# A* Routing Steiner Trees

Kevin Morgan
ECE 6133
Georgia Institute of Technology

# A* Search Basics

- Heuristic Graph Search Algorithm
- Basic Idea:
  - Expand search nodes based on heuristic measurement of cost to a target
  - If a less costly path than the one currently be searched is found, begin searching it

$$f_{\text{cost}} = g + h$$

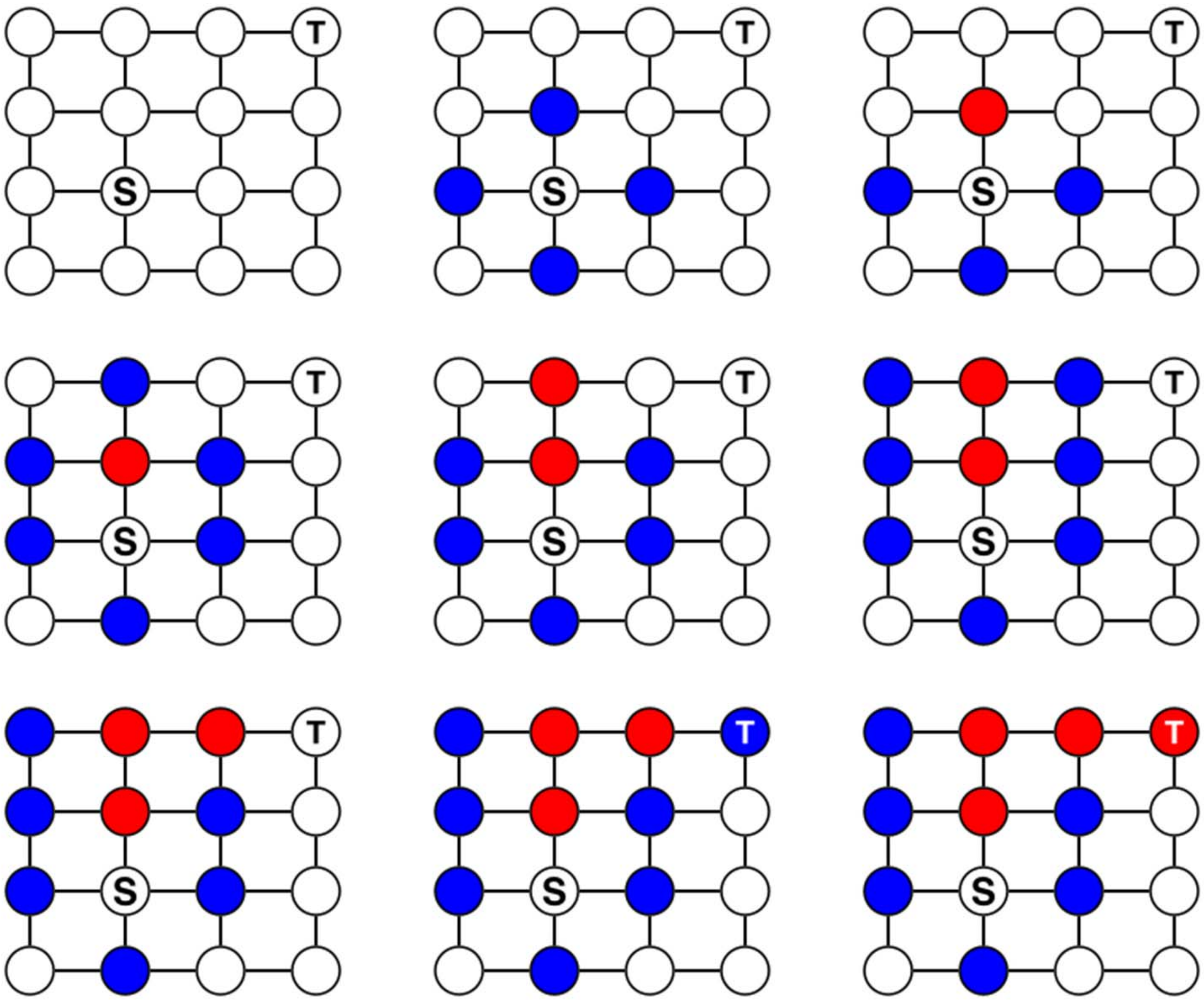
g = cost to get from source to current node
h = predicted cost from current node to target

# A* Algorithm Details (2-Terminal)

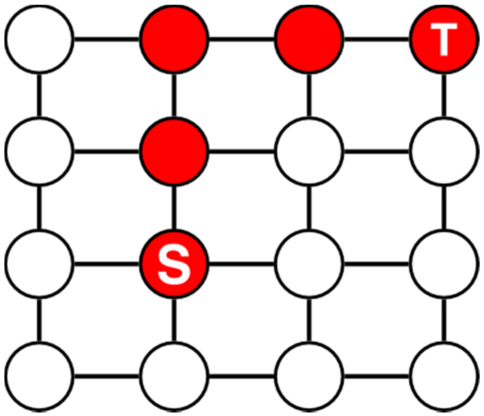
```
openSet = priority queue on f cost
s = source node
t = target node
s.f = g + h(s, t) // g = 0
while(openSet is not empty):
    v = openSet.pop() // item with lowest f cost
    if(v is closed):
          continue // already been searched
    if(v == t):
          // found a target, backtrace through parents to source
          return backtrace(t)
    v.closed = true
    for each neighbor, n, of v:
          if(n cannot be searched):
                    continue // obstacles
          n.parent = v
          n.g = v.g + dist(n, v)
          n.h = h(n, t) // distance from n to t
          n.f = n.g + n.h
          openSet.add(n)

// If open set is empty and we haven't found the target
return failure
```

# A* Algorithm Example

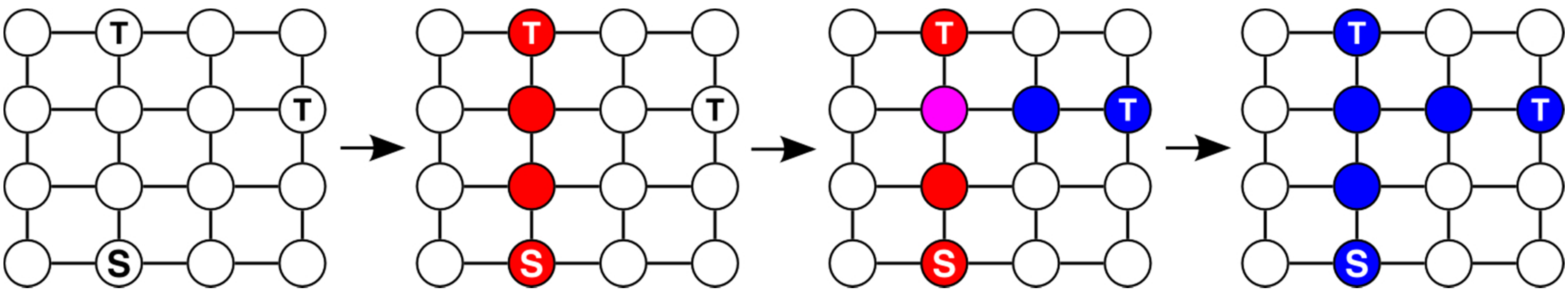



Result



# A* Algorithm for Multiple Terminals

- A* algorithm can be extended to routing multiple terminals:
  1. Route the source to an initial target
  2. For all remaining targets connect the closest target to the tree to the closest point on the tree

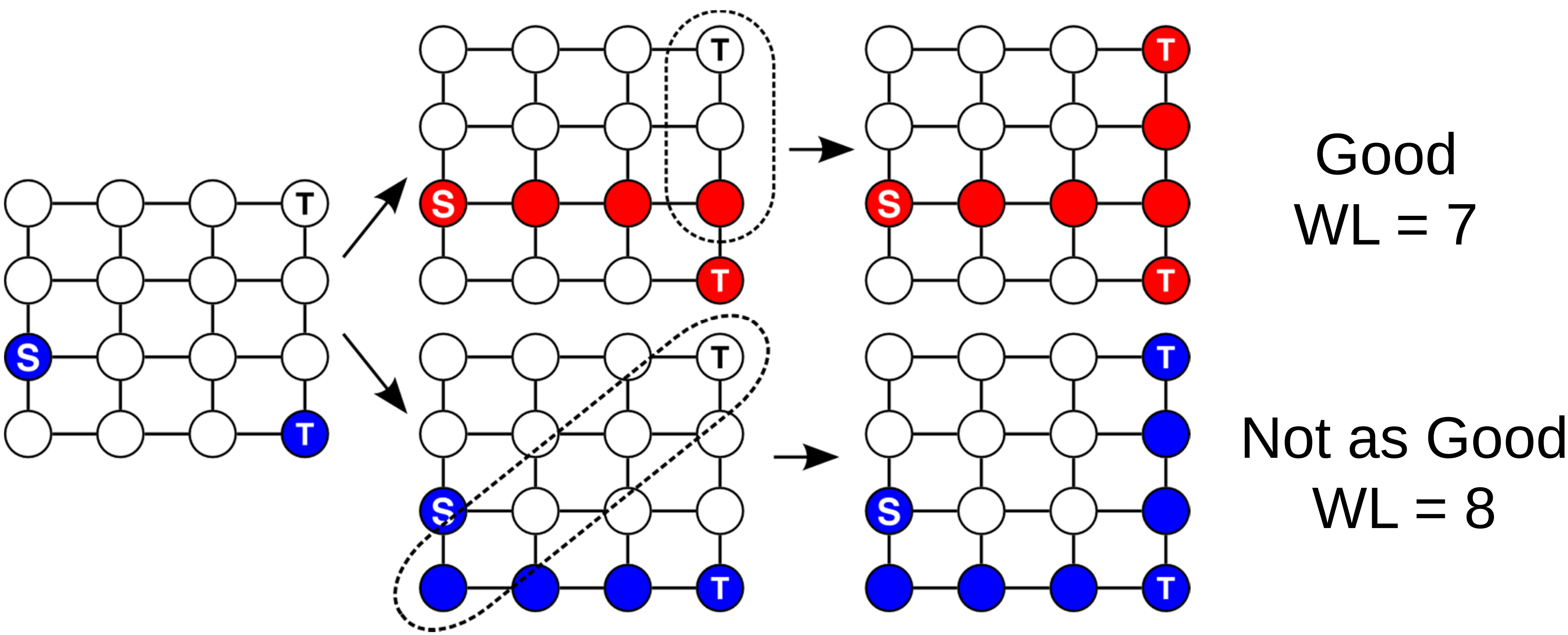


# Improving A*-Multi Results

- A* Multi-terminal algorithm can be extended to achieve “better” or morphed trees
- Critical Nodes:
  - Delay/Wirelength improvement [1]
  - Simply the order in which nodes are routed - more critical nodes are routed first and generally have the “best” paths
  - Implications when considering the delay of a route
- Biasing:
  - Wirelength improvement technique [1]
  - “Exact” biasing [1] based on a center of mass measurement
    - Relatively expensive calculation
  - My Improvement: Approximate biasing based on relative node counts
    - Advantage - can be precomputed for a given tree

# Biasing

- Method for breaking critical ties - when f cost and path length are both the same when comparing two nodes
- “Exact” biasing is ~O(3*t) for every expanded node [1]
- Approximate biasing is a one time computation cost of $O(t^2)$
  - Can bound radius though it does not improve run time

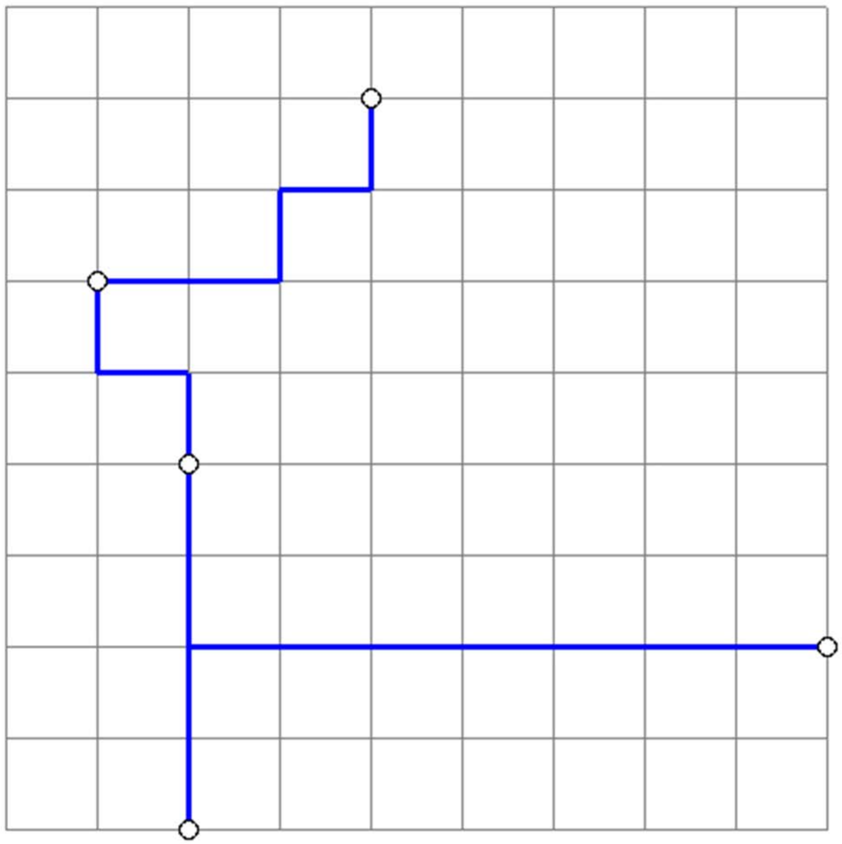


# A* Router Implementation

- Programmed in C
  - ~2500 Lines of Code
  - Major Data Structures:
    - Set: Hash Table based
    - Priority Queue: Binary (Min) Heap based
- Supports:
  - Routing on uniform and Hanan grids
  - Adding random obstacles to a uniform grid
  - Biasing and critical nodes
- Requires libpng
- Data taken on version compiled with -O3 flag

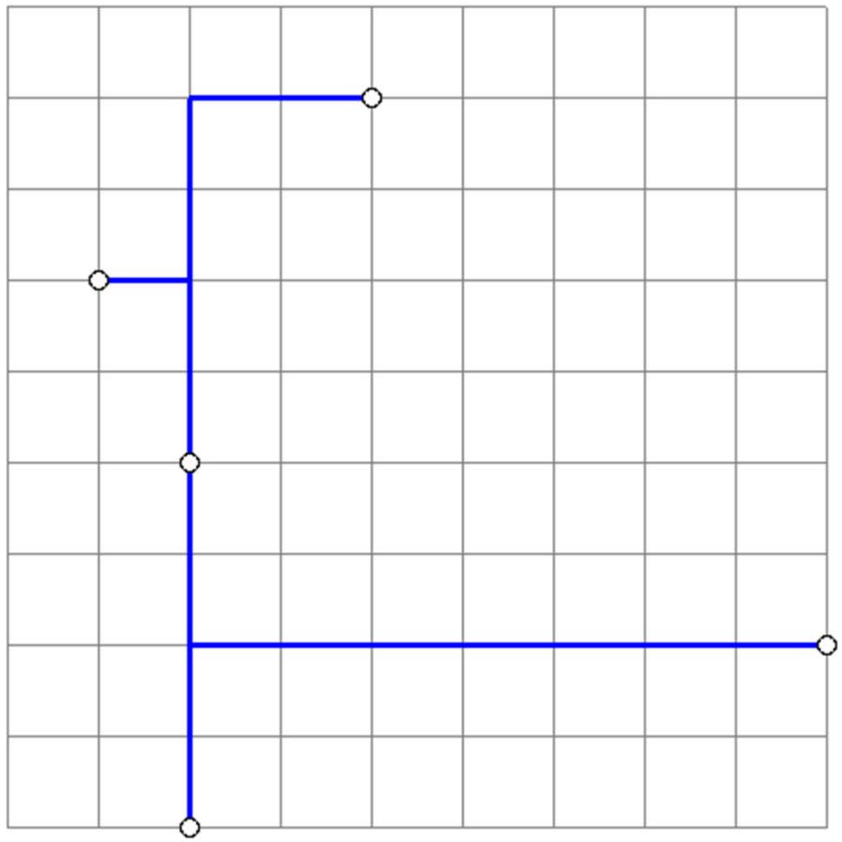
# Example Routes: 5 Nodes, 10x10 Grid

Uniform Grid
Exact Biasing
No Critical Nodes



WL = 19
Runtime = 0.270 ms
Routing: 0.170 ms
Grid Construction: 0.100 ms

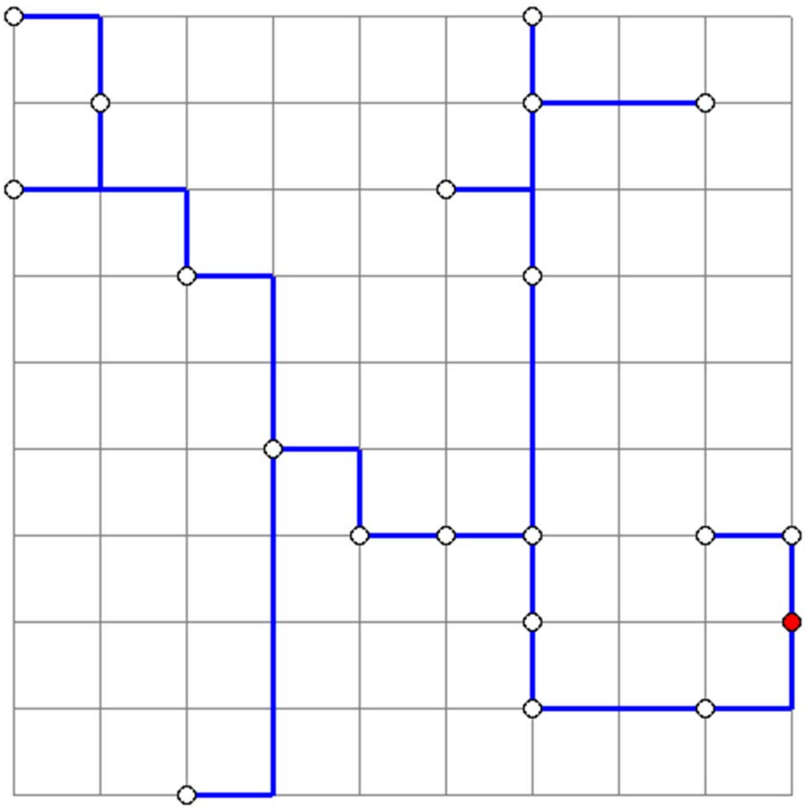
**Hanan Grid**
Exact Biasing
No Critical Nodes



WL = 18
Runtime = 0.145 ms
Routing: 0.100 ms
Grid Construction: 0.045 ms

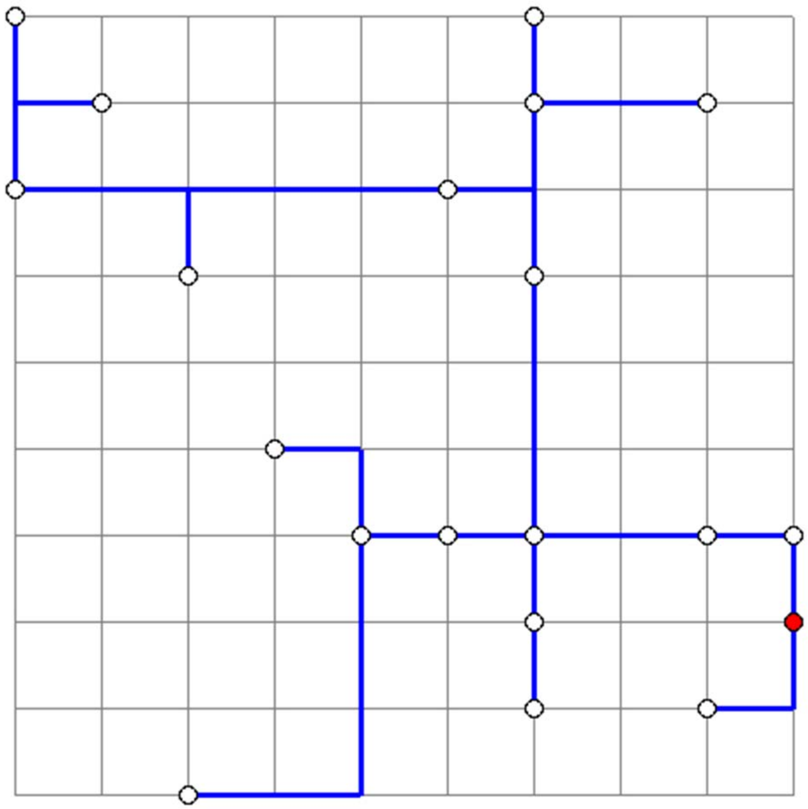
# Example Routes: 20 Nodes, 10x10 Grid

Uniform Grid
Exact Biasing
No Critical Nodes



WL = 35
Runtime = 0.820 ms
Routing: 0.720 ms
Grid Construction: 0.100 ms

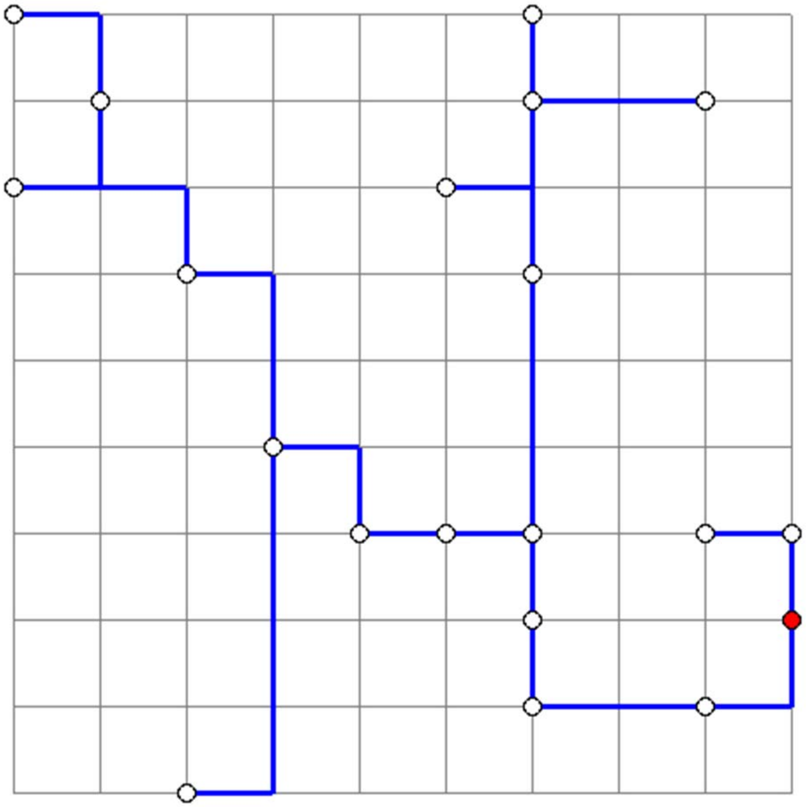
Uniform Grid
Exact Biasing
**Random Critical Nodes**



WL = 35
Runtime = 0.675 ms
Routing: 0.575 ms
Grid Construction: 0.100 ms

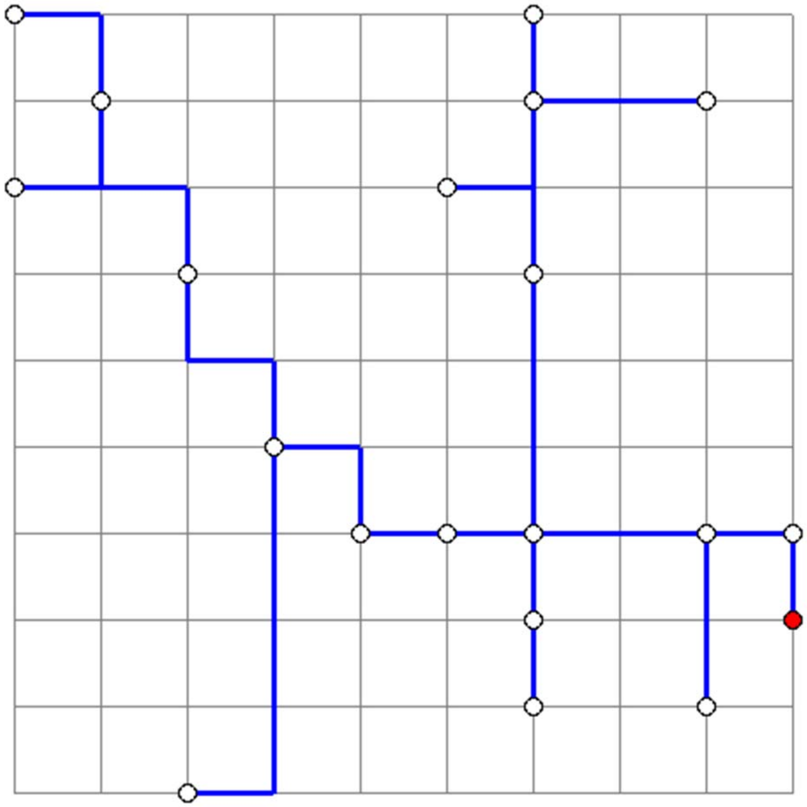
# Example Routes: 20 Nodes, 10x10 Grid

Uniform Grid
Exact Biasing
No Critical Nodes



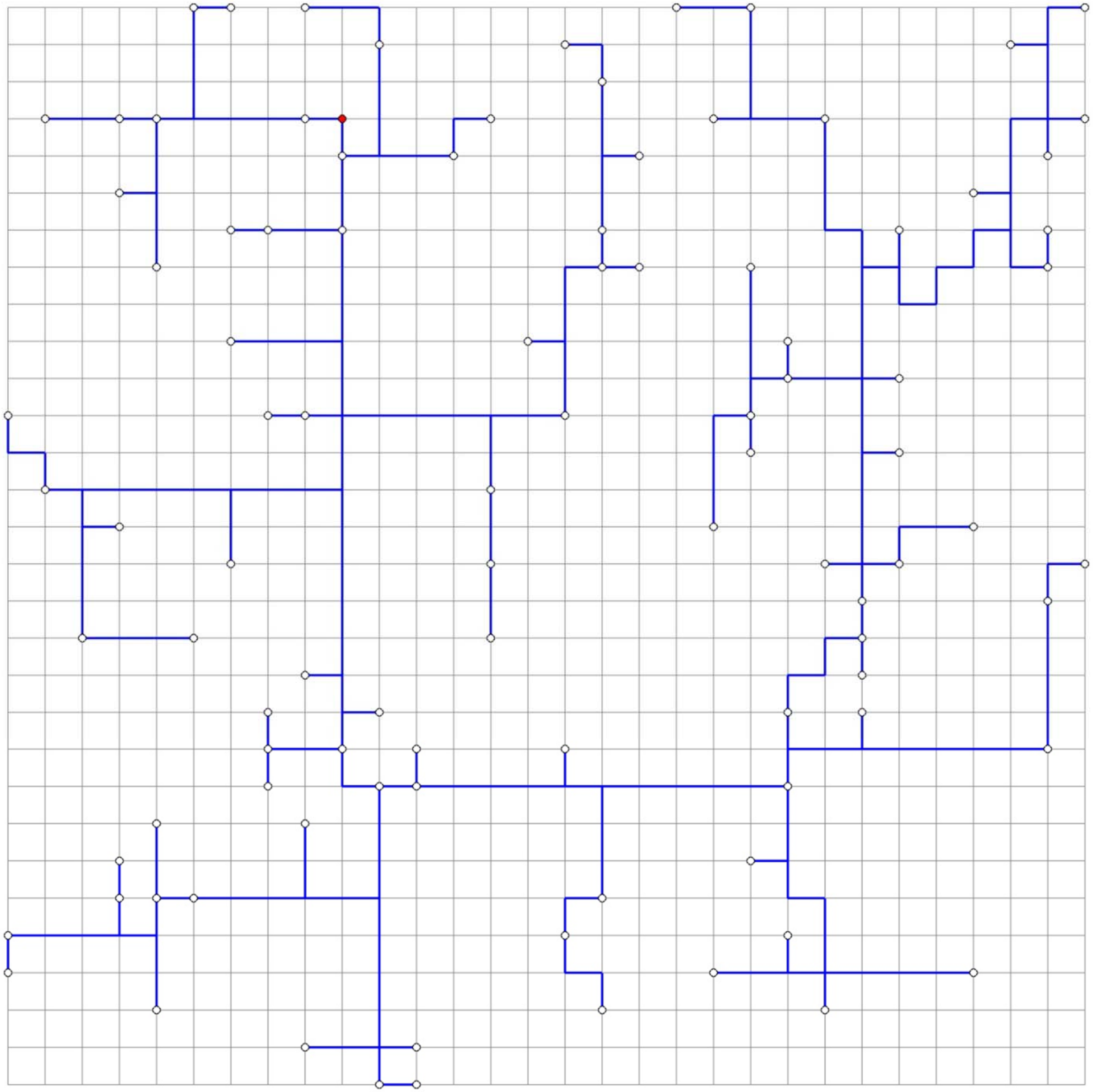
WL = 35
Runtime = 0.820 ms
Routing: 0.720 ms
Grid Construction: 0.100 ms

Uniform Grid
**Approximate Biasing**
No Critical Nodes



WL = 35
Runtime = 0.605 ms
Routing: 0.505 ms
Grid Construction: 0.100 ms

# Example Routes: 100 Nodes, 30x30 Grid

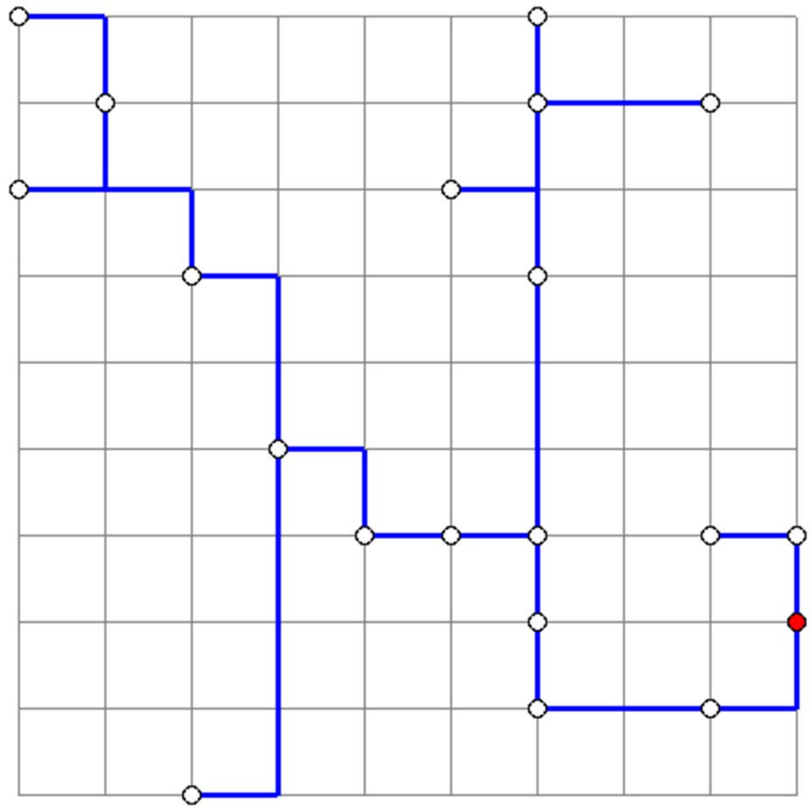


Uniform Grid
Approximate Biasing
Random Critical Nodes

WL = 265
Runtime = 9.94 ms
Routing: 9.10 ms
Grid Construction: 0.840 ms

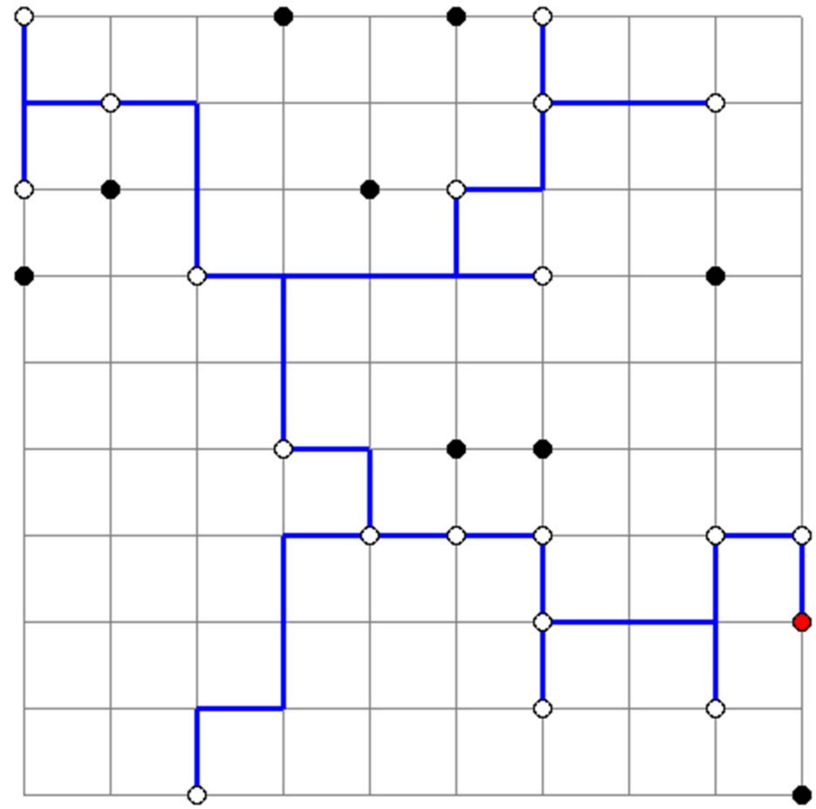
# Example Routes: 20 Nodes, 10x10 Grid, Random Obstacles

Uniform Grid
Exact Biasing
No Critical Nodes



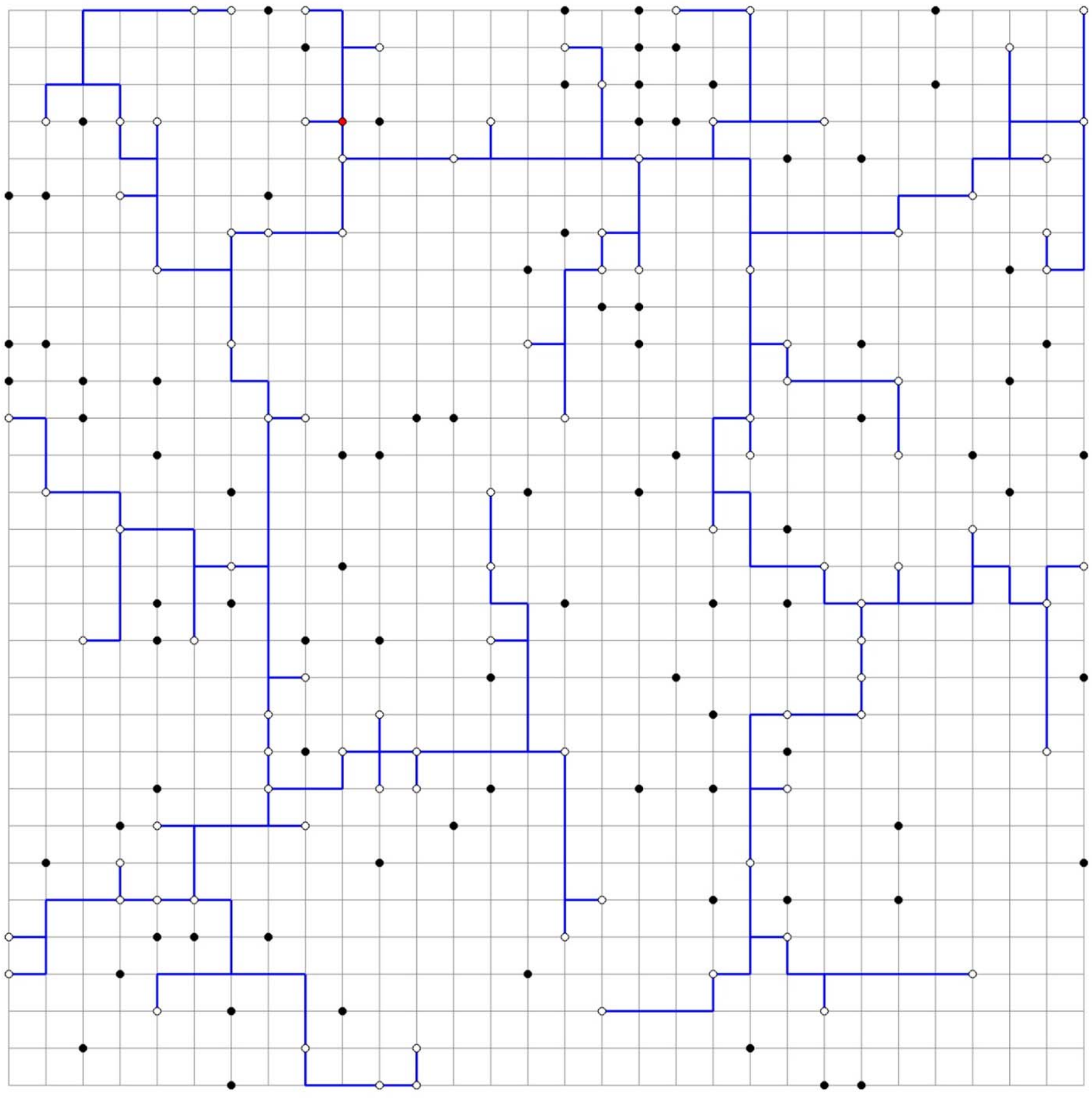
WL = 35
Runtime = 0.820 ms
Routing: 0.720 ms
Grid Construction: 0.100 ms

Uniform Grid
Exact Biasing
No Critical Nodes



WL = 36
Runtime = 0.884 ms
Routing: 0.784 ms
Grid Construction: 0.100 ms

# Example Routes: 100 Nodes, 30x30 Grid, Random Obstacles



Uniform Grid
Approximate Biasing
Random Critical Nodes

WL = 261
Runtime = 213.7 ms
Routing: 212.9 ms
Grid Construction: 0.825 ms

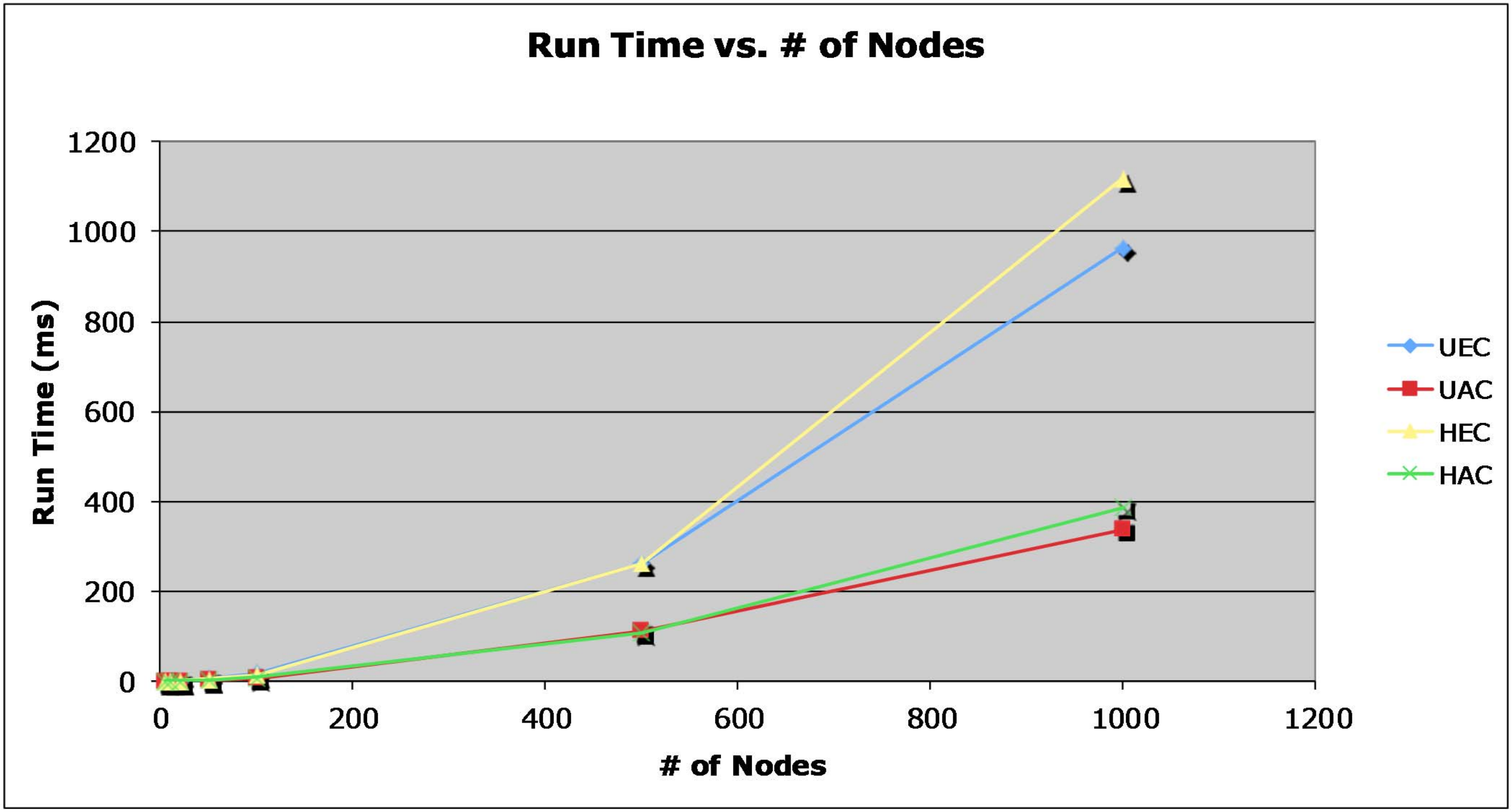
# Runtime Performance

| # of Nodes | Average Runtime (UEC) | Average Runtime (UAC) | Average Runtime (HEC) | Average Runtime (HAC) |
|---|---|---|---|---|
| 5 | 0.110 ms | 0.100 ms | 0.066 ms | 0.053 ms |
| 10 | 0.250 ms | 0.170 ms | 0.135 ms | 0.132 ms |
| 20 | 0.500 ms | 0.320 ms | 0.443 ms | 0.283 ms |
| 50 | 5.58 ms | 2.81 ms | 4.53 ms | 2.75 ms |
| 100 | 17.3 ms | 5.99 ms | 13.78 ms | 7.98 ms |
| 500 | 262 ms | 111 ms | 262 ms | 108 ms |
| 1000 | 965 ms | 337 ms | 1120 ms | 386 ms |

U: Uniform Grid
H: Hanan Grid
E: Exact Bias
A: Approximate Bias
C: Random Critical Nodes

Times are average runtimes (grid generation and routing) of many random node sets

# Runtime Performance



# Questions?

# References

[1] R. Hentschke, J. Narasimhan, M. Johann, and R. Reis, “Maze Routing Steiner Trees with Delay Versus Wire Length Tradeoff,” in IEEE Transactions on VLSI Systems, vol. 17, no. 8, pp. 1073 – 1086, August 2009.