INFORMED SEARCH ALGORITHMS

CHAPTER 4
Outline

◊ Best-first search
◊ A* search
◊ Heuristics
◊ Hill-climbing
◊ Simulated annealing
◊ Genetic algorithms
◊ Local search in continuous spaces
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

Straight-line distance to Bucharest:
- Arad: 366 km
- Bucharest: 0 km
- Craiova: 160 km
- Dobrogea: 242 km
- Eforie: 161 km
- Fagaras: 178 km
- Giurgiu: 77 km
- Hirsova: 151 km
- Iasi: 226 km
- Lugoj: 244 km
- Mehadia: 241 km
- Neamt: 234 km
- Oradea: 380 km
- Pitesti: 98 km
- Rimnicu Vilcea: 193 km
- Sibiu: 253 km
- Timisoara: 329 km
- Urziceni: 80 km
- Vaslui: 199 km
- Zerind: 374 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 \textit{appears} to be closest to goal
Greedy search example

Arad
366
Greedy search example

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

Diagram showing a network of cities with distances between them.
Properties of greedy search

Complete??
Properties of greedy search

**Complete??** No can get stuck in loops, e.g., with Oradea as goal,

<table>
<thead>
<tr>
<th>lasi</th>
<th>Neamt</th>
<th>lasi</th>
<th>Neamt</th>
</tr>
</thead>
<tbody>
<tr>
<td>lasi</td>
<td>Neamt</td>
<td>lasi</td>
<td>Neamt</td>
</tr>
</tbody>
</table>

Complete in finite space with repeated-state checking

**Time??**
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**?? \( O(b^m) \), but a good heuristic can give dramatic improvement

**Space**??
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??  $O(b^m)$, 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.,
   \[ \text{ls} \rightarrow \text{Neamt} \rightarrow \text{ls} \rightarrow \text{Neamt} \rightarrow \]
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

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

Arad

Fagaras

Oradea

Rimnicu Vilcea

Craiova

Pitesti

Sibiu

Sibiu

Arad

646=280+366
415=239+176
671=291+380

Timisoara

447=118+329

Zerind

449=75+374

671=291+380
526=366+160
417=317+100
553=300+253

646=280+366
447=118+329
449=75+374
A* search example

- Arad
  - Fagaras
    - Sibiu: 646=280+366
    - Bucharest: 450=450+0
  - Oradea: 671=291+380
  - Rimnicu Vilcea
    - Craiova: 526=366+160
    - Pitesti: 417=317+100
    - Sibiu: 553=300+253
- Timisoara: 447=118+329
- Zerind: 449=75+374
A* search example

![A* search graph]

- **Arad**
  - **Sibiu**
    - **Fagaras**
    - **Oradea**
    - **Rimnicu Vilcea**
      - **Bucharest**
      - **Craiova**
      - **Pitesti**
        - **Bucharest**
        - **Craiova**
        - **Rimnicu Vilcea**

- **Timisoara**
  - **Zerind**

Costs:
- Arad to Sibiu: 646 = 280 + 366
- Sibiu to Fagaras: 671 = 291 + 380
- Sibiu to Oradea: 591 = 338 + 253
- Sibiu to Rimnicu Vilcea: 526 = 366 + 160
- Bucharest to Craiova: 418 = 418 + 0
- Craiova to Pitesti: 615 = 455 + 160
- Pitesti to Sibiu: 553 = 300 + 253
- Bucharest to Fagaras: 447 = 118 + 329
- Bucharest to Zerind: 449 = 75 + 374
- Bucharest to Sibiu: 591 = 338 + 253
- Sibiu to Bucharest: 526 = 366 + 160
- Craiova to Pitesti: 615 = 455 + 160
- Pitesti to Bucharest: 526 = 366 + 160
- Sibiu to Bucharest: 591 = 338 + 253
- Bucharest to Timisoara: 646 = 280 + 366
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 $G$.

\begin{align*}
  f(G_2) &= g(G_2) \quad \text{since } h(G_2) = 0 \\
  &> g(G) \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??
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 [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 \( [\text{relative error in } h \times \text{length of soln.}] \)

**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

Goal State

\[
\begin{align*}
h_1(S) &= ?? \\
h_2(S) &= ??
\end{align*}
\]
Admissible heuristics

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]
\[ \text{(i.e., no. of squares from desired location of each tile)} \]

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

\[
h_1(S) = ?? \quad 6 \\
h_2(S) = ?? \quad 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14
\]
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:

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

$d = 24 \quad$ IDS $\approx$ 54,000,000,000 nodes
  \quad A^*(h_1) = 39,135 \text{ nodes}
  \quad 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.

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

Variants of this approach get within 1% of optimal very quickly with thousands of cities
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.

Almost always solves \(n\)-queens problems almost instantaneously for very large \(n\).
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*

```
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 $e^{\Delta E / T}$
```
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

Idea: keep $k$ states instead of 1; choose top $k$ of all their successors

Not the same as $k$ searches run in parallel!
Searches that find good states recruit other searches to join them

Problem: quite often, all $k$ states end up on same local hill

Idea: choose $k$ successors randomly, biased towards good ones

Observe the close analogy to natural selection!
Genetic algorithms

Search and optimization methods inspired by natural selection

Population based stochastic search + generate successors from **pairs** of states

```
24748552  24  31%  32752411  32748552  32748152  24752411
32752411  23  29%  24748552  24752411  24752411
24415124  20  26%  32752411  32752124  32252124
32543213  11  14%  24415124  24415411  24415417
```

**Fitness**  **Selection**  **Pairs**  **Cross-Over**  **Mutation**
Genetic algorithms contd.

GAs require states encoded as strings (GPs use trees representing programs)

Crossover helps when substrings are meaningful components (decomposable problems)

GAs and GPs are examples of Evolutionary Computation methods
Continuous state spaces

Suppose we want to site three airports in Romania:
- 6-D state space defined by \((x_1, y_2), (x_2, y_2), (x_3, y_3)\)
- objective function \(f(x_1, y_2, x_2, y_2, x_3, y_3) = \text{sum of squared distances from each city to nearest airport}\)

Discretization methods turn continuous space into discrete space, e.g., empirical gradient considers \(\pm \delta\) change in each coordinate

Gradient methods compute

\[
\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)
\]

to increase/reduce \(f\), e.g., by \(x \leftarrow x + \alpha \nabla f(x)\)

Sometimes can solve for \(\nabla f(x) = 0\) exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates \(x \leftarrow x - H_f^{-1}(x) \nabla f(x)\)
to solve \(\nabla f(x) = 0\), where \(H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}\)