Informed Search

- How can we make search smarter?
- Use problem-specific knowledge beyond the definition of the problem itself
- Specifically, incorporate knowledge of how good a non-goal state is

Best-First Search

- Node selected for expansion based on an evaluation function \( f(n) \). i.e. expand the node that appears to be the best
- Node with lowest evaluation is selected for expansion
- Uses a priority queue
- We’ll talk about Greedy Best-First Search and A* Search

Heuristic Function

- \( h(n) = \) estimated cost of the cheapest path from node \( n \) to a goal node
- \( h(\text{goal node}) = 0 \)
- Contains additional knowledge of the problem

Greedy Best-First Search

- Expands the node that is closest to the goal
- \( f(n) = h(n) \)
**Greedy Best-First Search Example**

- **Corvallis**
- **Albany**: 49
- **Salem**: 28
- **Portland**: 17
- **McMinville**: 18

**Corvallis → McMinville → Wilsonville = 74 miles**

**Evaluating Greedy Best-First Search**

<table>
<thead>
<tr>
<th>Complete?</th>
<th>No (could start down an infinite path)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal?</td>
<td>No</td>
</tr>
<tr>
<td>Time Complexity</td>
<td></td>
</tr>
<tr>
<td>Space Complexity</td>
<td></td>
</tr>
</tbody>
</table>

**Greedy Best-First Search results in lots of dead ends which leads to unnecessary nodes being expanded**

**But the route below is much shorter than the route found by Greedy Best-First Search!**

Corvallis → Albany → Salem → Wilsonville = 67 miles
Evaluating Greedy Best-First Search

<table>
<thead>
<tr>
<th>Complete?</th>
<th>No (could start down an infinite path)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal?</td>
<td>No</td>
</tr>
<tr>
<td>Time Complexity</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>Space Complexity</td>
<td></td>
</tr>
</tbody>
</table>

Greedy Best-First search results in lots of dead ends which leads to unnecessary nodes being expanded

A* Search

- A much better alternative to greedy best-first search
- Evaluation function for A* is: $f(n) = g(n) + h(n)$
  where $g(n)$ = path cost from the start node to $n$
- If $h(n)$ satisfies certain conditions, A* search is optimal and complete!

A* Search Example

<table>
<thead>
<tr>
<th>Corvallis</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albany</td>
<td>49</td>
</tr>
<tr>
<td>Salem</td>
<td>28</td>
</tr>
<tr>
<td>Portland</td>
<td>17</td>
</tr>
<tr>
<td>McMinnville</td>
<td>18</td>
</tr>
</tbody>
</table>

Admissible Heuristics

- A* is optimal if $h(n)$ is an admissible heuristic
- An admissible heuristic is one that never overestimates the cost to reach the goal
- Admissible heuristic = optimistic
- Straight line distance was an admissible heuristic

A* Search Example

<table>
<thead>
<tr>
<th>Corvallis</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albany</td>
<td>49</td>
</tr>
<tr>
<td>Salem</td>
<td>28</td>
</tr>
<tr>
<td>Portland</td>
<td>17</td>
</tr>
<tr>
<td>McMinnville</td>
<td>18</td>
</tr>
</tbody>
</table>
**A* Search Example**

Straight line distance (as the crow flies) to Wilsonville in miles

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corvallis</td>
<td>56</td>
</tr>
<tr>
<td>Albany</td>
<td>49</td>
</tr>
<tr>
<td>Salem</td>
<td>28</td>
</tr>
<tr>
<td>Portland</td>
<td>17</td>
</tr>
<tr>
<td>McMinnville</td>
<td>18</td>
</tr>
</tbody>
</table>

Note: Don’t stop when you put a goal state on the priority queue (otherwise you get a suboptimal solution)

**Proof that A* using TREE-SEARCH is optimal if h(n) is admissible**

- Suppose A* returns a suboptimal goal node $G_2$.
- $G_2$ must be the least cost node in the fringe. Let the cost of optimal solution be $C^*$.
- Because $G_2$ is suboptimal: $f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$.
- Now consider a fringe node $n$ on an optimal solution path to the goal $G$.
- If $h(n)$ is admissible then: $f(n) = g(n) + h(n) \leq C^*$
- We have shown that $f(n) \leq C^* < f(G_2)$, so $G_2$ will not get expanded before $n$. Hence A* must return an optimal solution.

**What about search graphs (more than one path to a node)?**

- What if we expand a state we’ve already seen?
- Suppose we use the GRAPH-SEARCH solution and not expand repeated nodes.
- Could discard the optimal path if it’s not the first one generated.
- One simple solution: ensure optimal path to any repeated state is always the first one followed (like in Uniform-cost search).
- Requires an extra requirement on $h(n)$ called consistency (or monotonicity).
Consistency

- A heuristic is consistent if, for every node \( n \) and every successor \( n' \) of \( n \) generated by any action \( a \):
  \[
  h(n) \leq c(n,a,n') + h(n')
  \]
  - A form of the triangle inequality – each side of the triangle cannot be longer than the sum of the two sides

\[
\begin{array}{c}
  n \quad \text{G} \quad n' \\
  c(n,a,n') \quad h(n') = 1 \quad h(n) = 2 \\
  h(n) \quad c(n,a,n') = 2 \quad h(n') = 1 \\
  h(n) = 2 \quad c(n,a,n') = 2 \\
  \end{array}
\]

CONSISTENT  INCONSISTENT

A* is Optimally Efficient

- Among optimal algorithms that expand search paths from the root, A* is optimally efficient for any given heuristic function.
  - Optimally efficient: no other optimal algorithm is guaranteed to expand fewer nodes than A*
    - Fine print: except A* might possibly expand more nodes with \( f(n) = C^* \) where \( C^* \) is the cost of the optimal path – tie-breaking issues
  - Any algorithm that does not expand all nodes with \( f(n) < C^* \) runs the risk of missing the optimal solution

Evaluating A* Search

With a consistent heuristic, A* is complete, optimal and optimally efficient. Could this be the answer to our searching problems?

The Dark Side of A*…

Time complexity is exponential (although it can be reduced significantly with a good heuristic).
The really bad news: space complexity is exponential (usually need to store all generated states). Typically runs out of space on large-scale problems.
Summary of A* Search

Complete? Yes if \( h(n) \) is consistent, \( b \) is finite, and all step costs exceed some finite \( \varepsilon \).

Optimal? Yes if \( h(n) \) is consistent and admissible.

Time Complexity \( O(b^d) \) (In the worst case but a good heuristic can reduce this significantly).

Space Complexity \( O(b^d) \) – Needs \( O \) (number of states), will run out of memory for large search spaces.

1 Since \( f(n) \) is nondecreasing, we must eventually hit an \( f(n) = \text{cost of the path to a goal state} \).

Iterative Deepening A*

- Use iterative deepening trick to reduce memory requirements for A*.
- In each iteration do a “cost-limited” depth first search.
  - Cutoff is based on the \( f \)-cost (\( g+h \)) rather than the depth
- After each iteration, the new cutoff is the smallest \( f \)-cost that exceeded the cutoff in the previous iteration

Complete, Optimal but more costly than A* and can take a while to run with real-valued costs.

Examples of heuristic functions

The 8-puzzle

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

Start State

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

End State

Heuristic #1: \( h_1 = \text{number of misplaced tiles eg. start state has 8 misplaced tiles}. \)
This is an admissible heuristic.
Examples of heuristic functions

The 8-puzzle

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

Heuristic #2: total Manhattan distance (sum of horizontal and vertical moves, no diagonal moves). Start state is 3+1+2+3+2+2+3+18 moves away from the end state. This is also an admissible heuristic.

Which heuristic is better?

- $h_2$ dominates $h_1$. That is, for any node $n$, $h_2(n) \geq h_1(n)$.
- $h_2$ never expands more nodes than $A^*$ using $h_1$ (except possibly for some nodes with $f(n) = C^*$).

Proof:

Let $h_1$ and $h_2$ be admissible heuristics.

Every node with $f(n) < C^*$ will surely be expanded, since $A^*$ is optimal with an admissible heuristic. Since $f(n) = g(n) + h(n)$, every node with $h(n) < C^* - g(n)$ will surely be expanded for either heuristic.

Since $h_2$ is at least as big as $h_1$ for all nodes, every node expanded with $A^*$ using $h_2$ will also be expanded with $A^*$ using $h_1$. But $h_2$ might expand other nodes as well. In other words, we have $h_1(n) \leq h_2(n) < C^* - g(n)$.

- Better to use $h_2$ provided it doesn’t overestimate (i.e., it is also admissible) and its computation time isn’t too expensive.

Which heuristic is better?

<table>
<thead>
<tr>
<th>Depth</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>
</tr>
<tr>
<td>3</td>
<td>112</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>880</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>6384</td>
<td>39</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>47127</td>
<td>93</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>3646035</td>
<td>227</td>
<td>73</td>
</tr>
<tr>
<td>8</td>
<td>1449</td>
<td>113</td>
<td>113</td>
</tr>
<tr>
<td>9</td>
<td>1551</td>
<td>211</td>
<td>211</td>
</tr>
<tr>
<td>10</td>
<td>3556</td>
<td>363</td>
<td>363</td>
</tr>
<tr>
<td>11</td>
<td>7276</td>
<td>676</td>
<td>676</td>
</tr>
<tr>
<td>12</td>
<td>18094</td>
<td>1219</td>
<td>1219</td>
</tr>
<tr>
<td>13</td>
<td>39135</td>
<td>1641</td>
<td>1641</td>
</tr>
</tbody>
</table>

From Russell and Norvig Figure 4.8 (Results averaged over 100 instances of the 8-puzzle for depths 2-24).

Inventing Admissible Heuristics

- Relaxed problem: a problem with fewer restrictions on the actions.
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.
- If we relax the rules so that a square can move anywhere, we get heuristic $h_1$.
- If we relax the rules to allow a square to move to any adjacent square, we get heuristic $h_2$.

What you should know

- Be able to run $A^*$ by hand on a simple example.
- Why it is important for a heuristic to be admissible and consistent.
- Pros and cons of $A^*$.
- How do you come up with heuristics.
- What it means for a heuristic function to dominate another heuristic function.