Informed search algorithms

Chapter 4, Sections 1–2

Outline

◊ Best-first search
◊ A* search
◊ Heuristics

Review: Tree search

```
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)
   end loop
```

Greedy search

Evaluation function \( h(n) \) (heuristic) = estimate of cost from \( n \) to the closest goal

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

Greedy search expands the node that appears to be closest to goal

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

<table>
<thead>
<tr>
<th>City</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>Dobroiu</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>164</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Blajna</td>
<td>151</td>
</tr>
<tr>
<td>Eisai</td>
<td>226</td>
</tr>
<tr>
<td>Lupeni</td>
<td>244</td>
</tr>
<tr>
<td>Micolau</td>
<td>241</td>
</tr>
<tr>
<td>Nantus</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>96</td>
</tr>
<tr>
<td>Rimnoca</td>
<td>195</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Vaslui</td>
<td>80</td>
</tr>
<tr>
<td>Vatra</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy search example

Chapter 4, Sections 1-2

Greedy search example

Chapter 4, Sections 1-2

Chapter 4, Sections 1-2

Chapter 4, Sections 1-2

Chapter 4, Sections 1-2

Chapter 4, Sections 1-2

Chapter 4, Sections 1-2

Greedy search example

Chapter 4, Sections 1-2

Properties of greedy search

Chapter 4, Sections 1-2

Properties of greedy search

Chapter 4, Sections 1-2

Properties of greedy search

Chapter 4, Sections 1-2

Properties of greedy search

Chapter 4, Sections 1-2

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 →
- **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
### 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 $G_1$.

$$f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0$$

$$> g(G_1) \quad \text{since } G_2 \text{ is suboptimal}$$

$$\geq f(n) \quad \text{since } h \text{ is admissible}$$

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 $f_i < f_{i+1}$

---

**Diagram**

- **A* search example**
- **Optimality of A* (standard proof)**
- **Optimality of A* (more useful)**
Properties of \( A^- \)

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

**Time??** Exponential 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

\( \text{A}^- \text{ expands all nodes with } f(n) < C^* \)

\( \text{A}^- \text{ expands some nodes with } f(n) = C^* \)

\( \text{A}^- \text{ expands no nodes with } f(n) > C^* \)

---

**Proof of lemma: Consistency**

A heuristic is **consistent** if

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

If \( h \) is consistent, we have

\[
f(n') = g(n') + h(n')
= g(n) + c(n, a, n') + h(n')
\geq g(n) + h(n)
= f(n)
\]

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)

<table>
<thead>
<tr>
<th>Start State</th>
<th>Goal State</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 2 4 1 2 3</td>
<td>5 6 4 5 6 8</td>
</tr>
<tr>
<td>8 3 1</td>
<td>7 8</td>
</tr>
</tbody>
</table>

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

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:
\[ d = 14 \] IDS = 3,473,941 nodes
\[ A'(h_1) = 539 \] nodes
\[ A'(h_2) = 113 \] nodes
\[ d = 24 \] IDS = 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

Dominance 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

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