INFORMED SEARCH ALGORITHMS

CHAPTER 4, SECTIONS 1–2
Outline

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

```plaintext
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

<table>
<thead>
<tr>
<th>Location</th>
<th>Step Cost (km)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Neamt</td>
<td>366</td>
</tr>
<tr>
<td>Oradea</td>
<td>140</td>
</tr>
<tr>
<td>Zerind</td>
<td>118</td>
</tr>
<tr>
<td>Timisoara</td>
<td>146</td>
</tr>
<tr>
<td>Dobreta</td>
<td>120</td>
</tr>
<tr>
<td>Sibiu</td>
<td>99</td>
</tr>
<tr>
<td>Fagaras</td>
<td>199</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</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>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Iasi</td>
<td>92</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Fagaras</td>
<td>92</td>
</tr>
<tr>
<td>Pitesti</td>
<td>142</td>
</tr>
<tr>
<td>Vaslui</td>
<td>86</td>
</tr>
<tr>
<td>Iasi</td>
<td>87</td>
</tr>
<tr>
<td>Neamt</td>
<td>87</td>
</tr>
<tr>
<td>Oradea</td>
<td>86</td>
</tr>
<tr>
<td>Eforie</td>
<td>90</td>
</tr>
<tr>
<td>Zerind</td>
<td>71</td>
</tr>
</tbody>
</table>

**Straight-line distance to Bucharest**

- Arad: 366 km
- Bucharest: 0 km
- Craiova: 160 km
- Dobreta: 242 km
- Eforie: 161 km
- Fagaras: 178 km
- Giurgiu: 77 km
- Hirsova: 151 km
- Iasi: 226 km
- Lugoj: 244 km
- Mehadia: 241 km
- Neamt: 234 km
- Oradea: 380 km
- Pitesti: 98 km
- Rimnicu Vilcea: 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 \textcolor{red}{appears} to be closest to goal
Greedy search example

Arad
366
Greedy search example

- Sibiu
  - 253

- Timisoara
  - 329

- Zerind
  - 374
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,

\[ \text{Iasi} \rightarrow \text{Neamt} \rightarrow \text{Iasi} \rightarrow \text{Neamt} \rightarrow \]

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.,
\[ \text{lasi} \rightarrow \text{Neamt} \rightarrow \text{lasi} \rightarrow \text{Neamt} \rightarrow \]
Complete in finite space with repeated-state checking

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

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

**Optimal**?
Properties of greedy search

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

\[ \text{Iasi } \rightarrow \text{Neamt } \rightarrow \text{Iasi } \rightarrow \text{Neamt } \rightarrow \]

Complete in finite space with repeated-state checking

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

**Space**? \( O(b^m) \)—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

Chapter 4, Sections 1-2
A* search example

Chapter 4, Sections 1-2
A* search example

Arad

Sibiu

Fagaras

Oradea

Rimnicu Vilcea

Craiova

Pitesti

Sibiu

Arad

415 = 239 + 176

671 = 291 + 380

447 = 118 + 329

526 = 366 + 160

417 = 317 + 100

553 = 300 + 253

526 = 366 + 160

417 = 317 + 100

553 = 300 + 253

Zerind

449 = 75 + 374

646 = 280 + 366

553 = 300 + 253

671 = 291 + 380

417 = 317 + 100
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 $G_1$.

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

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion.
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 \( h \times \text{length of soln.} \)]

**Space**
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**
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)

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

Start State

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

Goal State

\[
\begin{array}{ccc}
h_1(S) = ?? \\
h_2(S) = ??
\end{array}
\]
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)

\[ h_1(S) = 6 \]
\[ h_2(S) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14 \]
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:

\[
d = 14 \quad \text{IDS} = 3,473,941 \text{ nodes} \\
\text{A}^*(h_1) = 539 \text{ nodes} \\
\text{A}^*(h_2) = 113 \text{ nodes}
\]

\[
d = 24 \quad \text{IDS} \approx 54,000,000,000 \text{ nodes} \\
\text{A}^*(h_1) = 39,135 \text{ nodes} \\
\text{A}^*(h_2) = 1,641 \text{ nodes}
\]

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

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

is also admissible and dominates \( h_a, h_b \)
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](image)

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