Admissibility and Monotonicity*

- Admissible heuristic \( h(n) = \) never overestimates the actual cost \( h^* \) to reach a goal & \( h(n) \geq 0 \) & \( h(\text{goal}) = 0 \)
- Monotone (Consistency) heuristic
  - For every node \( n \) and every successor \( n' \) reached from \( n \) by action \( a \)
  - \( h(n) \leq \) cost of \( (n, a, n') + h(a') + h(\text{goal}) \)
  - The deeper you go along a path the better (or as good) the estimate of the distance to the goal state
  - The value (which is \( g+h \)) never decreases along any path.
  - Implies \( \Rightarrow \) each state reached has the minimal \( g(n) \)
- When \( h \) is admissible, monotonicity can be maintained when combined with pathmax: \( f(n') = \max(f(n), g(n') + h(n')) \)
  - Create new \( h' \) from the above:
    - \( f(n') = \max(f(n), g(n') + h(n')) \)
  - Force \( f \) to never decrease along path since \( (n') \Rightarrow f(n) \)

Does monotonicity in \( f \) imply admissibility?

Consistent \( h \Rightarrow \) Monotone \( f \)

A heuristic is consistent if

\[
\begin{align*}
h(n) & \leq c(n, a, n') + h(n') \\
\text{If } h \text{ is consistent, we have} & \\
f(n') & = g(n') + h(n') \\
& = g(n) + c(n, a, n') + h(a') \\
& \geq g(n) + h(n) \quad \text{[or } f(n) \text{]} \\
\end{align*}
\]

I.e., \( f(n) \) is nondecreasing along any path.
Proof of Optimality of A*

Let \( O \) be an optimal solution with path cost \( f^* \).
Let \( SO \) be a suboptimal goal state, that is \( g(SO) > f^* \).
Suppose that A* terminates the search with \( SO \).
Let \( n \) be a leaf node on the optimal path to \( O \)
\[ f^* \geq f(n) \quad \text{admissibility of } h \]
\[ f(n) \geq f(SO) \quad n \text{ was not chosen for expansion} \]
\[ f^* \geq f(n) \geq f(SO) \]
\[ f(SO) = g(SO) \quad SO \text{ is a goal, } h(SO) = 0 \]
\[ f^* \geq g(SO) \quad \text{contradiction!} \]

A* is maximally efficient

- For a given heuristic function, no optimal algorithm is guaranteed to do less work in terms of nodes expanded.
- Aside from ties in \( f \), A* expands every node necessary for the proof that we’ve found the shortest path, and no other nodes.

Questions*

- What is the implications of local monotonicity
  - Amount of storage
- What happens if \( h_1 \leq h_2 \leq h \) for all states
  - \( h_2 \) dominates \( h_1 \)
- If \( h_1 \) and \( h_2 \) are admissible, is \( \max\{h_1, h_2\} \) admissible?
  - Is it better than \( h_1 \) and \( h_2 \)?
- What are the implications of overestimating \( h \)
  - Suppose you can bound overestimation
- What if you are doing a maximizing search

Completeness of A*

- A* is complete unless there are infinitely many nodes with \( f(n) < f^* \).
- A* is complete when:
  - there is a positive lower bound on the cost of operators.
  - the branching factor is finite.
Heuristic Function Performance

While informed search can produce dramatic real (average-case) improvements in complexity, it typically does not eliminate the potential for exponential (worst-case) performance.

The performance of heuristic functions can be compared using several metrics:

- Average number of nodes expanded ($N$)
- Penetrance ($P = d/N$)
- Effective branching factor ($b^*$)

- If solution depth is $d$ then $b^*$ is the branching factor that a uniform search tree would have to have to generate $N$ nodes.

Given $d$ and $N$ what is the $b^*$

EBF tends to be relatively independent of the solution depth.

Note that these definitions completely ignore the cost of applying the heuristic function.

Measuring the heuristic payoff

Iterative Deepening vs $A^*$

<table>
<thead>
<tr>
<th>$d$</th>
<th>IDS $h_0$</th>
<th>$A^* h_0$</th>
<th>IDS $h_2$</th>
<th>$A^* h_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>2.45</td>
<td>1.79</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>2.67</td>
<td>1.48</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>2.75</td>
<td>1.54</td>
</tr>
<tr>
<td>8</td>
<td>4034</td>
<td>39</td>
<td>2.86</td>
<td>1.53</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>2.70</td>
<td>1.36</td>
</tr>
<tr>
<td>12</td>
<td>26465</td>
<td>227</td>
<td>2.78</td>
<td>1.42</td>
</tr>
<tr>
<td>14</td>
<td>347041</td>
<td>539</td>
<td>2.83</td>
<td>1.44</td>
</tr>
<tr>
<td>16</td>
<td>1361</td>
<td>211</td>
<td>1.45</td>
<td>1.25</td>
</tr>
<tr>
<td>18</td>
<td>1095</td>
<td>383</td>
<td>1.46</td>
<td>1.26</td>
</tr>
<tr>
<td>20</td>
<td>1276</td>
<td>676</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>24</td>
<td>10504</td>
<td>1219</td>
<td>1.48</td>
<td>1.28</td>
</tr>
<tr>
<td>26</td>
<td>30135</td>
<td>1841</td>
<td></td>
<td>1.29</td>
</tr>
</tbody>
</table>

Meta-Level Reasoning

- Search cost involves both the cost to expand nodes and the cost to apply heuristic function.
- Typically, there is a trade-off between the cost and performance of a heuristic function.
- E.g., we can always get a “perfect” heuristic function by having the function do a search to find the solution and then use that solution to compute $h(node)$.

Meta-Level Reasoning (cont’d)

This trade-off is often referred to as the meta-level vs. base-level trade-off:

- Base-level refers to the operator level, at which the problem will actually be solved;
- Meta-level refers to the control level, at which we decide how to solve the problem.

We must evaluate the cost to execute the heuristic function relative to the cost of expanding nodes and the reduction in nodes expanded.
**IDA* - Iterative deepening A* (Space/time trade-off)**

- A* requires open (& close?) list for remembering nodes
  - Can lead to very large storage requirements
- Exploit the idea the use of monotone $f$:
  - $f = g + h \leq f^*$ (actual cost) and $f(n) \leq f(\text{next node after } n)$
  - create incremental subspaces searched depth-first
  - much less storage
- Key issue is how much extra computation
  - How bad an underestimate $f$, how many steps does it take to get $f = f^*$
  - Worse case N computation for A*, versus $N^2$ for IDA*

**Iterative-Deepening-A**

- Algorithm: Iterative-Deepening-A*
  
  1) Set THRESHOLD = the heuristic evaluation of the start state.
  2) Conduct a depth-first search based on minimal cost from current node, pruning any branch when its total cost function $(g + h)$ exceeds THRESHOLD. If a solution path is found during the search, return it as the optimal solution.
  3) Otherwise, increment THRESHOLD by the minimum amount it was exceeded during the previous step, and then go to Step 2.
  - Start state always on path, so initial estimate is always underestimate and never decreasing.

**f-Cost Contours**

- Monotonic heuristics allow us to view A* in terms of exploring
  - The more informed a heuristic, the more the contours will be “stretched” toward the goal (they will be more focused around the optimal path).???
Stages in an IDA* Search for Bucharest

Nodes are labeled with $f = g + h$. The $h$ values are the straight-line distances to Bucharest...

What is the next Contour??

Experimental Results on IDA*

- IDA* is asymptotically same time as $A^*$ but only $O(d)$ in space - versus $O(b^d)$ for $A^*$
- Also avoids overhead of sorted queue of nodes
- IDA* is simpler to implement - no closed lists (limited open list).
- In Korf's 15-puzzle experiments IDA*: solved all problems, ran faster even though it generated more nodes than $A^*$??.
- $A^*$: solved no problems due to insufficient space; ran slower than IDA*

RBFS - Recursive Best-First Search

- Mimics best-first search with linear space
- Similar to recursive depth-first
  - Limits recursion by keeping track of the $f$-value of the best alternative path from any ancestor node – one step look-ahead
  - If current node exceeds this value, recursion unwinds back to the alternative path – same idea as contour
  - As recursion unwinds, replaces $f$-value of node with best $f$-value of children
  - Allows to remember whether to re-expand path at later time
- Exploits information gathered from previous searches about minimum $f$ so as to focus further

RBFS - Recursive Best-First Search Algorithm

```
function RBFS(problem, node, f_limit) returns a solution, or failure
RBFS(MakeNode() problem, [initial node], f_limit)

function RBFS(node, f_limit) if problem has a solution, or failure and a new f-limit
if node is a failure then return failure
else if node is a solution then return node
else for each of node's children do
    $f = \text{heuristic value}$
    if $f < f_{\text{limit}}$ then
        $f_{\text{limit}} = f$
    end
end repeat
$\text{Define contour limit} ^{\text{Pathheuristic guarantee monotonic}}$

f-limit = $f_{\text{limit}}$
Repeat
lookup = the lowest $f$-value node in ancestors
if $f_{\text{limit}} < f$ then return failure, $f_{\text{limit}}$
else for each node in ancestors(f)
    if $f < f_{\text{limit}}$ then
        $f_{\text{limit}} = f$
    end
end
else for each of node's children do
    $f = \text{heuristic value}$
    if $f < f_{\text{limit}}$ then
        $f_{\text{limit}} = f$
    end
end repeat
$\text{Define next highest f-contour}$

Repeat
lookup = the lowest $f$-value node in ancestors
if $f_{\text{limit}} < f$ then return failure, $f_{\text{limit}}$
else for each node in ancestors(f)
    if $f < f_{\text{limit}}$ then
        $f_{\text{limit}} = f$
    end
end
end repeat
Repeat
lookup = the lowest $f$-value node in ancestors
if $f_{\text{limit}} < f$ then return failure, $f_{\text{limit}}$
 else for each node in ancestors(f)
    if $f < f_{\text{limit}}$ then
        $f_{\text{limit}} = f$
    end
end
end repeat
end repeat
```
Next lecture

- Other Time and Space Variations of A*
  - Finish off RBFS
  - SMA*
  - Anytime A*
  - RTA* (maybe if have time)