Heuristic

From Wikipedia:

any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal or perfect but sufficient for the immediate goals
Outline

- Best-first search
- A* search

- Heuristic algorithms
  - hill-climbing
  - simulated annealing
  - genetic algorithms (briefly)
  - local search in continuous spaces (very briefly)
best-first search
Review: Tree Search

function $\text{TREE-SEARCH}(\text{problem}, \text{fringe})$ returns a solution, or failure

$\text{fringe} \leftarrow \text{INSERT}(\text{MAKE-NODE}(\text{INITIAL-STATE}[\text{problem}]), \text{fringe})$

loop do
  if $\text{fringe}$ is empty then return failure
  $\text{node} \leftarrow \text{REMOVE-FRONT}(\text{fringe})$
  if $\text{GOAL-TEST}[\text{problem}]$ applied to $\text{STATE}(\text{node})$ succeeds return $\text{node}$
  $\text{fringe} \leftarrow \text{INSERT-ALL}(\text{EXPAND}(\text{node}, \text{problem}), \text{fringe})$

• Search space is in form of a tree

• 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

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobrogea</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</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>Pitești</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vălcea</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>
Greedy Search

- State 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
Greedy Search Example

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>166</td>
</tr>
<tr>
<td>Dobrogea</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehedia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</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>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>190</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy Search Example

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>166</td>
</tr>
<tr>
<td>Dobroța</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fugaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hârtova</td>
<td>151</td>
</tr>
<tr>
<td>Iași</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 Vilea</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>86</td>
</tr>
<tr>
<td>Vaslui</td>
<td>190</td>
</tr>
<tr>
<td>Zărand</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy Search Example

Straight-line distance to Bucharest:
- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobrota: 242
- Eforie: 161
- Fagaras: 178
- Giurgiu: 77
- Hirsova: 151
- Iasi: 226
- Lugoj: 244
- Mehadia: 341
- Neamt: 234
- Oradea: 380
- Pitesti: 98
- Rimnicu Vlclea: 193
- Sibiu: 253
- Timisoara: 329
- Urziceni: 80
- Vaslui: 190
- Zerind: 374
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?** $O(b^m)$, but a good heuristic can give dramatic improvement

- **Space?** $O(b^m)$ — keeps all nodes in memory

- **Optimal?** No
a* search
**A* Search**

- **Idea:** avoid expanding paths that are already expensive

- **State 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

Straight-line distance to Bucharest:

- Arad: 0
- Bucharest: 0
- Cluj-Napoca: 180
- Debrecen: 202
- Eforie: 161
- Fagaras: 178
- Giurgiu: 72
- Hîncești: 151
- Iași: 226
- Lași: 264
- Miercurea Ciuc: 261
- Neamț: 234
- Oradea: 380
- Pitești: 98
- Rimnicu Vîlcea: 193
- Sibiu: 253
- Timișoara: 318
- Vâlcea: 86
- Vâlcea: 199
- Zerind: 374
A* Search Example
A* Search Example
A* Search Example
A* Search Example

Straight-line distance vs Bucharest:

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>166</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobrota</td>
<td>202</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>70</td>
</tr>
<tr>
<td>Hirssova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lapaic</td>
<td>264</td>
</tr>
<tr>
<td>Mieadadia</td>
<td>361</td>
</tr>
<tr>
<td>Neamt</td>
<td>354</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitești</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>153</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>380</td>
</tr>
<tr>
<td>Uracensi</td>
<td>86</td>
</tr>
<tr>
<td>Vâlcea</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

Arad
Fagaras
Oradea
Rimnicu Vilcea
Craiova
Pitești
Sibiu

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

Fagaras
646 = 366 + 160
417 = 317 + 100
553 = 300 + 253
A* Search Example
A* Search Example

Straight-Line Distance to Bucharest

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>166</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>166</td>
</tr>
<tr>
<td>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>79</td>
</tr>
<tr>
<td>Hirssova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lapej</td>
<td>264</td>
</tr>
<tr>
<td>Media</td>
<td>301</td>
</tr>
<tr>
<td>Neamt</td>
<td>334</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vlaicu</td>
<td>153</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>378</td>
</tr>
<tr>
<td>Vahal</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

Grafo de búsqueda A*
A* Search Example
A* Search Example
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 terminate at $G_2$
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 solution]

- **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

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

- I.e., \( f(n) \) is nondecreasing along any path.
Admissible Heuristics

- E.g., for the 8-puzzle
  - $h_1(n) =$ number of misplaced tiles
  - $h_2(n) =$ total Manhattan distance
    (i.e., no. of squares from desired location of each tile)

- $h_1(S) =$?
- $h_2(S) =$?
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)

- $h_1(S) =$ 6
- $h_2(S) =$ 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14
Dominance

- If \( h_2(n) \geq h_1(n) \) for all \( n \) (both admissible)
  \[
  \rightarrow h_2 \text{ dominates } h_1 \text{ and is better for search}
  \]

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

- Given any admissible heuristics \( h_a, h_b \),
  \[
  h(n) = \max(h_a(n), h_b(n))
  \]

  is also admissible and dominates \( h_a, h_b \)
• 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 ⇒ $h_1(n)$ gives the shortest solution.

• If the rules are relaxed so that a tile can move to any adjacent square ⇒ $h_2(n)$ 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

- Well-known example: travelling salesperson problem (TSP)

- Find the shortest tour visiting all cities exactly once

- Minimum spanning tree
  - can be computed in $O(n^2)$
  - is a lower bound on the shortest (open) tour
Summary: A*

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest \( h \)
  - incomplete and not always optimal
- A* search expands lowest \( g + h \)
  - complete and optimal
  - also optimally efficient (up to tie-breaks, for forward search)
- Admissible heuristics can be derived from exact solution of relaxed problems
iterative improvement algorithms
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
  - 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 quickly with 1000s 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 \( n \), e.g., \( n = 1 \) million
Hill-Climbing

- For instance Gradient Ascent (or Descent)
- “Like climbing Everest in thick fog with amnesia”

```plaintext
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

- Useful to consider state space landscape

- Random-restart hill climbing overcomes local maxima—trivially complete
- Random sideways moves 😊 escape from shoulders 😊 loop on flat 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
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 $e^{ΔE/T}$
```

Philipp Koehn  Artificial Intelligence: Informed Search  24 September 2015
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 $\Rightarrow$ always reach best state $x^*$
  
  because $e^{\frac{E(x^*)}{kT}}/e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small $T$

- 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

- Stochastic local beam search + generate successors from **pairs** of states
Genetic Algorithms

- GAs require states encoded as strings (GPs use programs)
- Crossover helps iff substrings are meaningful components
- GAs ≠ evolution: e.g., real genes encode replication machinery!
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 \(f(x_1, y_2, x_2, y_2, x_3, y_3) = \) 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 \(H_{ij} = \partial^2 f / \partial x_i \partial x_j\)
Summary

- Exact search
  - exhaustive exploration of the search space
  - search with heuristics: a*

- Approximate search
  - hill-climbing
  - simulated annealing
  - genetic algorithms (briefly)
  - local search in continuous spaces (very briefly)