## Compare two heuristics

<table>
<thead>
<tr>
<th>$d$</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
<th>$d$</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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3644035</td>
<td>227</td>
<td>73</td>
<td>2.78</td>
<td>1.42</td>
<td>1.24</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>–</td>
<td>539</td>
<td>113</td>
<td>–</td>
<td>1.44</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>1301</td>
<td>211</td>
<td>–</td>
<td>1.45</td>
<td>1.25</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>3056</td>
<td>363</td>
<td>–</td>
<td>1.46</td>
<td>1.26</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>7276</td>
<td>676</td>
<td>–</td>
<td>1.47</td>
<td>1.27</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>–</td>
<td>18094</td>
<td>1219</td>
<td>–</td>
<td>1.48</td>
<td>1.28</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>39135</td>
<td>1641</td>
<td>–</td>
<td>1.48</td>
<td>1.26</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.8** Comparison of the search costs and effective branching factors for the **ITERATIVE-DEEPENING-SEARCH** and $A^*$ algorithms with $h_1$, $h_2$. Data are averaged over 100 instances of the 8-puzzle, for various solution lengths.
Compare these two heuristics

$h_2$ is always better than $h_1$

- for any node, $n$, $h_2(n) \geq h_1(n)$
- $h_2$ dominates $h_1$
- Recall all nodes with $f(n) < C^*$ will be expanded?
  - This means all nodes, $h(n) < C^* - g(n)$, will be expanded
  - All nodes $h_2$ expands will also be expanded by $h_1$ and because $h_1$ is smaller, others will be expanded as well
Inventing admissible heuristic funcs

How can you create $h(n)$?

- Simplify problem by reducing restrictions on actions
  - Allow 8-puzzle pieces to sit atop on another
  - Call this a relaxed problem
  - The cost of optimal solution to relaxed problem is admissible heuristic for original problem
    - It is at least as expensive for the original problem
Examples of relaxed problems

A tile can move from square A to square B if

A is horizontally or vertically adjacent to B

and B is blank

- A tile can move from A to B if A is adjacent to B (overlap)
- A tile can move from A to B if B is blank (teleport)
- A tile can move from A to B (teleport and overlap)

Solutions to these relaxed problems can be computed without search and therefore heuristic is easy to compute
Multiple Heuristics

If multiple heuristics available:

- $h(n) = \max \{h_1(n), h_2(n), \ldots, h_m(n)\}$
Use solution to subproblem as heuristic

What is optimal cost of solving some portion of original problem?

- subproblem solution is heuristic of original problem
Pattern Databases

**Store optimal solutions to subproblems in database**

- We use an exhaustive search to solve every permutation of the 1,2,3,4 piece subproblem of the 8-puzzle.
- During solution of 8-puzzle, look up optimal cost to solve the 1,2,3,4 piece subproblem and use as heuristic.
Learning

*Could also build pattern database while solving cases of the 8-puzzle*

- Must keep track of intermediate states and true final cost of solution
- **Inductive learning** builds mapping of state -> cost
- Because too many permutations of actual states
  - Construct important **features** to reduce size of space
Local Search Algorithms and Optimization Problems
Characterize Techniques

**Uninformed Search**
- Looking for a solution where solution is a path from start to goal
- At each intermediate point along a path, we have no prediction of value of path

**Informed Search**
- Again, looking for a path from start to goal
- This time we have insight regarding the value of intermediate solutions
Now change things a bit

**What if the path isn’t important, just the goal?**

- So the goal is unknown
- The path to the goal need not be solved

**Examples**

- What quantities of quarters, nickels, and dimes add up to $17.45 and minimizes the total number of coins
- Is the price of Microsoft stock going up tomorrow?
Local Search

*Local search does not keep track of previous solutions*

- Instead it keeps track of current solution (current state)
- Uses a method of generating alternative solution candidates

**Advantages**

- Use a small amount of memory (usually constant amount)
- They can find reasonable (note we aren’t saying optimal) solutions in infinite search spaces
Optimization Problems

**Objective Function**

- A function with vector inputs and scalar output
  - goal is to search through candidate input vectors in order to minimize or maximize objective function

**Example**

- \( f(q, d, n) = 1,000,000 \) if \( q*0.25 + d*0.1 + n*0.05 \neq 17.45 \)
  - \( = q + n + d \) otherwise

- minimize \( f \)
Search Space

*The realm of feasible input vectors*

- Also called state space landscape
- Usually described by
  - number of dimensions (3 for our change example)
  - domain of dimensions (q is discrete from 0 to 69…)
  - nature of relationship between input vector and objective function output
    - no relationship
    - smoothly varying
    - discontinuities
Search Space

Looking for global maximum (or minimum)

Figure 4.10 A one-dimensional state space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.
Hill Climbing

Also called Greedy Search

• Select a starting point and set \textbf{current}
• evaluate (\textbf{current})
• loop do
  – neighbor = highest value successor of \textbf{current}
  – if evaluate (neighbor) \leq evaluate (\textbf{current})
    ▪ return \textbf{current}
  – else \textbf{current} = neighbor

Figure 4.10 – A one-dimensional state space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.
Hill climbing gets stuck

Hiking metaphor (you are wearing glasses that limit your vision to 10 feet)

- Local maxima
  - Ridges
- Plateau
  - why is this a problem?
Hill Climbing Gadgets

Variants on hill climbing play special roles

• stochastic hill climbing
  – don’t always choose the best successor

• first-choice hill climbing
  – pick the first good successor you find
    ▪ useful if number of successors is large

• random restart
  – follow steepest ascent from multiple starting states
  – probability of finding global max increases with number of starts
Hill Climbing Usefulness

*It Depends*

- Shape of state space greatly influences hill climbing
- Local maxima are the Achilles heel
- What is cost of evaluation?
- What is cost of finding a random starting location?