Steve Tanimoto’s latest A*

1. For the start state $s_0$, compute $f(s_0) = g(s_0) + h(s_0) = h(s_0)$ and put $[s_0, f(s_0)]$ on a list (priority queue) OPEN.

2. If OPEN is empty, output “DONE” and stop.

3. Find and remove the item $[s, p]$ on OPEN having lowest $p$. Break ties arbitrarily.
   
   Put $[s, p]$ on CLOSED.
   
   If $s$ is a goal state: output its description (and backtrace a path), and if $h$ is known to be admissible, halt.

4. Generate the list $L$ of $[s', f(s')]$ pairs where the $s'$ are the successors of $s$ and their $f$ values are computed using $f(s') = g(s') + h(s')$.

   Consider each $[s', f(s')]$.
   
   If there is already a pair $[s', q]$ on CLOSED (for any value $q$):
      
      if $f(s') > q$, then remove $[s', f(s')]$ from $L$.
      
      If $f(s') \leq q$, then remove $[s', q]$ from CLOSED.
   
   Else if there is already a pair $[s', q]$ on OPEN (for any value $q$):
      
      if $f(s') > q$, then remove $[s', f(s')]$ from $L$.
      
      If $f(s') \leq q$, then remove $[s', q]$ from OPEN.

5. Insert all members of $L$ onto OPEN.

6. Go to Step 2.
Thought Question

- Do you have to keep the list of successors for each node through the whole search?
- Rich/Knight did (why?)
- Tanimoto did not
- If you keep it, what might it be used for?
A* Extra Examples

• To show what happens when
  1. It encounters a node whose state is already on OPEN
  2. It encounters a node whose state is already on CLOSED
A* Example

• Newly generated node s, but OLD on OPEN has the same state.
• Shortest path in Romania, but the goal is now Giurgiu, not Bucharest.

**Straight line distances to Giurgiu (I made them up)**
- Arad 390
- Sibiu 275
- Fagaras 200
- Rimnicu 205
- Pitesi 125
- Craiova 120
- Bucharest 80
- Drobeta 240
Goal is Giurgiu

 Forget the other 2

Giurgiu
508=508+0
GOAL

Bucharest
498=418+80
BETTER

Craiova
575=455+120
WORSE

Drobeta
726=486+240
WORSE

Rimnicu
717=512+205
WORSE

Pitesi
629=504+125
WORSE

Bucharest
439=239+200

Fagaras
415=140+275

Sibiu
425=220+205

Rimnicu
486=366+120

Pitesi
530=450+80

OLD on OPEN

GOAL

Forget the other 2
A* Example (abstract, pretend it’s time)

- Newly generated node s, but OLD on CLOSED has the same state.
The Heuristic Function $h$

- If $h$ is a **perfect estimator** of the true cost then A* will always pick the correct successor with no search.

- If $h$ is **admissible**, A* with TREE-SEARCH is guaranteed to give the optimal solution.

- If $h$ is **consistent**, too, then GRAPH-SEARCH is optimal.

- If $h$ is not admissible, no guarantees, but it can work well if $h$ is not often greater than the true cost.
Complexity of A*

- Time complexity is exponential in the length of the solution path unless for “true” distance $h^*$
  
  $$|h(n) - h^*(n)| < \Theta(\log h^*(n))$$

  which we can’t guarantee.

- But, this is AI, computers are fast, and a good heuristic helps a lot.

- Space complexity is also exponential, because it keeps all generated nodes in memory.

Big Theta notation says 2 functions have about the same growth rate.
Why not always use A*?

- Pros
- Cons
Solving the Memory Problem

- Iterative Deepening A*
- Recursive Best-First Search
- Depth-First Branch-and-Bound
- Simplified Memory-Bounded A*
Iterative-Deepening A*

- Like iterative-deepening depth-first, but...
- Depth bound modified to be an \textit{f-limit}
  - Start with \( f\text{-limit} = h(\text{start}) \)
  - Prune any node if \( f(\text{node}) > f\text{-limit} \)
  - Next \( f\text{-limit}=\text{min-cost of any node pruned} \)
Recursive Best-First Search

• Use a variable called \textit{f-limit} to keep track of the best alternative path available from any ancestor of the current node.

• If \( f(\text{current node}) > f\text{-limit} \), back up to try that alternative path.

• As the recursion unwinds, replace the f-value of each node along the path with the \textit{backed-up value}: the best f-value of its children.
Simplified Memory-Bounded A*

- Works like A* until memory is full
- When memory is full, drop the leaf node with the highest f-value (the worst leaf), keeping track of that worst value in the parent
- Complete if any solution is reachable
- Optimal if any optimal solution is reachable
- Otherwise, returns the best reachable solution
Performance of Heuristics

• How do we evaluate a heuristic function?

• effective branching factor $b^*$
  – If A* using $h$ finds a solution at depth $d$ using $N$ nodes, then the effective branching factor is $b^*$ where $N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$

• Example:

  depth 0
  depth 1
  depth 2

  $d=2$
  $b=3$
### Table of Effective Branching Factors

<table>
<thead>
<tr>
<th>b</th>
<th>d</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>364</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>88573</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>9331</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>72,559,411</td>
</tr>
</tbody>
</table>

How might we use this idea to evaluate a heuristic?
How Can Heuristics be Generated?

1. From *Relaxed Problems* that have fewer constraints but give you ideas for the heuristic function.

2. From *Subproblems* that are easier to solve and whose exact cost solutions are known.

The cost of solving a relaxed problem or subproblem is not greater than the cost of solving the full problem.
Still may not succeed

- In spite of the use of heuristics and various smart search algorithms, not all problems can be solved.

- Some search spaces are just too big for a classical search.

- So we have to look at other kinds of tools.
HW 2: A* Search

• A robot moves in a 2D space.
• It starts at a start point \((x_0,y_0)\) and wants to get to a goal point \((x_g,y_g)\).
• There are rectangular obstacles in the space.
• It cannot go THROUGH the obstacles.
• It can only move to corners of the obstacles, ie. search space limited.
How can the robot get from (0,0) to (9,6)?
What is the minimal length path?
More next time.