INFORMED SEARCH ALGORITHMS

Chapter 3, Sections 3.5-3.6
Reading Assignment

• For Thursday: Chapter 4.3-4.5 (we’re skipping 4.1-4.2)

• For next week: Chapter 5
Q1: Suppose the pieces fit together exactly. Give formulation of the task as a search problem.
Q2: Identify a suitable uninformed search algorithm for this task, and explain why it is suitable.
Q3: Why does removing any one of the “fork” pieces make the problem unsolvable?
Q4: Give an upper bound on the total size of the state space defined for this formulation. (Ignore problem of overlapping pieces and loose ends. Reason primarily about max branching factor and max depth. Pretending unique pieces.)
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 km
- Bucharest: 0 km
- Craiova: 160 km
- Dobrogea: 242 km
- Eforie: 161 km
- Fagaras: 178 km
- Giurgiu: 77 km
- Hirsova: 151 km
- Iasi: 226 km
- Lugoj: 244 km
- Mehadia: 241 km
- Neamț: 234 km
- Oradea: 380 km
- Pitesti: 98 km
- Rimnicu Vâlcea: 193 km
- Sibiu: 253 km
- Timisoara: 329 km
- Urziceni: 80 km
- Vaslui: 199 km
- Zerind: 374 km
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 *appears* to be closest to goal
Greedy search example

Arad
366
Greedy search example

Chapter 4, Sections 1–2
Greedy search example

Chapter 4, Sections 1–2
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.,
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 $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

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

Chapter 4, Sections 1-2
A* search example

Arad
Sibiu
Timisoara
Zerind

447 = 118 + 329
449 = 75 + 374

646 = 280 + 366
415 = 239 + 176
671 = 291 + 380
413 = 220 + 193
A* search example

Chapter 4, Sections 1–2
A* search example

Chapter 4, Sections 1-2
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 $G_1$.

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

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion.
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***

<table>
<thead>
<tr>
<th><strong>Complete</strong></th>
<th>Yes, unless there are infinitely many nodes with ( f \leq f(G') )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td>Exponential in [relative error in ( h \times \text{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 \( h \times \) 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.
Exercise: Search Algs.

Consider the following scoring function for heuristic search:

\[ f(n) = w \times g(n) + (1 - w) \times h(n) \quad \text{where} \quad 0 \leq w \leq 1 \]

i. Which search algorithm do you get with \( w \) set to 0?
Exercise: Various Search Algs.

1) Prove that breadth-first search is a special case of uniform-cost search.
Exercise: Various Search Algs.

2) Prove that breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.
Exercise: Various Search Algs.

3) Prove that uniform-cost search is a special case of A* search
Since A* keeps all nodes in memory, it usually runs out of space before it runs out of time.

- **Memory-bounded heuristic search**
  - Iterative-deepening A* (IDA*), where cutoff is the f-cost (g+h), rather than the depth.
  - Recursive best-first search – like best-first search, but only uses linear space.
    - Keeps track of value of best alternative from any ancestor of current node.
    - If current node exceeds this limit, then recursion unwinds back to alternative path.
Memory-Bounded Heuristic Search

• Problem: IDA* and RBFS don’t use all the memory they could, leading to re-evaluation of states multiple times

• Memory-Bounded A* (MA*), and Simplified MA* (SMA*) are better.
**Simplified MA* (SMA*)**

- Proceeds like A*, expanding best leaf until memory is full
- Then, it drops worst leaf node (i.e., one with highest f value)
- If all leaf nodes have same f value, then delete the oldest node
- SMA*
  - Is **complete** if \( d \) is less than memory size
  - Is **optimal** if any optimal solution is reachable
    - Otherwise, returns best reachable solution
- But, on hard problems, SMA* thrashes between many candidate solution paths
  - Tradeoff between computation and memory
Admissible heuristics

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total \textit{Manhattan} distance} \]

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

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

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

Start State

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

Goal State

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

\[
\begin{align*}
h_1(S) &= 6 \\
h_2(S) &= 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14
\end{align*}
\]
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} \\
A^*(h_1) &= 539 \text{ nodes} \\
A^*(h_2) &= 113 \text{ nodes}
\end{align*}

\begin{align*}
d = 24 & \quad \text{IDS} \approx 54,000,000,000 \text{ nodes} \\
A^*(h_1) &= 39,135 \text{ nodes} \\
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 $h_a, h_b$
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.
Relaxed problems contd.

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
Can derive admissible heuristics from solution cost of subproblem of given problem

Example: 8-puzzle: subproblem is solution to getting tiles 1, 2, 3, 4 in place (or any 4 of the tiles)

Pattern databases: store exact solution costs for every possible subproblem instance (i.e., every configuration of the 4 tiles of the subproblem)

Then compute admissible heuristic by looking up subproblem in database
If subproblems are independent, then can add costs of subproblems to create admissible heuristic.

E.g., for 8-puzzle, 2 subproblems: 1-2-3-4, 5-6-7-8.

- Count cost of each subproblem only for the specified tiles (not all tiles)
- Then, the two subproblem costs can be added.
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
Reading Assignment

• For Thursday: Chapter 4.3-4.5 (we’re skipping 4.1-4.2)

• For next week: Chapter 5