Informed search algorithms

Chapter 4

Outline

♦ Best-first search
♦ A* search
♦ Heuristics
♦ Hill-climbing
♦ Simulated annealing
**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

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

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>City</th>
<th>Distance to Bucharest</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>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hargava</td>
<td>151</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>Oraadea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy search example

Arad
366

Sibiu
253

Timisoara
329

Zerind
374
Greedy search example
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? $O(b^m)$—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

\[ 366 = 0 + 366 \]

Sibiu
393 = 140+253

Timisoara
447 = 118+329

Zerind
449 = 75+374
**A* search example**

![A* search example diagram]

19

**A* search example**

![A* search example diagram]

20
\(\text{\textcopyright search example}\)

\[
\begin{array}{c}
\text{Arad} \\
\text{Sibiu} \\
\text{Fagaras} \\
\text{Oradea} \\
\text{Rimnicu Vilcea} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Arad} \\
646=280+366 \\
\text{Sibiu} \\
591=338+253 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Fagaras} \\
\text{Oradea} \\
\text{Rimnicu Vilcea} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Arad} \\
671=291+380 \\
\text{Sibiu} \\
526=366+160 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Sibiu} \\
\text{Bucharest} \\
\text{Craiova} \\
\text{Pitesi} \\
\text{Sibiu} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Bucharest} \\
\text{Craiova} \\
\text{Pitesi} \\
\text{Sibiu} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Bucharest} \\
\text{Craiova} \\
\text{Rimnicu Vilcea} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Arad} \\
447=118+329 \\
\text{Sibiu} \\
553=300+253 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Fagaras} \\
\text{Oradea} \\
\text{Rimnicu Vilcea} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Arad} \\
646=280+366 \\
\text{Sibiu} \\
591=338+253 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Fagaras} \\
\text{Oradea} \\
\text{Rimnicu Vilcea} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Arad} \\
671=291+380 \\
\text{Sibiu} \\
526=366+160 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Sibiu} \\
\text{Bucharest} \\
\text{Craiova} \\
\text{Pitesi} \\
\text{Sibiu} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Bucharest} \\
\text{Craiova} \\
\text{Pitesi} \\
\text{Sibiu} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Arad} \\
447=118+329 \\
\text{Sibiu} \\
553=300+253 \\
\end{array}
\]
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.

---

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?? 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**? 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

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

I.e., $f(n)$ is nondecreasing along any path.
Admissible heuristics

E.g., for the 8-puzzle:

\( h_1(n) \) = number of misplaced tiles
\( h_2(n) \) = total Manhattan distance

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

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

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

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

\( h_1(S) = \)?? 7
\( h_2(S) = \)?? 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:

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

Relaxed problems

Admissible heuristics can be derived from the \textit{exact} solution cost of a \textit{relaxed} version of the problem.

If the rules of the 8-puzzle are relaxed so that a tile can move \textit{anywhere}, then \( h_1(n) \) gives the shortest solution.

If the rules are relaxed so that a tile can move to \textit{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

Iterative improvement algorithms

In many optimization problems, *path* is irrelevant;
the goal state itself is the solution

Then state space = set of “complete” configurations;
find *optimal* configuration, e.g., TSP
or, find configuration satisfying constraints, e.g., timetable

In such cases, can use *iterative improvement* algorithms;
keep a single “current” state, try to improve it

Constant space, suitable for online as well as offline search
Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

Example: $n$-queens

Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Move a queen to reduce number of conflicts
Hill-climbing (or gradient ascent/descent)

“Like climbing Everest in thick fog with amnesia”

function \textsc{hill-climbing}(\textit{problem}) \textbf{returns} a state that is a local maximum
\hspace{1em} inputs: \textit{problem}, a problem
\hspace{1em} local variables: \textit{current}, a node
\hspace{1.5em} \textit{neighbor}, a node

\hspace{1em} \textit{current} \leftarrow \textsc{make-node}([\textsc{initial-state}[\textit{problem}]])
\hspace{1em} \textbf{loop} do
\hspace{1.5em} \textit{neighbor} \leftarrow \text{a highest-valued successor of} \textit{current}
\hspace{1.5em} \textbf{if} \textsc{value}[\textit{neighbor}] < \textsc{value}[\textit{current}] \textbf{then return} \textsc{state}[\textit{current}]
\hspace{1.5em} \textit{current} \leftarrow \textit{neighbor}
\hspace{1em} \textbf{end}

Hill-climbing contd.

Problem: depending on initial state, can get stuck on local maxima

In continuous spaces, problems w/ choosing step size, slow convergence
Simulated annealing

Idea: escape local maxima by allowing some “bad” moves but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
inputs: problem, a problem
        schedule, a mapping from time to “temperature”
local variables: current, a node
               next, a node
               T, a “temperature” controlling prob. of downward steps

current ← MAKE-NODE(INITIAL-STATE[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] − VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability e^ΔE/T
```

Properties of simulated annealing

At fixed “temperature” T, state occupation probability reaches Boltzmann distribution

\[ p(x) = \frac{e^{-\frac{E(x)}{kT}}}{\sum_x e^{-\frac{E(x)}{kT}}} \]

T decreased slowly enough \(\Rightarrow\) always reach best state

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.