Informed search algorithms

Chapter 4, Sections 1–2, 4

Outline

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

Review: General search

```plaintext
function GENERAL-SEARCH(problem) returns a solution, or failure
    node = MAKE-NODE(INITIAL-STATE[problem])
    repeat
        if node is failure then return failure
        if GOAL-TEST[node] applied to STATE(node) succeeds then return node
        node = EXPAND(node, OPERATORS[problem])
    end
```

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:
- QUEUE = keep successors in decreasing order of desirability
- Special cases:
  - greedy search
  - A* search

Greedy search

Evaluation function \( h(n) \) (heuristic)
- estimate of cost from \( n \) to goal

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

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

Romania with step costs in km

Straight-line distance

- to Bucharest:
  - Arad: 566
  - Bucharest: 0
  - Craiova: 160
  - Dobruja: 242
  - Eforie: 161
  - Fagaras: 178
  - Giurgiu: 77
  - Iasi: 151
  - Iasi: 226
  - Lugoj: 244
  - Neamt: 253
  - Oradea: 380
  - Piatra: 98
  - Rimnicu Vilcan: 193
  - Sibiu: 253
  - Timisoara: 129
  - Urziceni: 90
  - Vaslui: 199
  - Zerind: 314
Properties of greedy search

Complete?? No can get stuck in loops, e.g.,
  lasi → Neamt → lasi → Neamt →
Complete in finite space with repeated-state checking

Time?? $O(l^n)$, but a good heuristic can give dramatic improvement

Space?? $O(l^n)$—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 $n$.

E.g., $h_{SHE}(n)$ 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 $G_1$.

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

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

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

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 $f_i < f_{i+1}$

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

Proof of lemma: Pathmax

For some admissible heuristics, $f$ may *decrease* along a path

E.g., suppose $n'$ is a successor of $n$

- $n: g=5 \quad h=4 \quad f=9$
- $n': g'=6 \quad h'=2 \quad f'=8$

But this throws away information!

$$f(n) = 9 \Rightarrow \text{true cost of a path through } n \geq 9$$

Hence true cost of a path through $n'$ is $\geq 9$ also

Pathmax modification to A*:

Instead of $f(n') = g(n') + h(n')$, use $f(n') = \max(g(n') + h(n'), f(n))$

With pathmax, $f$ is always 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)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

$\text{Start \ State}$ \hspace{1cm} $\text{Goal \ State}$

$$h_1(S) = ??$$

$$h_2(S) = ??$$
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)

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

- $h_1(S) = ?? 7$
- $h_2(S) = ?? 2+3+3+2+4+2+0+2 = 18$

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
- $d = 14$ IDS = too many nodes
  - $A^*(h_1) = 39,135$ nodes
  - $A^*(h_2) = 1,641$ nodes

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.
- For TSP: let path be any structure that connects all cities
  $\Rightarrow$ minimum spanning tree heuristic

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., n-queens
- 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

Find the shortest tour that visits each city exactly once

Example: n-queens

Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
**Hill-climbing (or gradient ascent/descent)**

"Like climbing Everest in thick fog with amnesia"

```
function HILL-CLIMBING (problem) returns a solution state
    inputs: problem, a problem
    local variables: current, a node

    current = MAKE-NODE (INITIAL-STATE [problem])
    while true do
        next = a highest-valued successor of current
        if VALUE [next] < VALUE [current] then return current
        current = next
    end
```

**Hill-climbing contd.**

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

```
value
```

```
    global maximum

    local maximum
```

**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

    T = initial "temperature" controlling the probability of distant steps

    current = MAKE-NODE (INITIAL-STATE [problem])
    for t = 1 to T do
        T = schedule[t]        
        if T = 0 then return current
        next = a randomly selected successor of current
        if E [next] < E [current] then current = next
        else current = next with probability e^\(\Delta E / T\)
    end
```

**Properties of simulated annealing**

At fixed "temperature" T, state occupation probability reaches

Boltzmann distribution

\[ p(x) = \frac{e^{-\frac{E}{T}}}{Z} \]

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.