Heuristic Search

CPSC 470/570 – Artificial Intelligence
Brian Scassellati
Goal Formulation

• Well-defined function that identifies both the goal states and the conditions under which to achieve the goal
  – Fly from Boston to San Francisco
  – Quality might depend on
    • Least amount of money
    • Fewest number of transfers
    • Shortest amount of time in the air
    • Shortest amount of time in airports
Problem Formulation

- Well-defined problems
  - Fully observable
  - Deterministic
  - Discrete set of possible actions (operations)

- State space: the set of all states that are reachable from an initial state by any sequence of actions

- Path: sequence of actions leading from one state to another
Problem Formulation

- Goal: spend less $
- State space: flights and their costs
- Path: sequence of flights
- Picking the right level of abstraction
  - Fly from Boston to Chicago
  - Directions to the airport
  - Move left leg 18 inches forward
How to Search:
Generating Sequences and Data Structures

Branching Factor $b=3$
Measuring Performance

• **Completeness**: is the strategy guaranteed to find a solution when one exists?
• **Time Complexity**: how long does it take to find a solution?
• **Space Complexity**: how much memory does it require to perform the search?
• **Optimality**: Does the strategy find the best-quality solution when more than one solution exists?
Types of Blind Search

- Breadth-First Search
- Depth-First Search
- Depth Limited Search
- Iterative Deepening Search
- Bi-directional Search
Improving Blind Search: Avoiding Repeated States

• Simple caching could be used to store the expected values of sub-trees.
  – Must maintain a table of all visited states and the result

• Change the rules for generating the tree
  – Do not generate repeated states
  – Do not generate paths with cycles
Heuristic Functions

- These techniques are all still brute-force
- Can we do anything more intelligent?
- If we could identify an *evaluation function*, which described how valuable each state was in obtaining the goal, then we could simply always choose to expand the leaf node with the best value.
- A *heuristic function* is an inexact estimate of the evaluation function.
Greedy Best-First Search

- Rely on a heuristic function to determine which node to expand
- Better name is “best-guess-first” search
- Airline example
  - Find the shortest path from Boston to Phoenix
Greedy Best-First-Search

- Minimize estimated cost to reach a goal (in this case, the distance to Phoenix)

<table>
<thead>
<tr>
<th></th>
<th>Straight Line Distance to Phoenix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boston</td>
<td>2299</td>
</tr>
<tr>
<td>Chicago</td>
<td>1447</td>
</tr>
<tr>
<td>Nashville</td>
<td>1444</td>
</tr>
<tr>
<td>Key West</td>
<td>1927</td>
</tr>
<tr>
<td>Austin</td>
<td>870</td>
</tr>
<tr>
<td>San Francisco</td>
<td>658</td>
</tr>
</tbody>
</table>

Total Distance Flown
Greedy Best-First-Search

• Optimal?
  – No, as the previous example demonstrated
• Complete?
  – No, just as depth first search
• Worst-case time complexity?
  – $O(b^m)$ where $b=\text{branch factor}$, $m=\text{max. depth}$
• Worst-case space complexity?
  – Same as time complexity… entire tree kept in memory
• Actual time/space complexity
  – Depends on the quality of the heuristic function
A* Search

- Combine Greedy search with Uniform Cost Search
- Minimize the total path cost \( f = g + h \)

---

<table>
<thead>
<tr>
<th>City</th>
<th>Distance to Phoenix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boston</td>
<td>2299</td>
</tr>
<tr>
<td>Chicago</td>
<td>1447</td>
</tr>
<tr>
<td>Nashville</td>
<td>1444</td>
</tr>
<tr>
<td>Key West</td>
<td>1927</td>
</tr>
<tr>
<td>Austin</td>
<td>870</td>
</tr>
<tr>
<td>San Francisco</td>
<td>658</td>
</tr>
</tbody>
</table>

Total Distance Flown

- Boston: \( f = 0 + 2299 = 2299 \)
- Chicago: \( f = 856 + 1447 = 2303 \)
- Nashville: \( f = 945 + 1444 = 2389 \)
- Key West: \( f = 1371 + 1927 = 3298 \)
- San Francisco: \( f = (856+1863) + 658 = 3377 \)
- Phoenix: \( f = (856+1447) + 0 = 2303 \)
- Austin: \( f = 2567 \)
- Phoenix: \( f = 3846 \)

---
How does A* Search Work?

- The heuristic function $h$ must be **admissible**
  - It must never over-estimate the cost to reach the goal
- Most obvious heuristics are **monotonic**
  - If the total path cost is non-monotonic as you move down the tree, you can substitute a monotonic function based on the parent
- Allows the above contour interpretation
Proving the Optimality of A*

- Assume that $G_2$ has been chosen for expansion over $n$
- Because $h$ is admissible
  \[ f^* \geq f(n) \]
- If $n$ is not chosen for expansion over $G_2$, we must have
  \[ f(n) \geq f(G_2) \]
- Combining these, we get
  \[ f^* \geq f(G_2) \]
- However, this violates our assertion that $G_2$ is sub-optimal
- Therefore, A* never selects a sub-optimal goal for expansion
Completeness of A*

• A* expands nodes in order of increasing $f$
• When would a solution not be found?
  – Node with an infinite branching factor
  – A path with a finite path cost but an infinite number of nodes
• A* is complete when
  – There is a finite branching factor
  – Every operator costs at least some positive $\varepsilon$
Complexity of A*

• Computation time is limited by the quality of the heuristic function (but is still exponential)
  – Issue #1: Choosing the right heuristic function can have a large impact

• More serious problem is that all generated nodes need to be kept in memory
  – Issue #2: Can we limit the memory requirements?
**Issue #1:**
Choosing a Heuristic Functions

- Must be admissible (never over-estimate)
- Heuristics for the 8-Puzzle
  - $h_1 =$ number of tiles in the wrong position
  - $h_2 =$ sum of the distances of the tiles from their goal positions (city block distance)
Effect of Heuristic Accuracy on Performance in the 8-puzzle

<table>
<thead>
<tr>
<th>d</th>
<th>IDS</th>
<th>A*(h₁)</th>
<th>A*(h₂)</th>
<th>IDS</th>
<th>A*(h₁)</th>
<th>A*(h₂)</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>6384</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>47127</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>364404</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>3473941</td>
<td>539</td>
<td>113</td>
<td>2.83</td>
<td>1.44</td>
<td>1.23</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>1301</td>
<td>211</td>
<td>–</td>
<td>1.45</td>
<td>1.25</td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>3056</td>
<td>363</td>
<td>–</td>
<td>1.46</td>
<td>1.26</td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>7276</td>
<td>676</td>
<td>–</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>22</td>
<td>–</td>
<td>18094</td>
<td>1219</td>
<td>–</td>
<td>1.48</td>
<td>1.28</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>39135</td>
<td>1641</td>
<td>–</td>
<td>1.48</td>
<td>1.26</td>
</tr>
</tbody>
</table>

- Compare iterative-deepening with A* using $h₁$ (number of misplaced tiles) and $h₂$ (city block distance)
- Effective branching factor $b^*$
  - Number of expanded nodes = $1 + b^* + (b^*)^2 + \ldots + (b^*)^{depth}$
  - $b^*$ remains relatively constant across many measurements
- Always better to use a heuristic with higher values, so long as it does not over-estimate
Issue #2
Limiting Memory Utilization

• If we can maintain a bound on the memory, we might be willing to wait for a solution

• Two techniques for Memory Bounded Search:
  – Iterative deepening A* (IDA*)
  – Recursive Best-First-Search (RBFS)
Iterative Deepening A* Search (IDA*)

• Each iteration is a depth-first search with a limit based on $f$ rather than on depth
• Complete and optimal (with same caveats as A*)
• Requires space proportional to the longest path that it explores
• Can have competitive time complexity, since the overhead of maintaining the nodes in memory is greatly reduced
Problems with IDA*

- In the TSP, different heuristic function value for each state
- Each contour contains only one additional node
- If A* expands N nodes, the IDA* will expand \[1+2+3+4+\ldots+N = O(N^2)\] nodes
- If N is too large for memory, \(N^2\) is too long to wait
- Runs into problems because it recalculates every node
Recursive Best-First Search (RBFS)

- total path cost \((f) = \text{actual path so far} \,(g) + \text{heuristic estimate of future path to goal} \,(h)\)
- Red values best \(f\)-value in an alternate branch
Recursive Best-First Search (RBFS)

• RBFS will
  – be **complete** given sufficient memory to store the shallowest solution path
  – be **optimal** if the heuristic function is admissible (and you have enough memory to store the solution)

• Both RBFS and IDA* use not enough memory.
  – Require at most linear space with the depth of the tree
tinyurl.com/yale-robot-study

Play video games with a robot!