Artificial Intelligence: Search
Part 2: Heuristic search

Thomas Trappenberg

January 16, 2009

Based on the slides provided by Russell and Norvig, Chapter 4, Section 1–2,(4)
Outline

♦ Best-first search
♦ A* search
♦ Heuristics
function TREE-SEARCH(\textit{problem}, fringe) \textbf{returns} a solution, or failure
fringe ← INSERT(MAKE-NODE(INITIAL-STATE[\textit{problem}]), fringe)
loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[\textit{problem}] applied to STATE(node) succeeds return node
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
end loop

A strategy is defined by picking the \textbf{order of node expansion}
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

<table>
<thead>
<tr>
<th>Place</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>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</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>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vâlcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timișoara</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>
Greedy search

Evaluation function $h(n)$ (heuristic)

$= \text{estimate of cost from } n \text{ to the closest goal}$

E.g., $h_{\text{SLD}}(n) = \text{straight-line distance from } n \text{ to Bucharest}$

Greedy search expands the node that \textbf{appears} to be closest to goal
Greedy search example
Greedy search example
Greedy search example

![Diagram of a greedy search example with cities and distances]
Greedy search example
Properties of greedy search

**Complete** No—can get stuck in loops, e.g.,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

**Time** $O(b^m)$, but a good heuristic can give dramatic improvement

**Space** $O(b^m)$—keeps all nodes in memory

**Optimal** No
**A* search**

**Idea:** avoid expanding paths that are already expensive

Evaluation function \( f(n) = g(n) + h(n) \)

- \( g(n) \) = cost so far to reach \( n \)
- \( h(n) \) = estimated cost to goal from \( n \)
- \( f(n) \) = estimated total cost of path through \( n \) to goal

**A* search uses an admissible heuristic**

i.e., \( h(n) \leq h^*(n) \) where \( h^*(n) \) is the **true** cost from \( n \).

(Also require \( h(n) \geq 0 \), so \( h(G) = 0 \) for any goal \( G \).)

**E.g.,** \( h_{SLD}(n) \) never overestimates the actual road distance

**Theorem:** A* search is optimal
A* search example

Arad

366 = 0 + 366
A* search example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example

A* Search Example
A* search example
A* search example

Graph showing the A* search for cities, with distances and heuristic values.
A* search example

Starting from Arad, the algorithm explores the graph by considering the sum of the path cost from Arad to the current node and the heuristic cost from the current node to the goal. The algorithm visits nodes in order of their total cost. The graph shows the total cost of the path from Arad to each node, e.g., 447 = 118 + 329 for Zerind.

Key nodes and distances:
- Arad → Sibiu: 646 = 280 + 366
- Arad → Timisoara: 447 = 118 + 329
- Arad → Zerind: 449 = 75 + 374
- Sibiu → Oradea: 671 = 291 + 380
- Sibiu → Fagaras: 591 = 338 + 253
- Sibiu → Bucharest: 450 = 450 + 0
- Sibiu → Craiova: 526 = 366 + 160
- Sibiu → Pitesti: 417 = 317 + 100
- Sibiu → Sibiu: 553 = 300 + 253

The algorithm continues to explore other paths until the goal is reached or all possible paths are exhausted.
A* search example
Optimality of A*

A* expands nodes in order of increasing $f$ value*

Gradually adds “$f$-contours” of nodes (cf. breadth-first adds layers)
Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$
Properties of $A^*$

**Complete** Yes, unless there are infinitely many nodes with $f \leq f(G)$

**Time** Exponential in $[\text{relative error in } h \times \text{length of soln.}]$

**Space** Keeps all nodes in memory

**Optimal** Yes—cannot expand $f_{i+1}$ until $f_i$ is finished

$A^*$ expands all nodes with $f(n) < C^*$

$A^*$ expands some nodes with $f(n) = C^*$

$A^*$ expands no nodes with $f(n) > C^*$
A heuristic is **consistent** if

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

If \( h \) is consistent, we have

\[
\begin{align*}
    f(n') &= g(n') + h(n') \\
    &= g(n) + c(n, a, n') + h(n') \\
    &\geq g(n) + h(n) \\
    &= f(n)
\end{align*}
\]

I.e., \( f(n) \) is nondecreasing along any path.
Admissible heuristics

E.g., for the 8-puzzle:

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

\[
\begin{array}{ccc}
7 & 2 & 4 \\
5 & 6 & \\
8 & 3 & 1 \\
\end{array}
\quad
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & \\
\end{array}
\]

\( h_1(S) = 6 \)
\( h_2(S) = 4+0+3+3+1+0+2+1 = 14 \)
Dominance

If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$ and is better for search

Typical search costs:

- $d = 14$ \hspace{1cm} IDS = 3,473,941 nodes
  - $A^*(h_1) = 539$ nodes
  - $A^*(h_2) = 113$ nodes
- $d = 24$ \hspace{1cm} IDS $\approx$ 54,000,000,000 nodes
  - $A^*(h_1) = 39,135$ nodes
  - $A^*(h_2) = 1,641$ nodes

Given any admissible heuristics $h_a, h_b$,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates $h_a, h_b$
Relaxed problems

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

If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then \( h_1(n) \) gives the shortest solution.

If the rules are relaxed so that a tile can move to any adjacent square, then \( h_2(n) \) gives the shortest solution.

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.
Local beam search

**Idea:** keep $k$ states instead of 1; choose top $k$ of all their successors

Not the same as $k$ searches run in parallel!

Searches that find good states recruit other searches to join them

**Problem:** quite often, all $k$ states end up on same local hill

**Idea:** choose $k$ successors randomly, biased towards good ones

Observe the close analogy to natural selection!
Genetic algorithms

= stochastic local beam search + generate successors from pairs of states
Genetic algorithms contd.

GAs require states encoded as strings (GPs use )

Crossover helps iff substrings are meaningful components

GAs ≠ evolution: e.g., real genes encode replication machinery!
Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest $h$
  – incomplete and not always optimal

A* search expands lowest $g + h$
  – complete and optimal
  – also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems