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 being 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)

def AStarAlgorithm(s, t):
    openSet = priority_queue()  # open set
    s = Node(s)  # source node
    s.f = 0  # initial cost
    openSet.put(s)
    while not openSet.empty():  # while open set is not empty
        v = openSet.get()  # get lowest cost item
        if closed(v):  # if already searched
            continue
        if v == t:  # found target
            return traceback(t)
        v.c = True  # mark as closed
        for n in v.neighbors:  # for each neighbor n of v
            if obstacles(n):  # if obstacle
                continue
            n = Node(n)  # create node
            n.p = v  # set parent
            n.g = v.g + dist(n, v)  # update cost
            n.h = h(n, t)  # update heuristic
            n.f = n.g + n.h  # update total cost
            openSet.put(n)
    return failure  # if open set is empty and we haven't found the target
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 \( \sim O(3^t) \) 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

Good
WL = 7

Not as Good
WL = 8
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

Hanan Grid
Exact Biasing
No Critical Nodes

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

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 = 0.605 ms
Routing: 0.505 ms
Grid Construction: 0.100 ms

Uniform Grid
Approximate Biasing
No Critical Nodes

WL = 35
Runtime = 0.820 ms
Routing: 0.720 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

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

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

Run Time vs. # of Nodes

- UEC
- UAC
- HEC
- HAC

# of Nodes

Run Time (ms)
Questions?
References