INFORMED SEARCH ALGORITHMS

CHAPTER 4, SECTIONS 1–3 (THROUGH GAs)
Outline

♦ Best-first search
♦ A* search
♦ Heuristics
♦ Hill-climbing
♦ Simulated annealing
♦ Local (and stochastic) beam search
♦ Genetic algorithms
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)
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

<table>
<thead>
<tr>
<th>City</th>
<th>Step Costs in km</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Neamt</td>
<td>366</td>
</tr>
<tr>
<td>Vaslui</td>
<td>90</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>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</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>Timișoara</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>

Straight-line distance to Bucharest
- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobroța: 242
- Eforie: 161
- Făgăraș: 178
- Giurgiul: 77
- Hîrsova: 151
- Iasi: 226
- Lugoj: 244
- Mehadia: 241
- Neamț: 234
- Oradea: 380
- Pitesti: 98
- Rimnicu Vilcea: 193
- Sibiu: 253
- Timișoara: 329
- Urziceni: 80
- Vaslui: 199
- Zerind: 374
Greedy search

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

Arad
366
Greedy search example

- Sibiu: 253
- Timisoara: 329
- Zerind: 374
Greedy search example

Chapter 4, Sections 1–3 (through GAs)
Greedy search example

Chapter 4, Sections 1–3 (through GAs) 10
Properties of greedy search

Complete??
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??**
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 $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
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 \textit{admissible} heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the \textit{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

\textbf{Theorem:} A* search is optimal
A* search example

Arad
366=0+366
A* search example

![A* search example diagram]

- **Arad**
  - **Sibiu**: 393 = 140 + 253
  - **Timisoara**: 447 = 118 + 329
  - **Zerind**: 449 = 75 + 374
A* search example

Chapter 4, Sections 1–3 (through GAs)
**A* search example**

- **Cities:** Arad, Sibiu, Timisoara, Zerind, Fagaras, Oradea, Rimnicu Vilcea, Craiova, Pitesti, Sibiu

- **Distances:**
  - Arad to Fagaras: 646 = 280 + 366
  - Arad to Oradea: 415 = 239 + 176
  - Arad to Rimnicu Vilcea: 671 = 291 + 380
  - Sibiu to Craiova: 526 = 366 + 160
  - Sibiu to Pitesti: 415 = 317 + 100
  - Sibiu to Timisoara: 553 = 300 + 253
  - Timisoara to Zerind: 447 = 118 + 329
  - Zerind to Craiova: 449 = 75 + 374

- **Equations:**
  - 447 = 118 + 329
  - 449 = 75 + 374
  - 646 = 280 + 366
  - 415 = 239 + 176
  - 671 = 291 + 380
  - 526 = 366 + 160
  - 417 = 317 + 100
  - 553 = 300 + 253
A* search example

Chapter 4, Sections 1–3 (through GAs)   22
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.
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??
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??**
Properties of A*

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

**Time** Exponential in \([\text{relative error in } h \times \text{length of soln.}]\)

**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 $[\text{relative error in } h \times \text{length of soln.}]$

**Space??** Keeps all nodes in memory

**Optimal??** Yes—cannot expand $f_{i+1}$ until $f_i$ is finished

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 \textit{consistent} if

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

If \( h \) is consistent, we have

\[
\begin{align*}
    f(n') &= g(n') + h(n') \\
    &= g(n) + c(n, a, n') + h(n') \\
    &\geq g(n) + h(n) \\
    &= f(n)
\end{align*}
\]

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)

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

Start State \hspace{2cm} 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)

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

\[
\begin{align*}
h_1(S) &= ?? \quad 8 \\
h_2(S) &= ?? \quad 3+1+2+2+2+3+3+2 = 18
\end{align*}
\]
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:

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

```plaintext
function SIMULATED-AAnnealing(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 $e^{ΔE/T}$
```

Chapter 4, Sections 1–3 (through GAs) 41
Properties of simulated annealing

At fixed “temperature” $T$, state occupation probability reaches Boltzmann 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.
Local beam search

Keeps track of $k$ states, rather than just one.

Start with $k$ randomly generated states.

At each step, all successors of all $k$ states are generated.

If any is the goal, halt.

Otherwise, select $k$ best successors from the complete list.

NOTE: Useful information is shared among the $k$ parallel threads

Variant: *Stochastic beam search* – choose $k$ successors at random, with probability of choosing a successor being an increasing function of its value
Genetic algorithms (GA)

A GA is a variant of stochastic beam search

Successors are generated by combining 2 parent states

Begin with population of \( k \) randomly generated states

Rate individuals in population based on \textit{fitness function}

Choose two members (i.e., \textit{parents}) for \textit{reproduction}, based on fitness function

\textit{Crossover point} randomly chose between 2 parents and swap

\textit{Mutate} at randomly chosen points of individuals, with small probability

Create new population based on some function of fitness

Repeat for some number of \textit{generations}
Thought Discussion for next time

◇ (Read pages 947-949 of our text)
◇ “Weak AI: Can machines act intelligently?”

◇ Specifically: Consider argument from disability
  i.e., “A machine can never do X”