Informed Search

Philipp Koehn

25 February 2020
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** 

\[ \text{function } \textsc{Tree-Search}(\text{problem}, \text{fringe}) \text{ returns } \text{a solution, or failure} \]

\[ \text{fringe} \leftarrow \text{INSERT} (\text{MAKE-Node(Initial-State[problem]}), \text{fringe}) \]

**Loop**

\[ \text{loop do} \]

\[ \text{if fringe is empty then return failure} \]

\[ \text{node} \leftarrow \text{REMOVE-FRONT}(\text{fringe}) \]

\[ \text{if GOAL-TEST[problem] applied to STATE(node) succeeds return node} \]

\[ \text{fringe} \leftarrow \text{INSERT ALL(EXPAND(node, problem), fringe)} \]

- 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
Romania
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
Romania with Step Costs in km

<table>
<thead>
<tr>
<th>Straight-line distance to Bucharest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
</tr>
<tr>
<td>Bucharest</td>
</tr>
<tr>
<td>Craiova</td>
</tr>
<tr>
<td>Dobroța</td>
</tr>
<tr>
<td>Eforie</td>
</tr>
<tr>
<td>Făgăraș</td>
</tr>
<tr>
<td>Giurin</td>
</tr>
<tr>
<td>Hârșova</td>
</tr>
<tr>
<td>Iași</td>
</tr>
<tr>
<td>Lugoj</td>
</tr>
<tr>
<td>Mehadia</td>
</tr>
<tr>
<td>Neamț</td>
</tr>
<tr>
<td>Oradea</td>
</tr>
<tr>
<td>Pitești</td>
</tr>
<tr>
<td>Rimnicu Vâlcea</td>
</tr>
<tr>
<td>Sibiu</td>
</tr>
<tr>
<td>Timișoara</td>
</tr>
<tr>
<td>Urziceni</td>
</tr>
<tr>
<td>Vaslui</td>
</tr>
<tr>
<td>Zerind</td>
</tr>
</tbody>
</table>
Greedy Search Example

Straight-line distance to Bucharest

- Arad: 366
- Bucharest: 0
- Craiova: 166
- Dobreta: 242
- Eforie: 161
- Fagaras: 178
- Giurgiu: 77
- Hîrsova: 151
- Iasi: 226
- Lugoj: 244
- Mehadia: 241
- Neamt: 234
- Oradea: 380
- Pitești: 98
- Rimnicu Vilcea: 193
- Sibiu: 253
- Timișoara: 329
- Urziceni: 86
- Vaslui: 190
- Zerind: 374

Diagram showing distances between various cities in Romania.
Greedy Search Example

Straight-line distance to Bucharest

- Arad: 366
- Bucharest: 0
- Craiova: 166
- Dobrota: 242
- Eforie: 161
- Fagaras: 178
- Giurgiu: 77
- Hîrsova: 151
- Iași: 226
- Lugoj: 244
- Medgidia: 241
- Neamț: 234
- Oradea: 380
- Pitești: 98
- Rimnicu Vâlcea: 193
- Sibiu: 253
- Timișoara: 329
- Urziceni: 86
- Vaslui: 190
- Zerind: 374
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 Vilcea: 193
- Sibiu: 253
- Timisoara: 329
- Urziceni: 86
- Vaslui: 190
- Zerind: 374
Greedy Search Example

Straight-line distance to Bucharest
- Arad: 366
- Bucharest: 0
- Craiova: 166
- Dobrogea: 242
- Eforie: 161
- Fagaras: 178
- Giurgiu: 77
- Hirsova: 151
- Iasi: 226
- Lugoj: 244
- Mehedia: 241
- Neamt: 234
- Oradea: 380
- Pitesti: 98
- Rimnicu Vilea: 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: 366
- Bucharest: 0
- Cluj: 106
- Dobrogea: 242
- Eforie: 161
- Fagaras: 178
- Giurgiu: 21
- Hirssova: 151
- Iasi: 226
- Lapoj: 264
- Mediasia: 351
- Neamt: 314
- Oradea: 380
- Piteii: 98
- Rimnicu Vilcea: 192
- Sibiu: 253
- Timisoara: 378
- Uzuncen: 86
- Vailai: 199
- Zerind: 374
A* Search Example

Straight-Line distance to Bucharest

<table>
<thead>
<tr>
<th>Location</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>Cradova</td>
<td>166</td>
</tr>
<tr>
<td>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Ploiești</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>70</td>
</tr>
<tr>
<td>Hîrsova</td>
<td>151</td>
</tr>
<tr>
<td>Iași</td>
<td>226</td>
</tr>
<tr>
<td>Lăpuș</td>
<td>264</td>
</tr>
<tr>
<td>Vâlcea</td>
<td>351</td>
</tr>
<tr>
<td>Neamț</td>
<td>734</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitești</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vîlcea</td>
<td>593</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timișoara</td>
<td>378</td>
</tr>
<tr>
<td>Urziceni</td>
<td>86</td>
</tr>
<tr>
<td>Vâlcea</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

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

Straight-Line Distance to Bucharest

- Arad: 166
- Bucharest: 0
- Cluj: 106
- Dobrogea: 242
- Eforie: 161
- Fagaras: 178
- Gorj: 79
- Iasi: 151
- Iasi: 236
- Lapoj: 234
- Miercurea: 361
- Neamt: 334
- Oradea: 380
- Pitești: 98
- Rimnicu Vâlcea: 253
- Sibiu: 253
- Timisoara: 358
- Urziceni: 86
- Vâlcea: 199
- Zerind: 374

<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>Cluj</td>
<td>106</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>Gorj</td>
<td>79</td>
</tr>
<tr>
<td>Iasi</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>236</td>
</tr>
<tr>
<td>Lapoj</td>
<td>234</td>
</tr>
<tr>
<td>Miercurea</td>
<td>361</td>
</tr>
<tr>
<td>Neamt</td>
<td>334</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitești</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vâlcea</td>
<td>253</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>358</td>
</tr>
<tr>
<td>Urziceni</td>
<td>86</td>
</tr>
<tr>
<td>Vâlcea</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
A* Search Example
A* Search Example
A* Search Example
A* Search Example

Straight-Line Distance to Bucharest:

- Arad: 166
- Bucharest: 0
- Craiova: 166
- Dobrota: 202
- Iasi: 161
- Fagaras: 178
- Giurgiu: 70
- Hirssoa: 125
- Iasi: 226
- Lapej: 264
- Vienadia: 361
- Neamt: 334
- Oradea: 380
- Pitesti: 98
- Rumania Vilcea: 193
- Sibiu: 253
- Timisoara: 370
- Uraceni: 80
- Vaih: 199
- Zerind: 374
A* Search Example
A* Search Example
A* Search Example

Straight-line distance to Bucharest

- Arad: 166
- Bucharest: 0
- Craiova: 180
- Dobroța: 242
- Eforie: 161
- Făgăraș: 178
- Giurgiu: 7
- Hârlău: 151
- Iași: 226
- Lapoș: 264
- Medgidia: 351
- Neamț: 334
- Oradea: 350
- Pitesti: 58
- Rimnicu Vâlcea: 193
- Sibiu: 253
- Timișoara: 338
- Uzanești: 80
- Vâlcea: 199
- Zerind: 374
A* Search Example

Straight-line distance to Bucharest:
- Arad: 166
- Bucharest: 0
- Craiova: 160
- Dobrota: 242
- Eforie: 161
- Fagaras: 178
- Galati: 79
- Hârșova: 151
- Iași: 226
- L spațial
- Vâlcea: 311
- Neamț: 334
- Oradea: 380
- Pitești: 58
- Rimnicu Vîlcea: 153
- Sibiu: 253
- Timișoara: 380
- Urziceni: 80
- Vâlcea: 199
- Zerind: 374
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 $G$.

\[
\begin{align*}
    f(G_2) & = g(G_2) & \text{since } h(G_2) = 0 \\
    & > g(G) & \text{since } G_2 \text{ is suboptimal} \\
    & \ge f(n) & \text{since } h \text{ is admissible}
\end{align*}
\]

- 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

\[ A^* \text{ expands all nodes with } f(n) < C^* \]
\[ A^* \text{ expands some nodes with } f(n) = C^* \]
\[ A^* \text{ expands no nodes with } f(n) > C^* \]
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)

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

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

- $h_1(S) = 6$
- $h_2(S) = 4+0+3+3+1+0+2+1 = 14$
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 ($d =$ depth of solution for 8-puzzle)
  \[
  \begin{align*}
  d = 14 & \quad \text{IDS} = 3,473,941 \text{ nodes} \\
  & \quad A^*(h_1) = 539 \text{ nodes} \\
  & \quad A^*(h_2) = 113 \text{ nodes} \\
  d = 24 & \quad \text{IDS} \approx 54,000,000,000 \text{ nodes} \\
  & \quad A^*(h_1) = 39,135 \text{ nodes} \\
  & \quad A^*(h_2) = 1,641 \text{ nodes} \\
  \end{align*}
  \]

- Given any admissible heuristics $h_a, h_b,$
  \[
  h(n) = \max(h_a(n), h_b(n))
  \]
  is also admissible and dominates $h_a, h_b$
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, *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* × *n* board with no two queens on the same row, column, or diagonal.

- Move a queen to reduce the 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

```latex
function SIMULATED-ANNEALING\( (\text{problem, schedule}) \) 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
                 \( T \), a “temperature” controlling prob. of downward steps

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

\text{for } t \leftarrow 1 \text{ to } \infty \text{ do}
   \quad T \leftarrow \text{schedule}[t]
   \quad \text{if } T = 0 \text{ then return } \text{current}
   \quad \text{next} \leftarrow \text{a randomly selected successor of } \text{current}
   \quad \triangle E \leftarrow \text{VALUE}[\text{next}] - \text{VALUE}[\text{current}]
   \quad \text{if } \triangle E > 0 \text{ then } \text{current} \leftarrow \text{next}
   \quad \text{else } \text{current} \leftarrow \text{next} \text{ only with probability } e^{\triangle 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 $x^*$
  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)