Introduction to Artificial Intelligence

Informed Search

Bernhard Beckert

UNIVERSITÄT KOBLENZ-LANDAU

Winter Term 2004/2005
Outline

- Best-first search
- A* search
- Heuristics
Review: Tree search

function TREE-SEARCH(problem, fringe) returns a solution or failure

fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
loop do
  if fringe is empty then return failure
  node ← REMOVE-FIRST(fringe)
  if GOAL-TEST[problem] applied to STATE(node) succeeds then
    return node
  else
    fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
  end
Best-first search

Idea

Use an evaluation function for each node (estimate of "desirability")

Expand most desirable unexpanded node

Implementation

fringe is a queue sorted in decreasing order of desirability

Special cases

- Greedy search
- A* search
Romania with step costs in km

Straight–line distance to Bucharest

Arad 366
Bucharest 0
Craiova 160
Dobrogea 242
Eforie 161
Fagaras 178
Giurgiu 77
Hirsova 151
Iasi 226
Lugoj 244
Mehadia 241
Neamt 234
Oradea 380
Pitesti 98
Rimnicu Vilcea 193
Sibiu 253
Timisoara 329
Urziceni 80
Vaslui 199
Zerind 374
Greedy search

Heuristic

Evaluation function

\[ h(n) = \text{estimate of cost from } n \text{ to goal} \]

Greedy search expands the node that appears to be closest to goal

Example

\[ h_{SLD}(n) = \text{straight-line distance from } n \text{ to Bucharest} \]

Note

Unlike uniform-cost search the node evaluation function has nothing to do with the nodes explored so far
Greedy search: Example Romania

Arad
366
Greedy search: Example Romania

- Zerind
- Arad
- Sibiu
- Timisoara

Distances:
- Arad to Zerind: 374
- Arad to Timisoara: 329
- Arad to Sibiu: 253
Greedy search: Example Romania

- Arad
- Fagaras
- Oradea
- Rimnicu Vilcea
- Sibiu
- Timisoara
- Zerind

Distances:
- Arad to Fagaras: 176
- Arad to Oradea: 380
- Arad to Rimnicu Vilcea: 193
- Sibiu to Timisoara: 329
- Sibiu to Zerind: 374
Greedy search: Example Romania
Greedy search: Properties

Complete

Time

Space

Optimal
Greedy search: Properties

**Complete**  No

Can get stuck in loops

**Example:** Iasi to Oradea

Iasi → Neamt → Iasi → Neamt → ⋮

Complete in finite space with repeated-state checking

**Time**

**Space**

**Optimal**
Greedy search: Properties

**Complete**  No

Can get stuck in loops

**Example:** Iasi to Oradea

Iasi → Neamt → Iasi → Neamt → · · ·

Complete in finite space with repeated-state checking

**Time**  \(O(b^m)\)

**Space**

**Optimal**  No

Worst-case time same as depth-first search, worst-case space same as breadth-first. But a good heuristic can give dramatic improvement.
**Greedy search: Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Can get stuck in loops</td>
</tr>
<tr>
<td>Example:</td>
<td>Iasi to Oradea</td>
</tr>
<tr>
<td></td>
<td>Iasi → Neamt → Iasi → Neamt → · · ·</td>
</tr>
<tr>
<td></td>
<td>Complete in finite space with repeated-state checking</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>Optimal</td>
<td>No</td>
</tr>
</tbody>
</table>
Greedy search: Properties

Complete  No
Can get stuck in loops
Example: Iasi to Oradea
   Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ $\cdots$
Complete in finite space with repeated-state checking

Time  $O(b^m)$
Space  $O(b^m)$
Optimal  No
Greedy search: Properties

**Complete**  No
Can get stuck in loops

**Example:** Iasi to Oradea
Iasi → Neamt → Iasi → Neamt → ⋯
Complete in finite space with repeated-state checking

**Time**  $O(b^m)$

**Space**  $O(b^m)$

**Optimal**  No

**Note**
Worst-case time same as depth-first search,
Worst-case space same as breadth-first
But a good heuristic can give dramatic improvement
**A* search**

**Idea**

Avoid expanding paths that are already expensive

**Evaluation function**

\[ f(n) = g(n) + h(n) \]

where

\[ g(n) = \text{cost so far to reach } n \]

\[ h(n) = \text{estimated cost to goal from } n \]

\[ f(n) = \text{estimated total cost of path through } n \text{ to goal} \]
A* search: Admissibility

Admissibility of heuristic

$h(n)$ is admissible if

$$h(n) \leq h^*(n) \quad \text{for all } n$$

where $h^*(n)$ is the true cost from $n$ to goal
A* search: Admissibility

Admissibility of heuristic

$h(n)$ is admissible if

$$ h(n) \leq h^*(n) \quad \text{for all } n $$

where $h^*(n)$ is the true cost from $n$ to goal

Also required

$$ h(n) \geq 0 \quad \text{for all } n $$

In particular: $h(G) = 0$ for goal $G$
A* search: Admissibility

Admissibility of heuristic

$h(n)$ is admissible if

\[ h(n) \leq h^*(n) \quad \text{for all } n \]

where $h^*(n)$ is the true cost from $n$ to goal

Also required

\[ h(n) \geq 0 \quad \text{for all } n \]

In particular: $h(G) = 0$ for goal $G$

Example

Straight-line distance never overestimates the actual road distance
A* search: Admissibility

Theorem

A* search with admissible heuristic is optimal
A* search example

Arad
366 = 0 + 366
A* search example

- Sibiu: 393 = 140 + 253
- Timisoara: 447 = 118 + 329
- Zerind: 449 = 75 + 374
A* search example
A* search example

![A* search example diagram]
A* search example
A* search example
A* search: $f$-contours

A* gradually adds “$f$-contours” of nodes
Optimality of A* search: Proof

Suppose a suboptimal goal $G_2$ has been generated.
Let $n$ be an unexpanded node on a shortest path to an optimal goal $G$.

\[
\begin{align*}
  f(G_2) &= g(G_2) & \text{since } h(G_2) = 0 \\
  &> g(G) & \text{since } G_2 \text{ suboptimal} \\
  &= g(n) + h^*(n) \\
  &\geq g(n) + h(n) & \text{since } h \text{ is admissible} \\
  &= f(n)
\end{align*}
\]

Thus, A* never selects $G_2$ for expansion.
A* search: Properties

Complete

Time

Space

Optimal
A* search: Properties

Complete: Yes

(Unless there are infinitely many nodes $n$ with $f(n) \leq f(G)$)

Time

Space

Optimal
### $A^*$ search: Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>Exponential in</td>
</tr>
<tr>
<td>Space</td>
<td></td>
</tr>
<tr>
<td>Optimal</td>
<td></td>
</tr>
</tbody>
</table>

(unless there are infinitely many nodes $n$ with $f(n) \leq f(G)$)

Time: $\text{Exponential in } [\text{relative error in } h \times \text{length of solution}]$
A* search: Properties

Complete  Yes
(unless there are infinitely many nodes \( n \) with \( f(n) \leq f(G) \))

Time  Exponential in
[relative error in \( h \times \) length of solution]

Space  Same as time

Optimal
A* search: Properties

Complete  Yes

(Unless there are infinitely many nodes \( n \) with \( f(n) \leq f(G) \))

Time  Exponential in

[Relative error in \( h \times \) length of solution]

Space  Same as time

Optimal  Yes

B. Beckert: KI für IM – p.15
**A* search: Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complete</strong></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>(unless there are infinitely many nodes ( n ) with ( f(n) \leq f(G) ))</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>Exponential in</td>
</tr>
<tr>
<td></td>
<td>([\text{relative error in } h \times \text{length of solution}])</td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>Same as time</td>
</tr>
<tr>
<td><strong>Optimal</strong></td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note**

- \( A^* \) expands all nodes with \( f(n) < C^* \)
- \( A^* \) expands some nodes with \( f(n) = C^* \)
- \( A^* \) expands no nodes with \( f(n) > C^* \)
Admissible heuristics: Example 8-puzzle

Addmissible heuristics

\[ h_1(n) = \text{number of misplaced tiles} \]

\[ h_2(n) = \text{total \ Manhattan distance} \]
(i.e., no. of squares from desired location of each tile)
Admissible heuristics: Example 8-puzzle

Addmissible heuristics

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]
  (i.e., no. of squares from desired location of each tile)

In the example

\[ h_1(S) = \]

\[ h_2(S) = \]
Admissible heuristics: Example 8-puzzle

Addmissible heuristics

\[ h_1(n) = \text{number of misplaced tiles} \]

\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., no. of squares from desired location of each tile)

In the example

\[ h_1(S) = 6 \]

\[ h_2(S) = \]
Admissible heuristics: Example 8-puzzle

Addmissible heuristics

\[ h_1(n) = \text{number of misplaced tiles} \]

\[ h_2(n) = \text{total Manhattan distance} \]

\[ (i.e., \text{no. of squares from desired location of each tile}) \]

In the example

\[ h_1(S) = 6 \]

\[ h_2(S) = 2 + 0 + 3 + 1 + 0 + 1 + 3 + 4 = 14 \]
Dominance

**Definition**

$h_1, h_2$ two admissible heuristics

$h_2$ dominates $h_1$ if

$$h_2(n) \geq h_1(n) \quad \text{for all } n$$
**Dominance**

**Definition**

$h_1, h_2$ two admissible heuristics

$h_2$ dominates $h_1$ if

$$h_2(n) \geq h_1(n) \quad \text{for all } n$$

**Theorem**

If $h_2$ dominates $h_1$, then $h_2$ is better for search than $h_1$. 

B. Beckert: KI für IM – p.17
Dominance: Example 8-puzzle

**Typical search costs**

<table>
<thead>
<tr>
<th>d</th>
<th>IDS</th>
<th>IDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>3,473,941 nodes</td>
<td>539 nodes</td>
</tr>
<tr>
<td></td>
<td>( A^*(h_1) )</td>
<td>113 nodes</td>
</tr>
<tr>
<td></td>
<td>( A^*(h_2) )</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>too many nodes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( A^*(h_1) )</td>
<td>39,135 nodes</td>
</tr>
<tr>
<td></td>
<td>( A^*(h_2) )</td>
<td>1,641 nodes</td>
</tr>
</tbody>
</table>

\( d \): depth of first solution  
IDS: iterative deepening search
Relaxed problems

Finding good admissible heuristics is an art!

**Deriving admissible heuristics**

Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem.
Relaxed problems

Finding good admissible heuristics is an art!

Deriving admissible heuristics

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem

Example

If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then we get heuristic $h_1$

If the rules are relaxed so that a tile can move to any adjacent square, then we get heuristic $h_2$
Relaxed problems

Finding good admissible heuristics is an art!

Deriving admissible heuristics

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem

Example

If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then we get heuristic $h_1$

If the rules are relaxed so that a tile can move to any adjacent square, then we get heuristic $h_2$

Key point

The optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem