Informed search algorithms

CHAPTER 4, SECTIONS 1–2, 4

Outline

◊ Best-first search
◊ A* search
◊ Heuristics
◊ Hill-climbing
◊ Simulated annealing
Review: General search

function General-Search(problem, QUEUING-FN) returns a solution, or failure
    nodes ← Make-Queue(Make-Node(Initial-State(problem)))
    loop
        if nodes is empty then return failure
        node ← REMOVE-FRONT(nodes)
        if GOAL-TEST(problem) applied to State(node) succeeds then return node
        nodes ← QUEUING-FN(nodes, EXPAND(node, Operators(problem)))
    end loop

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:
QUEUING-FN = insert 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., $h_{SLD}(n) =$ straight-line distance from $n$ to Bucharest

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

Properties of greedy search

- Complete??
- Time??
- Space??
- Optimal??
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?? \( O(b^n) \), but a good heuristic can give dramatic improvement

Space?? \( O(b^n) \)—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., \( h_{\text{SLD}}(n) \) 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 $G_1$.

\[
f(G_2) = g(G_2) + h(G_2) = g(G_2) + 0 = g(G_2) \geq f(n)
\]

since $h(G_2) = 0$

$g(G_2)$ since $G_2$ is suboptimal

$\geq f(n)$ since $h$ 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 \( n \):

\[
\begin{array}{ccc}
\text{ } & g=5 & h=4 & f=9 \\
1 \quad \text{ } & \text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } & \text{ } \\
n' \quad g'=6 & h'=2 & f'=8 \\
\end{array}
\]

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)

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

\( h_1(S) = ?? \)

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

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

\[ h_1(S) = ?? 7 \]
\[ h_2(S) = ?? 2+3+3+2+4+2+0+2 = 18 \]

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:

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

\[ d = 14 \quad \text{IDS} = \text{too many nodes} \]
\[ A^*(h_1) = 39,135 \text{ nodes} \]
\[ A^*(h_2) = 1,641 \text{ nodes} \]
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 $\implies$ 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
next, a node

current ← MAKE-NODE(INITIAL-STATE[problem])
loop 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
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
input: problem, a problem
schedule, a mapping from time to “temperature”
local variables: current, a node
next, a node
T, a “temperature” controlling the probability 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

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

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.