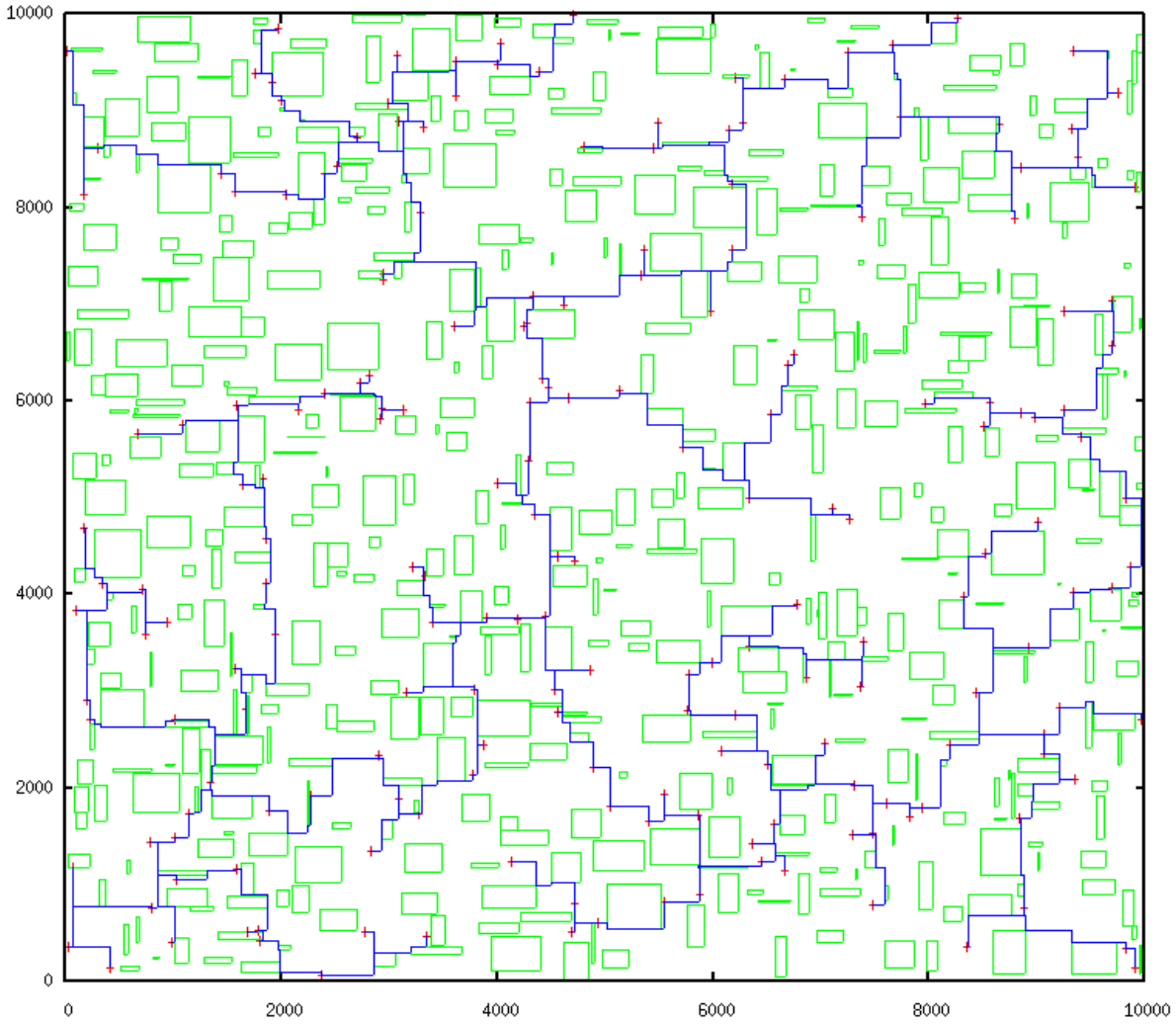
# Experimental Results



# Experimental Results



# Experimental Results



# Experimental Results

## Commonly Used Test Cases

| | m | k | Tree Length | | | | Running Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Feng's | Lin's | Ours | Impr% | Feng's | Lin's | Ours | Speedup |
| IND03 | 10 | 50 | - | 623 | 633 | -1.63% | 0.01 | 0.01 | 0.01 | 1X |
| IND04 | 25 | 79 | - | 1121 | 1131 | -0.89% | 0.02 | 0.02 | 0.02 | 1X |
| IND05 | 33 | 71 | - | 1392 | 1379 | -1.10% | 0.02 | 0.02 | 0.02 | 1X |
| RC10 | 500 | 100 | 198010 | 171519 | 168859 | 1.07% | 0.03 | 0.71 | 0.08 | 7.8X |
| RC11 | 1000 | 100 | 250570 | 237794 | 235795 | 0.35% | 0.04 | 0.33 | 0.15 | 8.5X |
| RC12 | 1000 | 10000 | 1723990 | 803483 | 852401 | -8.02% | 2.82 | 1.10 | 5.93 | 13.4X |

# Experimental Results

## Large Test Cases

| | m | k | Tree Length | | | | Running Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Feng's | Lin's | Ours | Impr% | Feng's | Lin's | Ours | Speedup |
| RL01 | 5000 | 5000 | - | 492865 | 504887 | -2.44% | - | 161.06 | 5.18 | 31.1X |
| RL02 | 10000 | 500 | - | 648508 | 641445 | 1.09% | - | 218.73 | 2.28 | 95.9X |
| RL03 | 10000 | 100 | - | 652241 | 644616 | 1.17% | - | 204.61 | 2.04 | 100.3X |
| RL04 | 10000 | 10 | - | 709904 | 701088 | 1.24% | - | 256.81 | 1.85 | 138.8X |
| RL05 | 10000 | 0 | - | 741697 | 731790 | 1.34% | - | 284.26 | 1.84 | 154.5X |

**Across a larger set of test cases, compared to Lin et al.'s heuristic, our algorithm achieves 25.8X speedup on average, while the length of the resulting OARSTs is only 1.58% larger on average**

# Conclusions

**Problem Formulation**

**Three-Step *O*(*n*log*n*) OARST Algorithm**

**OASG Generation** *O*(*n*log*n*)

**MTST Construction** *O*(*n*log*n*)

**Edge-Based Refinement** *O*(*n*log*n*)

**Experimental Results Illustrate the Effectiveness and Efficiency of Our Approach**

NORTHWESTERN
UNIVERSITY

# Thank You!

McCormick
Northwestern Engineering
Electrical Engineering and Computer Science