Informed search algorithms

Chapter 4

Outline

♦ Best-first search
♦ A* search
♦ Heuristics
♦ Hill-climbing
♦ Simulated annealing
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 ← INSERT-ALL(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**

Z. Duric

CS 480

Greedy search

Evaluation function \( h(n) \) (heuristic)

\[ h(n) = \text{estimate of cost from } n \text{ to the closest goal} \]

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
Greedy search example

Arad

Z. Duric

GMU
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

Starting from Arad, the algorithm expands nodes based on the heuristic and the cost to reach the node. The example shows the path to Sibiu with a total cost of 393 = 140 + 253. The path to Timisoara has a total cost of 447 = 118 + 329, and the path to Zerind has a total cost of 449 = 75 + 374. The algorithm continues to explore the graph, always choosing the path with the lowest cost.
A* search example

Z. Duric

GMU

CS 480

20
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 $f_i < f_{i+1}$
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??

Optimal??

Z. Duric

GMU
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:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]

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

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

---

Start State

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

Goal State

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

Z. Duric  
GMU

---

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)

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

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

Z. Duric  
GMU
**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:

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

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

**Relaxed problems**

Admissible heuristics can be derived from the *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 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: \textit{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 HILL-CLIMBING(problem) returns a state that is a local maximum

inputs: problem, a problem

local variables: current, a node
    neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])

loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] < VALUE[current] then return STATE[current]
    current ← neighbor
end

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

```plaintext
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
    ∆E ← VALUE[next] – VALUE[current]
    if ∆E > 0 then current ← next
    else current ← next only with probability e^{∆ E/T}
```

Properties of simulated annealing

At fixed “temperature” \( T \), state occupation probability reaches Boltzman distribution

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

\( T \) decreased slowly enough \( \implies \) 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.