Informed Search

Ron Parr
Department of Computer Science
Duke University

Example

For a uninformed strategy, $N_1$ and $N_2$ are just two nodes (at some position in the search tree)

Goal state
Example

For a heuristic strategy counting the number of misplaced tiles, $N_2$ is more promising than $N_1$.

Heuristic Function

• The heuristic function $h(N) \geq 0$ estimates the cost to go from $\text{STATE}(N)$ to a goal state.

  Value is independent of the current search tree; it depends only on $\text{STATE}(N)$ and the goal test GOAL?

• Example:

  \[
  \begin{array}{ccc}
  5 & 8 & \text{STATE}(N) \\
  4 & 2 & 1 \\
  7 & 3 & 6 \\
  \end{array} \quad \begin{array}{ccc}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
  7 & 8 & \text{Goal state} \\
  \end{array}
  \]

  $h(N) = \text{number of misplaced numbered tiles} = 6$

  [Why is it an estimate of the distance to the goal?]
Informed/Heuristic Search

- Idea: Give the search algorithm hints
- Heuristic function: \( h(x) \)
- \( h(x) = \) estimate of cost to goal from \( x \)
- If \( h(x) \) is 100% accurate, then we can find the goal in \( O(bd) \) time

\[
\begin{align*}
    h_1(N) &= \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2} \\
    h_2(N) &= |x_N - x_g| + |y_N - y_g| \quad \text{(L}_1\text{ or Manhattan distance)}
\end{align*}
\]
Greedy Best First Search

- Expand node with lowest $h(x)$
- (Implement priority queue on $h$)
- Optimal if $h(x)$ is 100% correct
- How can we get into trouble with this?

What Price Greed?

What’s broken with greedy search?
Best-First ≠ Efficiency

Local-minimum problem

\[ f(N) = h(N) = \text{straight distance to the goal} \]

A*

- Path cost so far: \( g(x) \)
- Total cost estimate: \( f(x) = g(x) + h(x) \)
- Maintain frontier as a \textit{priority queue} (on \( f \))
- \( O(bd) \) time if \( h \) is 100% accurate
- We want \( h \) to be an \textit{admissible} heuristic
- Admissible: never overestimates cost
- Why admissible?
  
  (guarantees optimality, completeness of A*)
8-Puzzle Heuristics

- $h_1(N)$ = number of misplaced tiles = 6 is admissible

- $h_2(N)$ = sum of the (Manhattan) distances of every tile to its goal position
  
  $= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13$

  is ???
8-Puzzle Heuristics

- $h_1(N)$ = number of misplaced tiles = 6 is admissible
- $h_2(N)$ = sum of the (Manhattan) distances of every tile to its goal position
  
  \[
  = 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13
  \]
  
  is admissible

Robot Navigation Heuristics

Cost of one horizontal/vertical step = 1
Cost of one diagonal step = $\sqrt{2}$

\[
h_1(N) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}
\]

is admissible
Robot Navigation Heuristics

Cost of one horizontal/vertical step = 1
Cost of one diagonal step = $\sqrt{2}$

$h_2(N) = |x_N - x_g| + |y_N - y_g|$ is admissible if moving along diagonals is not allowed, and not admissible otherwise.
Robot Navigation

\[ f(N) = h(N), \text{ with } h(N) = \text{Manhattan distance to the goal} \]

(not A*)
**Robot Navigation**

\[ f(N) = h(N), \text{ with } h(N) = \text{Manhattan distance to the goal (not A*)} \]

<table>
<thead>
<tr>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

**Robot Navigation**

\[ f(N) = g(N)+h(N), \text{ with } h(N) = \text{Manhattan distance to goal (A*)} \]

<table>
<thead>
<tr>
<th>8+3</th>
<th>7+4</th>
<th>6+3</th>
<th>5+6</th>
<th>4+7</th>
<th>3+8</th>
<th>2+9</th>
<th>3+10</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7+2</td>
<td>5+6</td>
<td>4+7</td>
<td>3+8</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6+1</td>
<td>3</td>
<td>2+9</td>
<td>1+100+11</td>
<td>1</td>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7+0</td>
<td>6+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8+1</td>
<td>7+2</td>
<td>6+3</td>
<td>5+4</td>
<td>4+5</td>
<td>3+6</td>
<td>2+7</td>
<td>3+8</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Some A* Properties

- Admissibility implies \( h(x) = 0 \) if \( x \) is a goal state
- Above implies \( f(x) = \text{cost to goal} \) if \( x \) is a goal state and \( x \) is popped off the queue

- What if \( h(x) = 0 \) for all \( x \)?
  - Is this admissible?
  - What does the algorithm do?

Result #1

A* is **complete and optimal**

[This result holds if nodes revisiting states are not discarded]
Proof (1/2)

If a solution exists, A* terminates and returns a solution

- For each node \( N \) on the frontier,
  \[ f(N) = g(N) + h(N) \geq g(N) \geq d(N) \times \epsilon, \]
  where \( d(N) \) is the depth of \( N \) in the tree.

- As long as A* hasn’t terminated, a node \( K \) on the frontier lies on a solution path.
Proof (1/2)

• If a solution exists, A* terminates and returns a solution
  - For each node N on the frontier, 
    \[ f(N) = g(N) + h(N) \geq g(N) \geq d(N) \times \epsilon, \]
    where \( d(N) \) is the depth of N in the tree
  - As long as A* hasn’t terminated, a node K on the frontier lies on a solution path
  - Since each node expansion increases the length of one path, K will eventually be selected for expansion, unless a solution is found along another path

Proof (2/2)

• Whenever A* chooses to expand a goal node, the path to this node is optimal
  - \( C^* \): cost of the optimal solution path
  - \( G' \): non-optimal goal node in the frontier
    \[ f(G') = g(G') + h(G') = g(G') > C^* \]
  - A node K in the frontier lies on an optimal path:
    \[ f(K) = g(K) + h(K) \leq C^* \]
  - So, \( G' \) will not be selected for expansion
What to do with revisited states?

The heuristic $h$ is clearly admissible.

If we discard this new node, then the search algorithm expands the goal node next and returns a non-optimal solution.
- Not harmful to discard a node revisiting a state if cost of the new path state is ≥ cost of previous path [so, in particular, one can discard a node if it re-visits a state already visited by one of its ancestors]

- A* remains optimal, but states may be re-visited multiple times [the size of the search tree can still be exponential in the number of visited states]

- Fortunately, for a large family of admissible heuristics – consistent heuristics – there is a much more efficient way to handle revisited states

---

**Consistent Heuristic**

- An admissible heuristic h is consistent (or monotone) if for each node N and each child N’ of N: $h(N) \leq c(N,N') + h(N')$

\[ \text{(triangle inequality)} \]

- Intuition: a consistent heuristics becomes more precise as we get deeper in the search tree
Consistency Violation

If \( h \) tells that \( N \) is 100 units from the goal, then moving from \( N \) along an arc costing 10 units should not lead to a node \( N' \) that \( h \) estimates to be 10 units away from the goal.

**Consistent Heuristic**

(alternative definition)

- A heuristic \( h \) is consistent (or monotone) if
  1. for each node \( N \) and each child \( N' \) of \( N \):
     \[
     h(N) \leq c(N,N') + h(N')
     \]
  2. for each goal node \( G \):
     \[
     h(G) = 0
     \]

A consistent heuristic is also admissible.
Admissibility and Consistency

• A consistent heuristic is also admissible

• An admissible heuristic may not be consistent, but many admissible heuristics are consistent

8-Puzzle

- $h_1(N) = \text{number of misplaced tiles}$
- $h_2(N) = \text{sum of the (Manhattan) distances of every tile to its goal position}$

are both consistent (why?)
Robot Navigation

Cost of one horizontal/vertical step = 1
Cost of one diagonal step = \( \sqrt{2} \)

\[ h(N) \leq c(N,N') + h(N') \]

Result #2

- If \( h \) is consistent, then whenever A* expands a node, it has already found an optimal path to this node’s state
Proof (1/2)

1. Consider a node $N$ and its child $N'$
   Since $h$ is consistent: $h(N) \leq c(N',N') + h(N')$

   $$f(N) = g(N) + h(N) \leq g(N) + c(N',N') + h(N') = f(N')$$
   So, $f$ is non-decreasing along any path

Proof (2/2)

2. If a node $K$ is selected for expansion, then any other node $N$ in the frontier verifies $f(N) \geq f(K)$

   - If one node $N$ lies on another path to the state of $K$, the cost of this other path is no smaller than that of the path to $K$:
     $$f(N') \geq f(N) \geq f(K) \quad \text{and} \quad h(N') = h(K)$$
     So, $g(N') \geq g(K)$
If a node $K$ is selected for expansion, then any other node $N$ in the fringe verifies $f(N) \geq f(K)$.

If one node $N$ lies on another path to the state of $K$, the cost of this other path is no smaller than that of the path to $K$:

- $f(N') \geq f(N) \geq f(K)$
- $h(N') = h(K)$
- $g(N') \geq g(K)$

If $h$ is consistent, then whenever $A^*$ expands a node, it has already found an optimal path to this node's state.

**Result #2**

**Implication of Result #2**

The path to $N$ is the optimal path to $S$.
Revisited States with Consistent Heuristic (Search#3)

- When a node is expanded, store its state into VISITED
- When a new node N is generated:
  - If STATE(N) is in VISITED, discard N
  - If there exists a node N’ in the frontier such that STATE(N’) = STATE(N), discard the node – N or N’ – with the largest f (or, equivalently, g)

Heuristic Accuracy

- Let $h_1$ and $h_2$ be two consistent heuristics such that for all nodes N:
  $$h_1(N) \leq h_2(N)$$
- $h_2$ is said to be more accurate (or more informed) than $h_1$
  - $h_1(N)$ = number of misplaced tiles
  - $h_2(N)$ = sum of distances of every tile to its goal position
  - $h_2$ is more accurate than $h_1$
Result #3

• Let $h_2$ be more accurate than $h_1$
• Let $A_1^*$ be $A^*$ using $h_1$
  and $A_2^*$ be $A^*$ using $h_2$
• Whenever a solution exists, all the nodes
  expanded by $A_2^*$, except possibly for some
  nodes such that
  
  $f_1(N) = f_2(N) = C^*$ (cost of optimal solution)

  are also expanded by $A_1^*$

Proof

• $C^*$ = cost of optimal solution

• Every node $N$ such that $f(N) < C^*$ is eventually expanded. No node $N$ such that
  $f(N) > C^*$ is ever expanded

• Every node $N$ such that $h(N) < C^* - g(N)$ is eventually expanded. So, every
  node $N$ such that $h_2(N) < C^* - g(N)$ is expanded by $A_2^*$. Since $h_1(N) \leq h_2(N)$, $N$ is
  also expanded by $A_1^*$

• If there are several nodes $N$ such that $f_1(N) = f_2(N) = C^*$ (such nodes include
  the optimal goal nodes, if there exists a solution), $A_1^*$ and $A_2^*$ may or may
  not expand them in the same order (until one goal node is expanded)
How to create good heuristics?

- By solving relaxed problems at each node
- In the 8-puzzle, the sum of the distances of each tile to its goal position ($h_2$) corresponds to solving 8 simple problems:
  
  \[
  d_i \text{ is the length of the shortest path to move tile } i \text{ to its goal position, ignoring the other tiles, e.g., } d_5 = 2 \quad h_2(N) = \sum_{i=1}^{9} d_i(N)
  \]

- It ignores negative interactions among tiles

Can we do better?

- For example, we could consider two more complex relaxed problems:

  \[
  d_{1234} = \text{length of the shortest path to move tiles } 1, 2, 3, \text{ and } 4 \text{ to their goal positions, ignoring the other tiles}
  \]

- \( h = d_{1234} + d_{5678} \) [disjoint pattern heuristic]
- How to compute $d_{1234}$ and $d_{5678}$?
Can we do better?

• For example, we could consider two more complex relaxed problems:

  \[ h = d_{1234} + d_{5678} \]  
  \[ \text{[disjoint pattern heuristic]} \]

  \[ \rightarrow \text{Several order-of-magnitude speedups for the 15- and 24-puzzle (see R\&N)} \]

  \[ \begin{array}{c|c|c|c} 
    4 & 2 & 1 \\
    \hline 
    1 & 2 & 3 \\
  \end{array} \rightarrow \begin{array}{c|c|c|c} 
    5 & 8 & 1 \\
    \hline 
    4 & 2 & 1 \\
  \end{array} \quad \begin{array}{c|c|c|c} 
    5 & 8 & 3 \\
    \hline 
    1 & 2 & 3 \\
  \end{array} \rightarrow \begin{array}{c|c|c|c} 
    5 & 8 & 3 \\
    \hline 
    4 & 2 & 1 \\
  \end{array} \]

  \[ \rightarrow \text{These distances are pre-computed and stored} \]

  \[ \text{[Each requires generating a tree of 3,024 nodes/states (breadth-first search)]} \]

Effective Branching Factor

• Used as measure the effectiveness of \( h \)

• Let \( n \) be the total number of nodes expanded by A* for a particular problem and \( d \) the depth of the solution

• The effective branching factor \( b^* \) is defined by fitting: \( n = 1 + b^* + (b^*)^2 + ... + (b^*)^d \)
Experimental Results
(see R&N for details)

- 8-puzzle with:
  - $h_1 =$ number of misplaced tiles
  - $h_2 =$ sum of distances of tiles to their goal positions
- Random generation of many problem instances
- Average effective branching factors (number of expanded nodes):

<table>
<thead>
<tr>
<th>$d$</th>
<th>$d$IDS</th>
<th>$A_1^*$</th>
<th>$A_2^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.45</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>6</td>
<td>2.73</td>
<td>1.34</td>
<td>1.30</td>
</tr>
<tr>
<td>12</td>
<td>2.78 (3,644,035)</td>
<td>1.42 (227)</td>
<td>1.24 (73)</td>
</tr>
<tr>
<td>16</td>
<td>--</td>
<td>1.45</td>
<td>1.25</td>
</tr>
<tr>
<td>20</td>
<td>--</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>24</td>
<td>--</td>
<td>1.48 (39,135)</td>
<td>1.26 (1,641)</td>
</tr>
</tbody>
</table>

Memory-bounded Search: Why?

- We run out of memory before we run out of time
- Problem: Need to remember entire search horizon
- Solution: Remember only a partial search horizon

- Issue: Maintaining optimality, completeness
- Issue: How to minimize time penalty
- Details: Not emphasized in class, but worth a skim so that you are aware of the issues
Iterative Deepening A* (IDA*)

- Idea: Reduce memory requirement of A* by applying cutoff on values of f
- Consistent heuristic function h
- Algorithm IDA*:
  - Initialize cutoff to $f$ (initial node)
  - Repeat:
    - Perform depth-first search by expanding all nodes $N$ such that $f(N) \leq$ cutoff
    - Reset cutoff to smallest value $f$ of non-expanded (leaf) nodes

Advantages/Drawbacks of IDA*

- Advantages:
  - Still complete and optimal
  - Requires less memory than A*
  - Avoid the overhead to sort the frontier
- Drawbacks:
  - Can’t avoid revisiting states not on the current path
  - Available memory is poorly used
  - Non-unit costs?

Cutoff = 3
h=1
h=1
h=1
h=2

RBFS

• Recursive best first search
• Objective: Linear space

• Idea: Remember best alternative
• Rewind, try alternatives if “best first” path gets too expensive
• Remember costs on the way back up

Assume $h=1$, initially along this path.

Replace with $f = 11$

Return to best alternative

Problem: Thrashing!
**SMA***

- Idea: Use all of available memory
- Discard the *worst* leaf when memory starts to run out, to make room for new leaves
- Values get backed up to parents
- Optimal if solution fits in memory
- Complete
- Thrashing still possible

**Recap**

- Heuristics change how we think about search
- A* is optimal, complete
- Dramatic improvements in efficiency possible with good heuristics
- Many extensions possible, e.g., dealing with limited memory