Informed search algorithms

Chapter 4, Sections 1–2
Outline

♦ Best-first search
♦ A* search
♦ Heuristics
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-Front(fringe)
  if Goal-Test[problem] applied to State(node) succeeds return node
  fringe ← InsertAll(Expand(node, problem), fringe)

A strategy is defined by picking the 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

- 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

Chapter 4, Sections 1–2
Greedy search

Evaluation function $h(n)$ (heuristic)

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

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

Greedy search expands the node that appears to be closest to goal
Greedy search example
Greedy search example
Greedy search example
Greedy search example
Properties of greedy search

Complete??
Properties of greedy search

**Complete**? No—can get stuck in loops, e.g., with Oradea as goal, Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

**Time**??
Properties of greedy search

**Complete??** No—can get stuck in loops, e.g.,
\[ \text{lasi} \rightarrow \text{Neamt} \rightarrow \text{lasi} \rightarrow \text{Neamt} \rightarrow \]
Complete in finite space with repeated-state checking

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

**Space??**
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
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

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

Arad

Sibiu

Zerind

Timisoara

447=118+329

646=280+366

415=239+176

671=291+380

413=220+193

449=75+374

Arad

Fagaras

Oradea

Rimnicu Vilcea
A* search example

\[
\begin{align*}
\text{Arad} & \rightarrow \text{Fagaras} & 646 = 280 + 366 \\
& \rightarrow \text{Oradea} & 415 = 239 + 176 \\
& \rightarrow \text{Rimnicu Vilcea} & 671 = 291 + 380 \\
\text{Sibiu} & \rightarrow \text{Timisoara} & 447 = 118 + 329 \\
& \rightarrow \text{Zerind} & 449 = 75 + 374 \\
\text{Craiova} & \rightarrow \text{Pitesti} & 526 = 366 + 160 \\
& \rightarrow \text{Sibiu} & 553 = 300 + 253
\end{align*}
\]
A* search example

Chapter 4, Sections 1-2
A* search example

Chapter 4, Sections 1-2
Optimality of A* (standard proof)

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

Since $f(G_2) = g(G_2)$ since $h(G_2) = 0$

$> g(G_1)$ since $G_2$ is suboptimal

$\geq f(n)$ since $h$ is admissible

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion
Optimality of A* (more useful)

**Lemma:** 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??
Properties of A*  

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

**Time?**
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**?
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complete</strong></td>
<td>Yes, unless there are infinitely many nodes with $f \leq f(G')$</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>Exponential in [relative error in $h$ $\times$ length of soln.]</td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>Keeps all nodes in memory</td>
</tr>
<tr>
<td><strong>Optimal</strong></td>
<td></td>
</tr>
</tbody>
</table>
Properties of A*

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

**Time**? Exponential in [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 \( f(n) > C^* \)
Proof of lemma: Consistency

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)

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

Start State \quad Goal State

\[
\begin{align*}
\frac{h_1(S)}{h_2(S)} = ??
\end{align*}
\]
**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)

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

\[
\begin{align*}
\frac{h_1(S)}{} & = ?? \quad 6 \\
\frac{h_2(S)}{} & = ?? \quad 4+0+3+3+1+0+2+1 = 14
\end{align*}
\]
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:

\[
\begin{align*}
  d = 14 & \quad \text{IDS} = 3,473,941 \text{ nodes} \\
  & \quad A^*(h_1) = 539 \text{ nodes} \\
  & \quad A^*(h_2) = 113 \text{ nodes} \\
  d = 24 & \quad \text{IDS} \approx 54,000,000,000 \text{ nodes} \\
  & \quad A^*(h_1) = 39,135 \text{ nodes} \\
  & \quad A^*(h_2) = 1,641 \text{ nodes}
\end{align*}
\]

Given any admissible heuristics \( h_a, h_b \),

\[
h(n) = \max(h_a(n), h_b(n))
\]

is also admissible and dominates \( h_a, h_b \)
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.
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