Informed Search: Search Heuristics

CSE 40171: Artificial Intelligence
Homework #2 has been released
It is due at 11:59PM on 9/30
Where do heuristics come from?
A good heuristic can go a long way…
Creating Admissible Heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.

Often, admissible heuristics are solutions to relaxed problems, where new actions are available.

Inadmissible heuristics are often useful too.

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Case Study: the 8-puzzle
How many steps long is the solution?
8-puzzle Stats

Average Solution Cost: about 22 steps

Branching Factor: about 3 steps

Exhaustive Tree Search: $3^{22} \approx 3.1 \times 10^{10}$ states

Graph Search: $9!/2 = 181,440$ states

- But the corresponding number for the 15-puzzle is $10^{13}$
Large Numbers and Search Spaces

How large is a state space of $10$ trillion possibilities?

On a 3.1Ghz Core i7, it takes:
- 3m42.240s to enumerate $10^{10}$ states (exhaustive 8-puzzle)

Heuristics are needed to speed this up!
Heuristic Function #1

$h_1 = \text{the number of misplaced tiles}$

All eight tiles are out of position

Start state: $h_1 = 8$

Q: Why is this heuristic admissible?

A: Any tile that is out of place must be moved at least once
Heuristic Function #2

\[ h_2 = \text{the sum of the distances of the tiles from their goal positions} \]

Tiles can’t move along diagonals, so what do we do?
Manhattan Distance

The distance between two points in a grid based on a strictly horizontal and/or vertical path.

$$d_1(p, q) = \|p - q\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$$

where

$$p = (p_1, p_2, \ldots, p_n) \text{ and } q = (q_1, q_2, \ldots, q_n)$$
Heuristic Function #2

For this start state:

\[ h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 \]

Q: Why is this heuristic admissible?
A: Any tile that is out of place must be moved at least once
Effective Branching Factor

\[ N = \text{the total number of vertices generated by A*} \]
\[ d = \text{the solution depth} \]
\[ b^* = \text{the branching factor that a uniform tree of depth } d \]
\[ \text{would need to contain } N+1 \text{ vertices} \]

\[ N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d \]

Example: if A* finds a solution at depth 5 using 52 vertices, then the effective branching factor is 1.92
Effective Branching Factor

A well-designed heuristic has a value of $b^*$ close to 1

How do $h_1$ and $h_2$ stack up?

<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>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>1301</td>
<td>212</td>
<td></td>
<td>-</td>
<td>1.45</td>
<td>1.25</td>
</tr>
<tr>
<td>18</td>
<td>3056</td>
<td>363</td>
<td></td>
<td>-</td>
<td>1.46</td>
<td>1.26</td>
</tr>
<tr>
<td>20</td>
<td>7276</td>
<td>676</td>
<td></td>
<td>-</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>22</td>
<td>18094</td>
<td>1219</td>
<td></td>
<td>-</td>
<td>1.48</td>
<td>1.28</td>
</tr>
<tr>
<td>24</td>
<td>39135</td>
<td>1641</td>
<td></td>
<td>-</td>
<td>1.48</td>
<td>1.26</td>
</tr>
</tbody>
</table>
Is $h_2$ always better than $h_1$?

Yes: for any vertex $v$, $h_2(v) \geq h_1(v)$

Assume $C^*$ is the cost of the optimal solution path

Every vertex with $f(v) < C^*$ will be expanded

i.e., $h(v) < C^* - g(v)$ will be expanded

Every vertex expanded by $h_2$ will also be expanded by $h_1$, but $h_1$ may expand others as well
Relaxed Problems

What if a tile could move anywhere?

What if a tile could move one square in any direction — even onto an occupied square?

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Example: 8-puzzle

Original Problem:
A tile can move from square A to square B if
A is horizontally or vertically adjacent to B and B is blank

Three Relaxed Problems:
(a) A tile can move from square A to square B if A is adjacent to B
(b) A tile can move from square A to square B if B is blank
(c) A tile can move from square A to square B

Can derive manhattan distance from this one
Which heuristic do we want?

If a collection of admissible heuristics $h_1, \ldots, h_m$ is available, but none of them dominates any of the others, which should we choose?

Use a composite heuristic that is most accurate on the vertex in question:

$$h(v) = \max \{h_1(v), \ldots, h_m(v)\}$$
Subproblems

The task is to get tiles 1, 2, 3, 4 into their correct positions.

The cost of this subproblem is a lower bound on the cost of the complete problem.
Pattern Databases

• Store the exact solution costs for every possible subproblem instance

• Compute an admissible heuristic $h_{DB}$ for each complete state by looking up a corresponding subproblem configuration

• Don’t build all at once; add to DB for each new problem instance
Pattern Databases

Each database yields an admissible heuristic.

Heuristics can be combined by taking the maximum value.
Disjoint Pattern Databases

# of 1-2-3-4 moves + # of 5-6-7-8 moves

Sum is a lower bound on the cost of solving the entire problem
Speed-up achieved is several orders of magnitude
Pattern Databases

Rather shoddy strategy for modeling intelligence
Were the preceding strategies for coming up with heuristics good?
A better approach: learn from experience
Learning Heuristics From Experience

How can we do this with the 8-puzzle?

Solve a lot of 8-puzzles…

$h(v)$ can be learned from examples from optimal puzzle solutions

Each example consists of a state from the solution path and the cost of the solution from that point

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Applicable Learning Algorithms

Neural Networks

Decision Trees

Decision tree for detecting a 3-clique in a 4-vertex graph © BY-SA 3.0 Thore Husfeldt
Features

For search, learning works well when features are available that predict a state’s value, rather than just a raw state description.

Example for the 8-puzzle:

\[ x_1(v) = \text{number of misplaced tiles} \]

\[ x_2(v) = \text{number of pairs of adjacent tiles that are not adjacent in the goal state} \]
Combining Features

A common approach is to use a linear combination of features:

\[ h(v) = c_1 x_1(v) + c_2 x_2(v) \]

Constants \( c_1 \) and \( c_2 \) are adjusted to give the best fit to the underlying data on solution costs.
Lots of options for real problems
We will have a lot more to say about search and machine learning later...