INFORMED SEARCH ALGORITHMS

CHAPTER 4, SECTIONS 1–2
Outline

♦ Best-first search
♦ A* search
♦ Heuristics
function Tree-Search\((\text{problem}, \text{fringe})\) returns a solution, or failure

\[
\text{fringe} \leftarrow \text{Insert}(\text{Make-Node(Initial-State[problem]}), \text{fringe})
\]

\[
\text{loop do}
\]

if \text{fringe} is empty then return failure

\[
\text{node} \leftarrow \text{Remove-Front} (\text{fringe})
\]

if Goal-Test[problem] applied to State(node) succeeds return node

\[
\text{fringe} \leftarrow \text{InsertAll}(\text{Expand}(\text{node}, \text{problem}), \text{fringe})
\]

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
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 \textit{appears} to be closest to goal
Greedy search example

Arad
366
Greedy search example

![Graph diagram showing Arad connected to Sibiu, Timisoara, and Zerind with distances 253, 329, and 374 respectively.]

Chapter 4, Sections 1–2
Greedy search example
Greedy search example

Chapter 4, Sections 1–2
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.,
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??
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 $f(n) = g(n) + h(n)$

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

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

$f(n) = \text{estimated total cost of path through } n \text{ to goal}$

A* search 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., $h_{\text{SLD}}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal
A* search example

Arad

366=0+366
A* search example

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

Chapter 4, Sections 1–2  20
A* search example
A* search example

Chapter 4, Sections 1–2
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 $G_1$.

\[
\begin{align*}
    f(G_2) &= g(G_2) \quad \text{since } h(G_2) = 0 \\
    &> g(G_1) \quad \text{since } G_2 \text{ is suboptimal} \\
    &\geq f(n) \quad \text{since } h \text{ is admissible}
\end{align*}
\]

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 [relative error in $h \times$ length of soln.]

**Space??**
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**
Properties of A*

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

**Time**?? Exponential in [relative error in $h \times$ 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)

\[ h_1(S) = ?? \]
\[ h_2(S) = ?? \]
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

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

\[
\begin{align*}
h(n) &= \max(h_a(n), h_b(n))
\end{align*}
\]

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.
Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once

Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour
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