Heuristic Functions

Outline

I. Properties of Heuristics

II. Variations of A* search

III. Generating heuristics

* Figures are from the textbook site (or drawn by the instructor) unless the source is specifically cited.
I. Admissible Heuristic

- A* search is *complete* (when the state space either has a solution or is finite).

- Whether it is optimal depends on the heuristic.
I. Admissible Heuristic

♦ A* search is \textit{complete} (when the state space either has a solution or is finite).

♦ Whether it is optimal depends on the heuristic.

A heuristic is \textit{admissible} if it \textit{never overestimates} the cost to reach a goal.
I. Admissible Heuristic

- A* search is *complete* (when the state space either has a solution or is finite).

- Whether it is optimal depends on the heuristic.

A heuristic is *admissible* if it *never overestimates* the cost to reach a goal.

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Drobeta</td>
<td>242</td>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
I. Admissible Heuristic

♦ A* search is **complete** (when the state space either has a solution or is finite).

♦ Whether it is optimal depends on the heuristic.

A heuristic is **admissible** if it *never overestimates* the cost to reach a goal.

$h_{SLD}$: straight-line distance

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Drobeta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehedia</td>
<td></td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urzicien</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

$h_{SLD}$ is admissible because the actual distance to Bucharest cannot be less than the straight-line distance.
Optimality of A*

**Theorem**  \( A^* \) is cost-optimal with an admissible heuristic.
Optimality of A*

**Theorem** A* is cost-optimal with an admissible heuristic.

**Proof** By contradiction. Suppose the algorithm returns a path with cost \( C \) greater than the optimal cost \( C^* \).
Theorem  A* is cost-optimal with an admissible heuristic.

Proof  By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$.

Let $n$ be the first node on the optimal path that is unexpanded.
Optimality of A*

**Theorem**  A* is cost-optimal with an admissible heuristic.

**Proof**  By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$. Let $n$ be the first node on the optimal path that is unexpanded. $g^*(n)$: optimal cost from start to $n$. 

![Diagram](PathDiagram.png)
Optimality of A*

**Theorem**  A* is cost-optimal with an admissible heuristic.

**Proof**  By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$.

Let $n$ be the first node on the optimal path that is unexpanded.

- $g^*(n)$: optimal cost from start to $n$
- $h^*(n)$: optimal cost from $n$ to a goal
Theorem A* is cost-optimal with an admissible heuristic.

Proof By contradiction. Suppose the algorithm returns a path with cost \( C \) greater than the optimal cost \( C^* \).

Let \( n \) be the first node on the optimal path that is unexpanded.

\[ g^*(n) : \text{optimal cost from start to } n \]
\[ h^*(n) : \text{optimal cost from } n \text{ to a goal} \]

\[ f(n) > C^* \] (otherwise \( f(n) \leq C^* < C \) so \( n \) would have been expanded)
Optimality of A*

Theorem  A* is cost-optimal with an admissible heuristic.

Proof  By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$.

Let $n$ be the first node on the optimal path that is unexpanded.

- $g^*(n)$: optimal cost from start to $n$
- $h^*(n)$: optimal cost from $n$ to a goal

But $f(n) > C^*$ (otherwise $f(n) \leq C^* < C$ so $n$ would have been expanded)

But $f(n) = g(n) + h(n)$
Optimality of A*

**Theorem** A* is cost-optimal with an admissible heuristic.

**Proof** By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$. Let $n$ be the first node on the optimal path that is *unexpanded.*

Let $n$ be the first node on the optimal path that is *unexpanded.*

- $g^*(n)$: optimal cost from start to $n$
- $h^*(n)$: optimal cost from $n$ to a goal

But $f(n) > C^*$ (otherwise $f(n) \leq C^* < C$ so $n$ would have been expanded)

But $f(n) = g(n) + h(n)$

$$= g^*(n) + h(n)$$ (n on the optimal path)
Optimality of A*}

**Theorem** A* is cost-optimal with an admissible heuristic.

**Proof** By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$. 

Let $n$ be the first node on the optimal path that is *unexpanded*.

- $g^*(n)$: optimal cost from start to $n$
- $h^*(n)$: optimal cost from $n$ to a goal

But $f(n) = g(n) + h(n) = g^*(n) + h(n)$ (on the optimal path)

$$f(n) > C^* \text{ (otherwise } f(n) \leq C^* < C \text{ so } n \text{ would have been expanded})$$

$$f(n) \leq g^*(n) + h^*(n) \quad (h(n) \leq h^*(n) \text{ due to admissibility})$$
Optimality of A*

**Theorem** A* is cost-optimal with an admissible heuristic.

**Proof** By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$.

Let $n$ be the first node on the optimal path that is *unexpanded*.

Let $g^*(n)$: optimal cost from start to $n$.

Let $h^*(n)$: optimal cost from $n$ to a goal.

$f(n) > C^*$ (otherwise $f(n) \leq C^* < C$ so $n$ would have been expanded)

But $f(n) = g(n) + h(n)$

$= g^*(n) + h(n)$

$\leq g^*(n) + h^*(n)$

$= C^*$

$h(n) \leq h^*(n)$ due to admissibility.
**Optimality of A***

**Theorem**  A* is cost-optimal with an admissible heuristic.

**Proof**  By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$. Let $n$ be the first node on the optimal path that is unexpanded.

- $g^*(n)$: optimal cost from start to $n$
- $h^*(n)$: optimal cost from $n$ to a goal

But $f(n) > C^*$ (otherwise $f(n) \leq C^* < C$ so $n$ would have been expanded)

$$
\begin{align*}
    f(n) &= g(n) + h(n) \\
    &= g^*(n) + h(n) \quad (n \text{ on the optimal path}) \\
    &\leq g^*(n) + h^*(n) \quad (h(n) \leq h^*(n) \text{ due to admissibility}) \\
    &= C^*
\end{align*}
$$

That is, $f(n) \leq C^*$, contradicting with $f(n) > C^*$. 

**Optimality of A***

**Theorem** A* is cost-optimal with an admissible heuristic.

**Proof** By contradiction. Suppose the algorithm returns a path with cost $C$ greater than the optimal cost $C^*$.

Let $n$ be the first node on the optimal path that is unexpanded.

- $g^*(n)$: optimal cost from start to $n$
- $h^*(n)$: optimal cost from $n$ to a goal

$f(n) > C^*$ (otherwise $f(n) \leq C^* < C$ so $n$ would have been expanded)

But $f(n) = g(n) + h(n)$

$$= g^*(n) + h(n) \quad (n \text{ on the optimal path})$$

$$\leq g^*(n) + h^*(n) \quad (h(n) \leq h^*(n) \text{ due to admissibility})$$

$$= C^*$$

That is, $f(n) \leq C^*$, contradicting with $f(n) > C^*$. 
A heuristic $h$ is consistent if for every two nodes $n$ and $n'$ such that $n'$ is generated from $n$ by some action $a$, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

*(triangle inequality)*
A heuristic $h$ is **consistent** if for every two nodes $n$ and $n'$ such that $n'$ is generated from $n$ by some action $a$, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

(*triangle inequality*)

- Every consistent heuristic is admissible.
A heuristic $h$ is *consistent* if for every two nodes $n$ and $n'$ such that $n'$ is generated from $n$ by some action $a$, the following inequality holds:

$$h(n) \leq c(n, a, n') + h(n')$$

*(triangle inequality)*

- Every consistent heuristic is admissible.
- A* with a consistent heuristic is cost-optimal.
What If the Heuristic Is Inadmissible?

In such a situation, A* may or may not be cost-optimal.

Two of the cases where A* finds an optimal path:

- $h$ is admissible for all the nodes on one optimal path.

- $h(n)$ does not overestimate the cost on each node $n$ by more than the difference between the costs of the optimal and the second-best paths.
A contour labeled by a cost $c$ encloses all the nodes $n$ with $f(n) = g(n) + h(n) \leq c$. 

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Drobota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsoua</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
A contour labeled by a cost $c$ encloses all the nodes $n$ with $f(n) = g(n) + h(n) \leq c$. 

(393 = 140 + 253)
Some Search Contours

- Dijkstra’s algorithm (i.e., uniform search) would have contours of $g$-cost to “circle” around the start state.
Some Search Contours

- Dijkstra’s algorithm (i.e., uniform search) would have contours of $g$-cost to “circle” around the start state.

$V_k = \{u_1 = s, \ldots, u_k\}$

Uses relaxation:

$$d(v) = \min\{ d(v), d(u_{k+1}) + w(u_{k+1}, v) \}$$
Dijkstra’s algorithm (i.e., uniform search) would have contours of $g$-cost to “circle” around the start state.

Uses relaxation:

\[ d(v) = \min \{ d(v), d(u_{k+1}) + w(u_{k+1}, v) \} \]
Some Search Contours

- Dijkstra’s algorithm (i.e., uniform search) would have contours of $g$-cost to “circle” around the start state.

\[ V_k = \{ u_1 = s, \ldots, u_k \} \]

Uses relaxation:

\[ d(v) = \min\{ d(v), d(u_{k+1}) + w(u_{k+1}, v) \} \]
Some Search Contours

- Dijkstra’s algorithm (i.e., uniform search) would have contours of $g$-cost to “circle” around the start state.

\[ V_k = \{ u_1 = s, ..., u_k \} \]

Uses relaxation:

\[ d(v) = \min \{ d(v), d(u_{k+1}) + w(u_{k+1}, v) \} \]

- With a good $h$, the $g + h$ bands will stretch toward a goal state.
Some Search Contours

- Dijkstra’s algorithm (i.e., uniform search) would have contours of $g$-cost to “circle” around the start state.

\[ V_k = \{u_1 = s, \ldots, u_k\} \]

Uses relaxation:

\[ d(v) = \min\{d(v), d(u_{k+1}) + w(u_{k+1}, v)\} \]

- With a good $h$, the $g + h$ bands will stretch toward a goal state.
Monotonicity?

- The $g$ cost increases along a path because action costs are positive.

Cost at $n$: $g(n) + h(n)$

Cost at its successor $n'$: $g(n) + c(n, a, n') + h(n') = g(n')$
Monotonicity?

- The $g$ cost increases along a path because action costs are positive.

Cost at $n$: $g(n) + h(n)$

Cost at its successor $n'$: $g(n) + c(n, a, n') + h(n')$

The path’s cost increases *monotonically* iff

$$g(n) + h(n) \leq g(n) + c(n, a, n') + h(n')$$
Monotonicity?

The $g$ cost increases along a path because action costs are positive.

Cost at $n$: $g(n) + h(n)$

Cost at its successor $n'$: $g(n) + c(n, a, n') + h(n')$

The path’s cost increases monotonically iff

$$g(n) + h(n) \leq g(n) + c(n, a, n') + h(n')$$

$$\iff$$

$$h(n) \leq c(n, a, n') + h(n')$$

(consistent heuristic)
Consecutive Nodes Scored the Same

\[ g(n) + h(n) = g(n) + c(n, a, n') + h(n') \]
\[ = g(n) + c(n, a, n') + c(n', a, n'') + h(n'') \]
\[ = g(n'') \]
Consecutive Nodes Scored the Same

\[ g(n) + h(n) = g(n) + c(n, a, n') + h(n') \]

\[ = g(n) + c(n, a, n') + c(n', a, n'') + h(n'') \]

\[ = g(n'') \]

\[ h(n) - h(n') = g(n') - g(n) \]

\[ h(n') - h(n'') = g(n'') - g(n') \]
Consecutive Nodes Scored the Same

\[ g(n) + h(n) = g(n) + c(n, a, n') + h(n') \]
\[ = g(n) + c(n, a, n') + c(n', a, n'') + h(n'') \]
\[ = g(n'') \]

\[ h(n) - h(n') = g(n') - g(n) \]
\[ h(n') - h(n'') = g(n'') - g(n') \]

\[ h \text{ decreases as much as } g \text{ increases after an action.} \]
Efficiency of A*

\( h \): admissible

\( C^* \): cost of the optimal solution path

- A* will expand every node reachable via a sequence of nodes that have costs < \( C^* \).

- A* will not expand any node \( n \) with \( f(n) > C^* \).

- A* might expand a node \( n \) with cost \( f(n) = C^* \) before selecting a goal node.
Efficiency of A*

\( h \): admissible

\( C^* \): cost of the optimal solution path

- A* will expand every node reachable via a sequence of nodes that have costs < \( C^* \).

- A* will not expand any node \( n \) with \( f(n) > C^* \).

- A* might expand a node \( n \) with cost \( f(n) = C^* \) before selecting a goal node.

A* prunes away nodes unnecessary for finding an optimal solution.
Efficiency of A*

$h$: admissible
$C^*$: cost of the optimal solution path

- A* will expand every node reachable via a sequence of nodes that have costs $< C^*$.
- A* will not expand any node $n$ with $f(n) > C^*$.
- A* might expand a node $n$ with cost $f(n) = C^*$ before selecting a goal node.

A* prunes away nodes unnecessary for finding an optimal solution.

- A* may take exponential time with a poor heuristic function.
II. Sacrificing Search

- A* explores a lot of nodes due to equal weighting of $g$ and $h$ in $f = g + h$ which often distracts it from the optimal path.

- Can explore fewer nodes if we are okay with satisficing (suboptimal but “good enough”) solutions.
II. Sacrificing Search

◆ A* explores a lot of nodes due to equal weighting of $g$ and $h$ in $f = g + h$ which often distracts it from the optimal path.

- Can explore fewer nodes if we are okay with satisficing (suboptimal but “good enough”) solutions.

Use an inadmissible heuristic.
II. Sacrificing Search

- A* explores a lot of nodes due to equal weighting of $g$ and $h$ in $f = g + h$ which often distracts it from the optimal path.

- Can explore fewer nodes if we are okay with satisficing (suboptimal but “good enough”) solutions.

Use an inadmissible heuristic.

Idea of detour index: multiplier applied to the straight-line distance.

e.g., a detour index of 1.3 implies a good estimate of 13 miles between two cities that are 10 miles apart.
Weighted A*

Evaluation function: \( f(n) = g(n) + W \times h(n) \) for some \( W > 1 \).

A* search
Gray bars: obstacles
Dots: reached states
Weighted A*

Evaluation function: $f(n) = g(n) + W \times h(n)$ for some $W > 1$. 

A* search
Gray bars: obstacles
Dots: reached states

Weighted A* search
($W = 2$ on the same grid)
Weighted A* As a Generalization

- Weighted A* finds a solution with cost between $C^*$ and $W \times C^*$.
- Cost is usually much closer to $C^*$ in practice.

Weighted A* search: $g(n) + W \times h(n)$ \hspace{1cm} (1 \leq W < \infty)
Weighted A* As a Generalization

- Weighted A* finds a solution with cost between $C^*$ and $W \times C^*$.
- Cost is usually much closer to $C^*$ in practice.

<table>
<thead>
<tr>
<th>Search Type</th>
<th>Formula</th>
<th>$W$ Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A* search</td>
<td>$g(n) + h(n)$</td>
<td>$W = 1$</td>
</tr>
<tr>
<td>Uniform-cost search</td>
<td>$g(n)$</td>
<td>$W = 0$</td>
</tr>
<tr>
<td>Best-cost search</td>
<td>$h(n)$</td>
<td>$W = \infty$</td>
</tr>
<tr>
<td>Weighted A* search</td>
<td>$g(n) + W \times h(n)$</td>
<td>$1 \leq W &lt; \infty$</td>
</tr>
</tbody>
</table>
Memory-Bounded Search

- **Beam search** keeps the $k$ nodes with the best $f$ scores.
  - Less memory and faster execution.
  - Incomplete and suboptimal.
Memory-Bounded Search

- **Beam search** keeps the $k$ nodes with the best $f$ scores.
  - ♠ Less memory and faster execution.
  - ♣ Incomplete and suboptimal.

- **Iterative-deepening A* search (IDA*)**
  - The cutoff at each iteration is the $f$-cost.
Memory-Bounded Search

- **Beam search** keeps the $k$ nodes with the best $f$ scores.
  - ♦ Less memory and faster execution.
  - ♣ Incomplete and suboptimal.

- **Iterative-deepening A* search (IDA*)**
  - The cutoff at each iteration is the $f$-cost.
    \[ S = \{n| n \text{ generated in the previous iteration and } f(n) > \text{old } f_{\text{cutoff}} \} \]
    (previous iteration)
Memory-Bounded Search

- **Beam search** keeps the $k$ nodes with the best $f$ scores.
  - Less memory and faster execution.
  - Incomplete and suboptimal.

- **Iterative-deepening A* search (IDA*)**
  - The cutoff at each iteration is the $f$-cost.
    
    $$S = \{ n \mid n \text{ generated in the previous iteration and } f(n) > \text{old } f_{\text{cutoff}} \}$$
    
    
    $$\text{new } f_{\text{cutoff}} \leftarrow \min_{n \in S} f(n)$$
    
    (previous iteration)
    
    (current iteration)
Memory-Bounded Search

- **Beam search** keeps the $k$ nodes with the best $f$ scores.
  - Less memory and faster execution.
  - Incomplete and suboptimal.

- **Iterative-deepening A* search (IDA*)**
  - The cutoff at each iteration is the $f$-cost.
    
    $$S = \{ n \mid n \text{ generated in the previous iteration and } f(n) > \text{old } f_{\text{cutoff}} \}$$
    
    $$\text{new } f_{\text{cutoff}} \leftarrow \min_{n \in S} f(n)$$

  - Steady progress towards the goal if $f$-cost of every path is an integer.
    
    $$\leq C^* \text{ iterations}$$
Memory-Bounded Search

- **Beam search** keeps the \( k \) nodes with the best \( f \) scores.
  - Less memory and faster execution.
  - Incomplete and suboptimal.

- **Iterative-deepening A* search (IDA*)**
  - The cutoff at each iteration is the \( f \)-cost.
    
    \[ S = \{ n | n \text{ generated in the previous iteration and } f(n) > \text{old } f_{\text{cutoff}} \} \]
    
    \[
    \text{new } f_{\text{cutoff}} \leftarrow \min_{n \in S} f(n) \\
    \text{(current iteration)}
    \]

  - Steady progress towards the goal if \( f \)-cost of every path is an integer.
    
    \[ \leq C^* \text{ iterations} \]

  - In the worst case, \#iterations = \#states
Recursive Best-First Search (RBFS)

\[ f_{\text{limit}}(v) \]: \( f \)-value of the best alternative path from any ancestor of the node \( v \).

- Best-first search if at the currently visited node \( v \) it holds that \( f(v) \leq f_{\text{limit}}(v) \).
Recursive Best-First Search (RBFS)

- $f_{\text{limit}}(v)$: $f$-value of the best alternative path from any ancestor of the node $v$.

- Best-first search if at the currently visited node $v$ it holds that $f(v) \leq f_{\text{limit}}(v)$.

- Otherwise (i.e., $f(v) > f_{\text{limit}}(v)$),
Recursive Best-First Search (RBFS)

$f_{limit}(v)$: $f$-value of the best alternative path from any ancestor of the node $v$.

- Best-first search if at the currently visited node $v$ it holds that $f(v) \leq f_{limit}(v)$.
- Otherwise (i.e., $f(v) > f_{limit}(v)$),
  - unwinds back to the alternative path $P$;
Recursive Best-First Search (RBFS)

\( f_{\text{limit}}(v) \): \( f \)-value of the best alternative path from any ancestor of the node \( v \).

- Best-first search if at the currently visited node \( v \) it holds that \( f(v) \leq f_{\text{limit}}(v) \).
- Otherwise (i.e., \( f(v) > f_{\text{limit}}(v) \)),
  - unwinds back to the alternative path \( P \);
Recursive Best-First Search (RBFS)

$f_{\text{limit}}(v)$: $f$-value of the best alternative path from any ancestor of the node $v$.

- Best-first search if at the currently visited node $v$ it holds that $f(v) \leq f_{\text{limit}}(v)$.
- Otherwise (i.e., $f(v) > f_{\text{limit}}(v)$),
  - unwinds back to the alternative path $P$;
  - update the $f$-value of every node along the path $Q$ with the best $f$-value of its children.
RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[ 140 \times 2 + 366 = 646 = 140 + 90 + 176 \]
RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[
140 \times 2 + 366 = 646
\]

\[
= 140 + 90 + 176
\]
RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[ f_{\text{limit}}(\text{Arad}) = \infty \]

\[ f_{\text{limit}}(\text{Sibiu}) = 447 \]

\[ 140 \times 2 + 366 = 646 \]

\[ = 140 + 90 + 176 \]
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[ 140 \times 2 + 366 = 646 \]
\[ = 140 + 90 + 176 \]
RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[
f_{limit}(Sibiu) + 140 \times 2 + 366 = 646
\]

\[
f_{limit}(Sibiu) + 140 + 90 + 176 > 415
\]
RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[ f_{\text{limit}}(\text{Arad}) \]

\[ f_{\text{limit}}(\text{Sibiu}) \]

\[ 140 \times 2 + 366 = 646 \]

\[ = 140 + 90 + 176 \]

\[ > 415 \]
RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[ f_{\text{limit}}(\text{Arad}) \]

\[ f_{\text{limit}}(\text{Sibiu}) \]

\[ 140 \times 2 + 366 = 646 \]

\[ = 140 + 90 + 176 \]

(b) After unwinding back to Sibiu and expanding Fagaras

\[ f_{\text{limit}}(\text{Sibiu}) \]
RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[ f_{\text{limit}}(Arad) \]

\[ f_{\text{limit}}(Sibiu) \]

\[ 140 \times 2 + 366 = 646 \]

\[ = 140 + 90 + 416 \]

(b) After unwinding back to Sibiu and expanding Fagaras

\[ f_{\text{limit}}(Arad) \]

\[ f_{\text{limit}}(Sibiu) \]

\[ 140 \times 2 + 366 = 646 \]

\[ = 140 + 90 + 416 \]
RBFS Example

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

\[ f_{\text{limit}}(\text{Arad}) \]

\[ f_{\text{limit}}(\text{Sibiu}) \]

140 \times 2 + 366 = 646

= 140 + 90 + 176

(b) After unwinding back to Sibiu and expanding Fagaras

\[ f_{\text{limit}}(\text{Sibiu}) \]

140

\[ \times \]

2

+ 366

= 140 + 90 + 176

> 415
RBFS Example (cont’d)

(b) After unwinding back to Sibiu and expanding Fagaras

(c) After switching back to Rimnicu Vilcea and expanding Pitesti
RBFS Example (cont’d)

(b) After unwinding back to Sibiu and expanding Fagaras

(c) After switching back to Rimnicu Vilcea and expanding Pitesti
RBFS Example (cont’d)

(b) After unwinding back to Sibiu and expanding Fagaras

(c) After switching back to Rimnicu Vilcea and expanding Pitesti
RBFS Summary

- Optimal with admissible heuristic function $h(n)$.
- Space complexity $O(bd)$.
- Time complexity difficult to analyze, depending on
  - accuracy of $h(n)$
  - how often the best path changes
- Slightly more efficient than IDA*.
- Both IDA* and RBFS suffering from using too little memory and may explore the same state multiple times.
III. 8-Puzzle: Large Search Space

Start State

Goal State
III. 8-Puzzle: Large Search Space

\[ \frac{9!}{2} = 181,400 \text{ reachable states from start.} \]
III. 8-Puzzle: Large Search Space

\[ \frac{9!}{2} = 181,400 \text{ reachable states from start.} \]

\[ \frac{16!}{2} > 10^{13} \text{ reachable states for the 15-puzzle!} \]
Two Heuristics

Heuristics are needed for searching the vast state space.

Start State

Goal State
Two Heuristics

Heuristics are needed for searching the vast state space.

\[ h_1 = \text{# tiles misplaced} \]
Two Heuristics

Heuristics are needed for searching the vast state space.

\[ h_1 = \# \text{ tiles misplaced} \]

Misplaced tiles: 
1, 2, 3, 4, 5, 6, 7, 8
Two Heuristics

Heuristics are needed for searching the vast state space.

\[ h_1 = \text{# tiles misplaced} \]

Misplaced tiles:
1, 2, 3, 4, 5, 6, 7, 8

\[ h_1 = 8 \]
Two Heuristics

Heuristics are needed for searching the vast state space.

$\diamond h_1 = \# \text{ tiles misplaced}$

Admissible: any tile out of place will require $\geq 1$ move to fix.

Misplaced tiles: 1, 2, 3, 4, 5, 6, 7, 8

$\Rightarrow h_1 = 8$
Two Heuristics

Heuristics are needed for searching the vast state space.

- $h_1 =$ # tiles misplaced
  - Admissible: any tile out of place will require $\geq 1$ move to fix.

- $h_2 =$ sum of Manhattan distances of the tiles from their goal positions

Misplaced tiles: 1, 2, 3, 4, 5, 6, 7, 8
Heuristics are needed for searching the vast state space.

- \( h_1 = \) \# tiles misplaced

Admissible: any tile out of place will require \( \geq 1 \) move to fix.

- \( h_2 = \) sum of Manhattan distances of the tiles from their goal positions

Misplaced tiles: 1, 2, 3, 4, 5, 6, 7, 8

<table>
<thead>
<tr>
<th>Tile</th>
<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>
<tbody>
<tr>
<td>Manhattan distance</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Heuristics are needed for searching the vast state space.

- $h_1 = \# \text{ tiles misplaced}$
  
  Admissible: any tile out of place will require $\geq 1$ move to fix.

- $h_2 = \text{sum of Manhattan distances of the tiles from their goal positions}$
Two Heuristics

Heuristics are needed for searching the vast state space.

- $h_1 = \# \text{ tiles misplaced}$
  
  **Admissible**: any tile out of place will require $\geq 1$ move to fix.

- $h_2 = \text{sum of Manhattan distances of the tiles from their goal positions}$
  
  **Admissible**: every move reduces the Manhattan distance of only one tile by $\leq 1$.  

Misplaced tiles: 1, 2, 3, 4, 5, 6, 7, 8

<table>
<thead>
<tr>
<th>Tile</th>
<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>
<tbody>
<tr>
<td>Manhattan distance</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

$h_2 = 18$
Two Heuristics

Heuristics are needed for searching the vast state space.

- $h_1 = \# \text{ tiles misplaced}$
  - Admissible: any tile out of place will require $\geq 1$ move to fix.

- $h_2 = \text{sum of Manhattan distances of the tiles from their goal positions}$
  - Admissible: every move reduces the Manhattan distance of only one tile by $\leq 1$.

Neither heuristic overestimates the shortest solution (26 actions for the problem instance).
Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

If A* generates \( N \) nodes to find a solution at depth \( d \), then its effective branching factor \( b^* \) is the root of the following equation:

\[
N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d
\]
Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

If A* generates $N$ nodes to find a solution at depth $d$, then its effective branching factor $b^*$ is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

Intuitively, the $N + 1$ nodes handled by A* would fill a tree of height $d$ in which every node at depth $< d$ has exactly $b^*$ children.
Quality of a heuristic is often measured by the effective branching factor.

If A* generates $N$ nodes to find a solution at depth $d$, then its effective branching factor $b^*$ is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

$$\iff (b^*)^{d+1} + (N + 1)b^* + N = 0$$

Intuitively, the $N + 1$ nodes handled by A* would fill a tree of height $d$ in which every node at depth $< d$ has exactly $b^*$ children.
Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

If A* generates $N$ nodes to find a solution at depth $d$, then its effective branching factor $b^*$ is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

$$\Rightarrow (b^*)^{d+1} + (N + 1)b^* + N = 0$$

(\textit{polynomial root finding} – Com S 477/577)

Intuitively, the $N + 1$ nodes handled by A* would fill a tree of height $d$ in which every node at depth $< d$ has exactly $b^*$ children.
Heuristic Accuracy on Performance

Quality of a heuristic is often measured by the effective branching factor.

If A* generates $N$ nodes to find a solution at depth $d$, then its effective branching factor $b^*$ is the root of the following equation:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

$$\iff (b^*)^{d+1} + (N + 1)b^* + N = 0$$

*(polynomial root finding – Com S 477/577)*

Intuitively, the $N + 1$ nodes handled by A* would fill a tree of height $d$ in which every node at depth $< d$ has exactly $b^*$ children.

e.g. A* finds a solution at depth 5 using 52 nodes has $b^* = 1.92$. 
## Performance Comparison on 8-Puzzle

<table>
<thead>
<tr>
<th>$d$</th>
<th>BFS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
<th>BFS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>128</td>
<td>24</td>
<td>19</td>
<td>2.01</td>
<td>1.42</td>
<td>1.34</td>
</tr>
<tr>
<td>8</td>
<td>368</td>
<td>48</td>
<td>31</td>
<td>1.91</td>
<td>1.40</td>
<td>1.30</td>
</tr>
<tr>
<td>10</td>
<td>1033</td>
<td>116</td>
<td>48</td>
<td>1.85</td>
<td>1.43</td>
<td>1.27</td>
</tr>
<tr>
<td>12</td>
<td>2672</td>
<td>279</td>
<td>84</td>
<td>1.80</td>
<td>1.45</td>
<td>1.28</td>
</tr>
<tr>
<td>14</td>
<td>6783</td>
<td>678</td>
<td>174</td>
<td>1.77</td>
<td>1.47</td>
<td>1.31</td>
</tr>
<tr>
<td>16</td>
<td>17270</td>
<td>1683</td>
<td>364</td>
<td>1.74</td>
<td>1.48</td>
<td>1.32</td>
</tr>
<tr>
<td>18</td>
<td>41558</td>
<td>4102</td>
<td>751</td>
<td>1.72</td>
<td>1.49</td>
<td>1.34</td>
</tr>
<tr>
<td>20</td>
<td>91493</td>
<td>9905</td>
<td>1318</td>
<td>1.69</td>
<td>1.50</td>
<td>1.34</td>
</tr>
<tr>
<td>22</td>
<td>175921</td>
<td>22955</td>
<td>2548</td>
<td>1.66</td>
<td>1.50</td>
<td>1.34</td>
</tr>
<tr>
<td>24</td>
<td>290082</td>
<td>53039</td>
<td>5733</td>
<td>1.62</td>
<td>1.50</td>
<td>1.36</td>
</tr>
<tr>
<td>26</td>
<td>395355</td>
<td>110372</td>
<td>10080</td>
<td>1.58</td>
<td>1.50</td>
<td>1.35</td>
</tr>
<tr>
<td>28</td>
<td>463234</td>
<td>202565</td>
<td>22055</td>
<td>1.53</td>
<td>1.49</td>
<td>1.36</td>
</tr>
</tbody>
</table>

**Figure 3.26** Comparison of the search costs and effective branching factors for 8-puzzle problems using breadth-first search, $A^*$ with $h_1$ (misplaced tiles), and $A^*$ with $h_2$ (Manhattan distance). Data are averaged over 100 puzzles for each solution length $d$ from 6 to 28.
Given two heuristic functions $h_1$ and $h_2$, we say $h_2$ dominates $h_1$ if $h_2(n) \geq h_1(n)$ at every node $n$. 
#Misplaced Tiles vs Manhattan Distance

Given two heuristic functions $h_1$ and $h_2$, we say $h_2$ *dominates* $h_1$ if $h_2(n) \geq h_1(n)$ at every node $n$.

If $h_2$ dominates $h_1$, A* using $h_2$ will not expand more nodes than using $h_1$. 
Given two heuristic functions $h_1$ and $h_2$, we say $h_2$ dominates $h_1$ if $h_2(n) \geq h_1(n)$ at every node $n$.

If $h_2$ dominates $h_1$, A* using $h_2$ will not expand more nodes than using $h_1$.

A* expands every node $n$ with $f(n) < C^*$ (optimal cost).
Given two heuristic functions $h_1$ and $h_2$, we say $h_2$ dominates $h_1$ if $h_2(n) \geq h_1(n)$ at every node $n$.

If $h_2$ dominates $h_1$, A* using $h_2$ will not expand more nodes than using $h_1$.

A* expands every node $n$ with $f(n) < C^*$ (optimal cost).

$\Updownarrow$

A* expands every node $n$ with $h(n) < C^* - g(n)$. 
# Misplaced Tiles vs Manhattan Distance

Given two heuristic functions $h_1$ and $h_2$, we say $h_2$ dominates $h_1$ if $h_2(n) \geq h_1(n)$ at every node $n$.

If $h_2$ dominates $h_1$, A* using $h_2$ will not expand more nodes than using $h_1$.

A* expands every node $n$ with $f(n) < C^*$ (optimal cost).

A* expands every node $n$ with $h(n) < C^* - g(n)$.

Suppose that a node $n$ is expanded by A* with $h_2$, and $h_2(n) < C^* - g(n)$.
Given two heuristic functions $h_1$ and $h_2$, we say $h_2$ dominates $h_1$ if $h_2(n) \geq h_1(n)$ at every node $n$.

If $h_2$ dominates $h_1$, A* using $h_2$ will not expand more nodes than using $h_1$.

\[ A^* \text{ expands every node } n \text{ with } f(n) < C^* \text{ (optimal cost).} \]

\[ \updownarrow \]

\[ A^* \text{ expands every node } n \text{ with } h(n) < C^* - g(n). \]

Suppose that a node $n$ is expanded by A* with $h_2$, and $h_2(n) < C^* - g(n)$.

\[ \downarrow \]

\[ h_1(n) \leq h_2(n) < C^* - g(n) \]

\[ * \quad 222111(n) \leq h_2(n) < C^* - g(n) \]
Given two heuristic functions $h_1$ and $h_2$, we say $h_2$ dominates $h_1$ if $h_2(n) \geq h_1(n)$ at every node $n$.

If $h_2$ dominates $h_1$, A* using $h_2$ will not expand more nodes than using $h_1$.

$A^*$ expands every node $n$ with $f(n) < C^*$ (optimal cost).

$A^*$ expands every node $n$ with $h(n) < C^* - g(n)$.

Suppose that a node $n$ is expanded by $A^*$ with $h_2$, and $h_2(n) < C^* - g(n)$.

$$h_1(n) \leq h_2(n) < C^* - g(n)$$

$\star$ $222111(n) \leq h_2(n) < C^* - g(n)$

The node $n$ will be expanded by $A^*$ with $h_1$. 
Given two heuristic functions $h_1$ and $h_2$, we say $h_2$ dominates $h_1$ if $h_2(n) \geq h_1(n)$ at every node $n$.

If $h_2$ dominates $h_1$, A* using $h_2$ will not expand more nodes than using $h_1$.

A* expands every node $n$ with $f(n) < C^*$ (optimal cost).

\[
\uparrow
\]

A* expands every node $n$ with $h(n) < C^* - g(n)$.

Suppose that a node $n$ is expanded by A* with $h_2$, and $h_2(n) < C^* - g(n)$.

\[
\downarrow
\]

\[
h_1(n) \leq h_2(n) < C^* - g(n)
\]

\[
\ast \ 2 \ 2 \ 2 \ 1 \ 1 \ 1 (n) \leq h_2(n) < C^* - g(n)
\]

The node $n$ will be expanded by A* with $h_1$.

\[
\blacklozenge \ h_1 \text{ might cause other nodes to be expanded as well.}
\]
Generating Heuristics by Relaxation

- An admissible heuristic can be derived from exact solution cost of a relaxed problem.

  with fewer restrictions on the actions
Generating Heuristics by Relaxation

- An admissible heuristic can be derived from exact solution cost of a relaxed problem.

  with fewer restrictions on the actions

  Relaxation 1: A tile can move anywhere.
An admissible heuristic can be derived from exact solution cost of a relaxed problem.

with fewer restrictions on the actions

Relaxation 1: A tile can move anywhere.

\[ h_1 \] would give the length of the shortest solution.
An admissible heuristic can be derived from exact solution cost of a relaxed problem with fewer restrictions on the actions.

**Relaxation 1:** A tile can move anywhere. 
$h_1$ would give the length of the shortest solution.

**Relaxation 2:** A tile can move one square in any direction, even onto an occupied square.
Generating Heuristics by Relaxation

- An admissible heuristic can be derived from exact solution cost of a relaxed problem.

\[ h_1 \]

with fewer restrictions on the actions

**Relaxation 1**: A tile can move anywhere.

\[ h_1 \]

would give the length of the shortest solution.

**Relaxation 2**: A tile can move one square in any direction, even onto an occupied square.

\[ h_2 \]

would give the length of the shortest solution.
Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.
Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.
- But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.
Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.
- But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

\[ \downarrow \]

The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.
Admissibility & Consistency

• A solution in the original problem is also a solution in the relaxed problem.

• But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

↓

The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.

• Such heuristic is an exact cost for the relaxed problem.
Admissibility & Consistency

• A solution in the original problem is also a solution in the relaxed problem.

• But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

  \[ \downarrow \]

  The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.

• Such heuristic is an exact cost for the relaxed problem.

  \[ \downarrow \]

  It must satisfy the triangle inequality.
Admissibility & Consistency

- A solution in the original problem is also a solution in the relaxed problem.

- But an optimal solution to the relaxed problem may be shorter than an optimal solution to the original problem.

  \[\downarrow\]

  The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.

- Such heuristic is an exact cost for the relaxed problem.

  \[\downarrow\]

  It must satisfy the triangle inequality.

  \[\downarrow\]

  The heuristic is consistent.
Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$ and $Y$ is blank.
Formal specification of a problem (8-puzzle):

A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$ and $Y$ is blank.

Relaxation by removing one or two conditions
Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$ and $Y$ is blank.

Relaxation by removing one or two conditions

(a) A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$. 
Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$ and $Y$ is blank.

Relaxation by removing one or two conditions

(a) A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$. $\Rightarrow h_2$ (Manhattan distance)
Formal specification of a problem (8-puzzle):

A tile can move from square \( X \) to square \( Y \) if \( X \) is adjacent to \( Y \) and \( Y \) is blank.

\[ \Downarrow \]\hspace{1cm} \text{Relaxation by removing one or two conditions}

(a) A tile can move from square \( X \) to square \( Y \) if \( X \) is adjacent to \( Y \). \( h_2 \) (Manhattan distance)
(b) A tile can move from square \( X \) to square \( Y \) if \( Y \) is blank.
Formal specification of a problem (8-puzzle):

A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$ and $Y$ is blank.

- Relaxation by removing one or two conditions:
  - (a) A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$. $\Rightarrow h_2$ (Manhattan distance)
  - (b) A tile can move from square $X$ to square $Y$ if $Y$ is blank.
  - (c) A tile can move from square $X$ to square $Y$. 
A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$ and $Y$ is blank.

Relaxation by removing one or two conditions

(a) A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$. $\xrightarrow{\quad}$ $h_2$ (Manhattan distance)

(b) A tile can move from square $X$ to square $Y$ if $Y$ is blank.

(c) A tile can move from square $X$ to square $Y$. $\xrightarrow{\quad}$ $h_1$ (misplaced tiles)
Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$ and $Y$ is blank.

Relaxation by removing one or two conditions

(a) A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$. \( \Rightarrow \) \( h_2 \) (Manhattan distance)
(b) A tile can move from square $X$ to square $Y$ if $Y$ is blank.
(c) A tile can move from square $X$ to square $Y$. \( \Rightarrow \) \( h_1 \) (misplaced tiles)

Allows decomposition of the problem into 8 independent subproblems, one for moving each tile.
Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$ and $Y$ is blank.

Relaxation by removing one or two conditions

(a) A tile can move from square $X$ to square $Y$ if $X$ is adjacent to $Y$. $\iff h_2$ (Manhattan distance)
(b) A tile can move from square $X$ to square $Y$ if $Y$ is blank.
(c) A tile can move from square $X$ to square $Y$. $\iff h_1$ (misplaced tiles)

- The relaxed problems should be solved without search. Otherwise, evaluation of the corresponding heuristic will be expensive.

Allows decomposition of the problem into 8 independent subproblems, one for moving each tile.
Heuristics from Formal Specification

Formal specification of a problem (8-puzzle):

A tile can move from square \( X \) to square \( Y \) if \( X \) is adjacent to \( Y \) and \( Y \) is blank.

\[
\begin{align*}
\text{Relaxation by removing one or two conditions} \\
(a) \text{ A tile can move from square } X \text{ to square } Y \text{ if } X \text{ is adjacent to } Y. \quad \Rightarrow \ h_2 \text{ (Manhattan distance)} \\
(b) \text{ A tile can move from square } X \text{ to square } Y \text{ if } Y \text{ is blank.} \\
(c) \text{ A tile can move from square } X \text{ to square } Y. \quad \Rightarrow \ h_1 \text{ (misplaced tiles)}
\end{align*}
\]

- The relaxed problems should be solved without search. Otherwise, evaluation of the corresponding heuristic will be expensive.

- Program ABSOLVER generates heuristics automatically from problem definitions, including the best one for the 8-puzzle and the first one for the Rubik’s Cube puzzle.

Allows decomposition of the problem into 8 independent subproblems, one for moving each tile.
Admissible heuristics $h_1, h_2, \ldots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_k(n)\}$$
Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \ldots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_k(n)\}$$

$h$ is *admissible* because $h_1, h_2, \ldots, h_k$ are.
Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \cdots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \cdots, h_k(n)\}$$

- $h$ is admissible because $h_1, h_2, \cdots, h_k$ are.
- $h$ dominates $h_1, h_2, \cdots, h_k$. 

Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \ldots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_k(n)\}$$

- $h$ is admissible because $h_1, h_2, \ldots, h_k$ are.
- $h$ dominates $h_1, h_2, \ldots, h_k$.
- $h$ takes longer to compute – at least $O(k)$. May randomly select one heuristic at each evaluation.
Multiple Heuristics Available

Admissible heuristics $h_1, h_2, \ldots, h_k$ are available but none is clearly better than the others.

Use a composite heuristic:

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_k(n)\}$$

- $h$ is *admissible* because $h_1, h_2, \ldots, h_k$ are.
- $h$ *dominates* $h_1, h_2, \ldots, h_k$.
- $h$ takes longer to compute – at least $O(k)$.
  May randomly select one heuristic at each evaluation.