Informed Search

Philipp Koehn

21 February 2019
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

```plaintext
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)
```

- 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
Greedy Search

- State evaluation function $h(n)$ (heuristic) = estimate of cost from $n$ to the closest goal
- E.g., $h_{SLD}(n) = \text{straight-line distance from } n \text{ to Bucharest}$
- Greedy search expands the node that appears to be closest to goal
Romania with Step Costs in km
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>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>77</td>
</tr>
<tr>
<td>Hirssova</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>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimmnicu 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

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>Moldavia</td>
<td>341</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitești</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>Vâlcea</td>
<td>190</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy Search Example

Straight-line distance to Bucharest:
- Arad: 366
- Bucharest: 0
- Craiova: 166
- Dobrota: 242
- Eforie: 161
- Fagaras: 178
- Giurgiu: 77
- Hirsova: 151
- Iasi: 226
- Lugoj: 244
- Mehadia: 341
- Neamt: 234
- Oradea: 380
- Pitesti: 98
- Rimnicu Vilea: 193
- Sibiu: 253
- Timisoara: 329
- Urizenci: 80
- Vaslui: 190
- Zerind: 374
Greedy Search Example

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>Location</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>Făgăraș</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hîrsova</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>341</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>Râmnicu Vâlcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiù</td>
<td>253</td>
</tr>
<tr>
<td>Timișoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vâlcea</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
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
A* Search Example

[Diagram showing a graph with cities and distances]

Straight-Line distance to Bucharest

- Arad: 166
- Bucharest: 0
- Craiova: 166
- Dobrogea: 242
- Eforie: 161
- Fagaras: 78
- Giurgiu: 142
- Herta: 151
- Iasi: 226
- Lapug: 242
- Most: 391
- Neamț: 242
- Oradea: 380
- Pitești: 98
- Rimnicu Vilceu: 153
- Sibiu: 253
- Timisoara: 374
- Urziceni: 90
- Vâlcea: 199
- Zerind: 374

Arad

- Sibiu: 393 = 140 + 253
- Timisoara: 447 = 118 + 329
- Zerind: 449 = 75 + 374
A* Search Example
A* Search Example
A* Search Example
A* Search Example
A* Search Example

Straight-Line distance to Bucharest

- Arad: 166
- Bucharest: 0
- Craiova: 166
- Dobrogea: 242
- Eforsie: 161
- Fagaras: 178
- Galati: 70
- Hirssoa: 151
- Iasi: 226
- Lapoj: 264
- Vihavdia: 361
- Neamt: 374
- Oradea: 380
- Pitesti: 98
- Rimnicu Vilcea: 193
- Sibiu: 253
- Timisoara: 137
- Urakent: 89
- Vachai: 199
- Zerind: 374

Arad

Sibiu

Fagaras

Oradea

Rimnicu Vilcea

Sibiu

Bucharest

Craiova

Pitesti

447 = 118 + 329

449 = 75 + 374

646 = 280 + 366

671 = 291 + 380

591 = 338 + 253

450 = 450 + 0

526 = 366 + 160

417 = 317 + 100

553 = 300 + 253

Philipp Koehn

Artificial Intelligence: Informed Search

21 February 2019
A* Search Example

Straight-Line Distance to Bucharest

- Arad: 166
- Bucharest: 0
- Craiova: 160
- Dobrota: 242
- Eforie: 141
- Fagaras: 178
- Gorj: 79
- Hlinsu: 151
- Iasi: 226
- Lagoj: 264
- Mehadia: 361
- Neamt: 334
- Oradea: 380
- Pitesti: 58
- Rimnicu Vilcea: 153
- Sibiu: 253
- Timisoara: 370
- Urziceni: 86
- Vâlcea: 199
- Zerind: 374
A* Search Example
Optimality of A*

- 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 $G$.

\[
f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0
\]
\[
> g(G) \quad \text{since } G_2 \text{ is suboptimal}
\]
\[
\geq f(n) \quad \text{since } h \text{ is admissible}
\]

- Since $f(G_2) > f(n)$, A* will never terminate at $G_2$. 

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 $f(n) > C^*$
Admissible Heuristics

- E.g., for the 8-puzzle
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)

\[
\begin{align*}
\text{Start State} & \quad \text{Goal State} \\
7 & \quad 1 \\
2 & \quad 2 \\
4 & \quad 3 \\
5 & \quad 4 \\
6 & \quad 5 \\
8 & \quad 6 \\
3 & \quad 7 \\
1 & \quad 8 \\
\end{align*}
\]

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

- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible)
  $\rightarrow h_2$ dominates $h_1$ and is better for search

- Typical search costs ($d =$ depth of solution for 8-puzzle)

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

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

- Given any admissible heuristics $h_a, h_b$,

  \[ h(n) = \max(h_a(n), h_b(n)) \]

  is also admissible and dominates $h_a, h_b$
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, \( h_1(n) \) 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
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, \textit{path} is irrelevant; the goal state itself is the solution

- Then state space = set of “complete” configurations
  - find \textit{optimal} configuration, e.g., TSP
  - find configuration satisfying constraints, e.g., timetable

- In such cases, can use \textit{iterative improvement} algorithms
  \rightarrow 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”

1. Start state = a solution (maybe randomly generated)
2. Consider neighboring states, e.g.,
   - move a queen
   - pairwise exchange in traveling salesman problem
3. No better neighbors? Done.
4. Adopt best neighbor state
5. Go to step 2
Hill-Climbing

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 \text{SIMULATED-ANNEALING}(\text{problem}, \text{schedule})\ returns a solution state

inputs: \text{problem}, a problem

\text{schedule}, a mapping from time to “temperature”

local variables: \text{current}, a node

\text{next}, a node

\text{T}, a “temperature” controlling prob. of downward steps

\text{current} \leftarrow \text{MAKE-NODE}(\text{INITIAL-STATE}[\text{problem}])

for \( t \leftarrow 1 \) to \( \infty \) do

\text{T} \leftarrow \text{schedule}[t]

if \( T = 0 \) then return \text{current}

\text{next} \leftarrow \text{a randomly selected successor of \text{current}}

\Delta E \leftarrow \text{VALUE}[\text{next}] - \text{VALUE}[\text{current}]

if \( \Delta E > 0 \) then \text{current} \leftarrow \text{next}

else \text{current} \leftarrow \text{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 $x^*$
  
  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!**

- **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
Summary

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

- Approximate search
  - hill-climbing
  - simulated annealing
  - local beam search (briefly)
  - genetic algorithms (briefly)