Informed Search

CISC481/681, Lecture #4
Ben Carterette

With material adapted from Prof. Marie desJardins (UMBC) and Kathy McCoy.

Quick Review

• Search problem formulation:
  – State space, initial state, successor function, path cost, goal test
• Search strategies defined by ordering of fringe
• Uninformed search strategies:
  – Breadth-first, uniform cost
  – Depth-first, depth-limited
  – Iterative deepening
• Tree-search and graph-search

Best-First Search

• Order fringe nodes \( n \) in increasing order of an evaluation function \( f(n) \)
  – Smaller \( f(n) \) suggests more promising node
• Best-first search describes a class of search strategies that work this way
  – Greedy best-first search
  – A* search
  – Uniform cost search?
• “Best” is limited by how good \( f(n) \) is. Optimality is not guaranteed!

Heuristic Functions

• A class of evaluation functions that use information about the problem domain
  – Not information about future possible states!
• \( h(n) = \) the estimated cost of the minimum-cost path from \( n \) to a goal state
  – \( h \) for heuristic
  – Estimate based on information computable from the current state description
Greedy Best-First Search

- Best-first search with \( f(n) = h(n) \)
  - i.e. order nodes on the fringe in increasing order of estimated distance from goal
  - Traveling Romania example: \( f(n) = \text{straight-line distance to Bucharest} \)
- Greedy best-first search expands the node that appears to be closest to the goal

Example: Traveling Romania

GBFS Performance

- Complete?
  - No: could get stuck in a loop and not find a solution
- Optimal?
  - No: the solution it finds is not necessarily best
  - E.g. Romania example
- Time and space complexity?
  - \( O(b^m) \) in the worst case
  - But a good heuristic function can ensure that the average case is much better

A* Search

- Idea: avoid expanding nodes that are already expensive
- Define evaluation function \( f(n) = g(n)+h(n) \)
  - \( g(n) = \text{cost from initial state to } n \)
  - \( h(n) = \text{estimated cost from } n \text{ to goal} \)
  - \( f(n) = \text{estimated total cost from start to finish while passing through } n \)
- Sort fringe nodes in increasing order of \( f(n) \)
Example: Traveling Romania

\[ f(n) = g(n) + h(n) \]
\[ g(n) \] is total cost from Arad to node \( n \)
\[ h(n) \] is straight-line distance from city node to Bucharest

A* Performance

- Complete?
  - Yes, if arc cost \( \geq \varepsilon > 0 \)
  - (Unless there are infinitely many nodes with \( f(n) \leq f(G) \))
- Optimal?
- Time and space complexity?

Admissible Heuristics

- Let \( h^*(n) \) be the true cost to reach the goal state from \( n \)
- Call a heuristic \( h(n) \) **admissible** if \( h(n) \leq h^*(n) \) for all \( n \)
- Admissible heuristics always **underestimate** the cost to reach the goal
  - They are **optimistic**
  - Romania example: straight-line distance is never more than actual road distance
Admissibility and Optimality

- Theorem: if \( h(n) \) is admissible, Tree-Search with A* is optimal

Why Tree Search?

- Tree search does not discard nodes that have already been expanded—so why use it?
- Let’s do graph search with an admissible heuristic

Consistent Heuristics

- Call a heuristic \( h(n) \) consistent if, for every node \( n \) and every successor node \( n' \) of \( n \) generated by action \( a \), \( h(n) \leq c(n,a,n')+h(n') \)
  - Note: every consistent heuristic is admissible
  - Not every admissible heuristic is consistent

Intuition: \( h \) gets more accurate as the search deepens

Consistency and Optimality

- Theorem: If \( h(n) \) is consistent, Graph-Search with A* is optimal
A* Performance

• Complete?
  – Yes, if arc cost \( \geq \epsilon > 0 \)
  – (Unless there are infinitely many nodes with \( f(n) \leq f(G) \))
• Optimal?
  – Yes!
• Time and space complexity?
  – Exponential
  – But this does not really tell us much about practicality

Choosing Heuristics

• A problem may have many possible consistent and/or admissible heuristic functions

  \[
  \begin{array}{ccc}
  7 & 2 & 4 \\
  5 & 5 & \text{Start Node} \\
  8 & 3 & 1 \\
  6 & 7 & 8 \\
  \end{array}
  \]

  \[
  \begin{array}{ccc}
  1 & 2 \\
  3 & 4 & 5 \\
  \text{Goal Node} \\
  \end{array}
  \]

• \( h_1(n) \) = number of misplaced tiles
• \( h_2(n) \) = total Manhattan distance of misplaced tiles to correct location
**Dominance**

- If \( h_1(n) \) and \( h_2(n) \) are both admissible heuristics, and \( h_2(n) \geq h_1(n) \) for all \( n \), then we say \( h_2 \) **dominates** \( h_1 \)
  - \( h_2 \) is “smarter” than \( h_1 \)
  - \( h_2 \) is “more informed” than \( h_1 \)
- **Theorem**: if \( h_2(n) \) dominates \( h_1(n) \), then the set of nodes expanded by \( A^* \) with \( h_2 \) is contained in the set of nodes expanded by \( A^* \) with \( h_1 \)

**Effective Branching Factor**

- To say \( A^* \) is exponential does not capture performance differences between heuristics
- If \( A^* \) with heuristic \( h(n) \) generates \( N \) nodes to find a solution at depth \( d \), the **effective branching factor** \( b^* \) is the branching factor that a tree of depth \( d \) would need in order to contain \( N+1 \) nodes
  - i.e. \( N+1 = 1 + b^* + (b^*)^2 + ... + (b^*)^d \)
  - (solve for \( b^* \))
- \( b^* \) is a good estimate of a heuristic’s performance

**Uniform-Cost Search Revisited**

- Uniform-cost search orders nodes on the fringe in increasing order of path cost
- Equivalent to \( A^* \) search with \( h(n) = 0 \)
  - \( f(n) = g(n) + h(n) = g(n) = \text{path cost} \)
- Therefore every heuristic with \( h(n) > 0 \) for some \( n \) is better than uniform-cost search

**Empirical A* Performance**

<table>
<thead>
<tr>
<th>( d )</th>
<th>IDS</th>
<th>A*/h_1</th>
<th>A*/h_2</th>
<th>IDS</th>
<th>A*/h_1</th>
<th>A*/h_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>2.45</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
<td>2.87</td>
<td>1.48</td>
<td>1.45</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
<td>2.73</td>
<td>1.34</td>
<td>1.30</td>
</tr>
<tr>
<td>8</td>
<td>6,384</td>
<td>39</td>
<td>25</td>
<td>2.80</td>
<td>1.33</td>
<td>1.24</td>
</tr>
<tr>
<td>10</td>
<td>47,127</td>
<td>93</td>
<td>39</td>
<td>2.79</td>
<td>1.38</td>
<td>1.22</td>
</tr>
<tr>
<td>12</td>
<td>3,644,035</td>
<td>227</td>
<td>73</td>
<td>2.78</td>
<td>1.42</td>
<td>1.24</td>
</tr>
<tr>
<td>14</td>
<td>3,644,035</td>
<td>539</td>
<td>113</td>
<td>2.78</td>
<td>1.44</td>
<td>1.23</td>
</tr>
</tbody>
</table>
Devising Better Heuristics

- The Manhattan distance heuristic for the 8-puzzle still ignores a lot of information
  - Like the fact that tiles will obstruct each other
- Think of the heuristic as breaking a problem down into “relaxed” subproblems
- More complex subproblems can result in better heuristics

8-Puzzle Heuristics

- \( h_1 \): number of out-of-place tiles
  - Subproblem: pull each tile out and put it in the right place
- \( h_2 \): Manhattan distance to correct location
  - Subproblem: move each tile one square at a time to the right place
- \( h_3 \): Number of moves required to get disjoint pairs of tiles to the right places
  - E.g. \( h_3(n) = d_{12} + d_{34} + d_{56} + d_{78} \)
- \( h_4 \): Number of moves required to get disjoint sets of 4 tiles to the right places
  - E.g. \( h_4(n) = d_{1234} + d_{5678} \)

8-Puzzle Heuristic \( h_4(n) \)

A* Summary

- Fringe nodes are sorted in increasing order of \( f(n) = g(n) + h(n) \)
  - \( g(n) \) = path cost from start node to \( n \)
  - \( h(n) \) = heuristic function value of node \( n \): estimated distance from \( n \) to goal state
- A* is complete
- A* is optimal if the heuristic function is consistent
- A* requires exponential time and space
  - But in practice, good heuristics provide much better time/space usage than other exponential search algos
More A* Examples

- Courtesy of Prof. Kathy McCoy:
  - 8-puzzle with \( h(n) = \) number of misplaced tiles
  - Robot navigation with \( h(n) = \) Manhattan distance from goal
Robot Navigation

\[ f(N) = h(N), \text{ with } h(N) = \text{Manhattan distance to the goal} \]

(not A*)

<table>
<thead>
<tr>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Iterative Deepening A*

- A* can still have very large memory requirements for high-depth problems
- Idea: use iterative deepening, except with increasing thresholds on \( f(n) \) instead of depth
- Initialize threshold \( c = f(\text{start state}) \)
- Loop until a solution is found:
  - Depth-first search limited to nodes with \( f(n) \leq c \)
  - \( c := \min f(n), n \text{ on fringe} \)

8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) = \text{number of misplaced tiles} \)

Cutoff=4
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) \) = number of misplaced tiles

Cutoff=4
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) \) = number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) \) = number of misplaced tiles

Cutoff=5

Simplified Memory-Bounded A*

- Perform A* as usual until memory is full
- Then drop the node \( n \) on the fringe with the largest \( f(n) \), and back up its value to its parent
  - After all children have been dropped, replace parent value with smallest child \( f(n) \)
  - The root of an erased subtree “remembers” the cost of the best path in that subtree
  - Subtree will be regenerated only if all other nodes in the fringe have greater \( f \) values
- Probably about the best general-purpose search algorithm available