Informed search algorithms

(Based on slides by Oren Etzioni, Stuart Russell)
### The problem

<table>
<thead>
<tr>
<th># Unique board configurations in search space</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-puzzle</td>
</tr>
<tr>
<td>9! = 362880</td>
</tr>
<tr>
<td>15-puzzle</td>
</tr>
<tr>
<td>16! = 209227898888000 ≈ 10^{13}</td>
</tr>
<tr>
<td>24-puzzle</td>
</tr>
<tr>
<td>25! ≈ 10^{25}</td>
</tr>
<tr>
<td>35-puzzle</td>
</tr>
<tr>
<td>36! ≈ 10^{41}</td>
</tr>
<tr>
<td>48-puzzle</td>
</tr>
<tr>
<td>49! ≈ 10^{63}</td>
</tr>
<tr>
<td>63-puzzle</td>
</tr>
<tr>
<td>64! ≈ 10^{89}</td>
</tr>
</tbody>
</table>

- Number of atoms in known universe ≈ 10^{80}
Outline

- Greedy best-first search
- A* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms
Best-first search

• A search strategy is defined by picking the order of node expansion

• Idea: use an evaluation function $f(n)$ for each node
  – estimate of "desirability"

  → Expand most desirable unexpanded node

• Implementation:
  Order the nodes in fringe in decreasing order of desirability

• Special cases:
  – greedy best-first search
  – $A^*$ search
Romania with step costs in km

<table>
<thead>
<tr>
<th>City</th>
<th>Distance (km)</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>Dobreta</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>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>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy best-first search

• Evaluation function $f(n) = h(n)$ (heuristic) = estimate of cost from $n$ to goal

• e.g., $h_{SLD}(n)$ = straight-line distance from $n$ to Bucharest

• Greedy best-first search expands the node that appears to be closest to goal
Properties of greedy best-first search

- **Complete?**
- No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
- **Time?**
- $O(b^m)$, but a good heuristic can give dramatic improvement
- **Space?**
- $O(b^m)$ -- keeps all nodes in memory
- **Optimal?**
- No
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>Dobroţa</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>Neamţ</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rîmnicul Vîlcea</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>
A* search

• Idea: avoid expanding paths that are already expensive
• Evaluation function $f(n) = g(n) + h(n)$
  
  • $g(n) = \text{cost so far to reach } n$
  • $h(n) = \text{estimated cost from } n \text{ to goal}$
  • $f(n) = \text{estimated total cost of path through } n \text{ to goal}$
A* search example
A* search example

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

- A heuristic \( h(n) \) is admissible if for every node \( n \), \( h(n) \leq h^*(n) \), where \( h^*(n) \) is the true cost to reach the goal state from \( n \).

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.

- Example: \( h_{SLD}(n) \) (never overestimates the actual road distance)

- **Theorem**: If \( h(n) \) is admissible, A* using TREE-SEARCH is optimal.
Properties of A*

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

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

- **Space?** \( O(b^m) \), Keeps all nodes in memory

- **Optimal?**
  Yes
Why optimal? By contradiction

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.
A* is “optimally efficient”

• With an admissible heuristic,
  – A* expands all nodes with \( f(n) < C \)
  – A* expands some nodes with \( f(n) = C \)
  – A* expands no nodes with \( f(n) > C \)

• So, except for the variable (usually small) number of nodes with \( f(n) = C \),
  – No optimal algorithm using \( h \) expands fewer nodes than A*
Admissible heuristics

Start State

Goal State
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)

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

- $h_1(S) = ?$ 8
- $h_2(S) = ? 3+1+2+2+2+3+3+2 = 18$
Dominance

- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$

- $h_2$ is at least as good as $h_1$ for search, and likely better
  - Why?
<table>
<thead>
<tr>
<th>$d$</th>
<th>Search Cost (nodes)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IDS</td>
<td>$A^{*}(h1)$</td>
<td>$A^{*}(h2)$</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>6386</td>
<td>39</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>39</td>
</tr>
<tr>
<td>12</td>
<td>3644035</td>
<td>227</td>
<td>73</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>539</td>
<td>113</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>1301</td>
<td>211</td>
</tr>
<tr>
<td>18</td>
<td>-</td>
<td>3056</td>
<td>363</td>
</tr>
</tbody>
</table>
Summary

- **A* search**
  - Expand nodes in increasing order of:
    \[ f(n) = g(n) + h(n) \]
    \[ = \text{cost so far} + \text{estimated cost to goal} \]
  - Optimal for *admissible* heuristics
    - Admissible = “optimistic”
  - Designing heuristics is key for performance
    - More next time
Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem.

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original 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.
Traveling Salesman Problem

- Goal: find the least-cost cycle in the graph that visits each node exactly once
TSP Relaxed Problem Heuristic

- Relaxed problem: find least-cost tree that connects all nodes (minimum spanning tree).
  - \( \text{Cost}(\text{MST}) \leq \text{Cost}(\text{Best Tour} - 1 \text{ edge}) < \text{Cost}(\text{Best Tour}) \)
Combining Heuristics

• Say we have two heuristics, $h_1$ and $h_2$, and neither dominates the other.
  – What can we do?

• $h_3(n) = \max(h_1(n), h_2(n))$
  – $h_3$ dominates $h_1$, $h_2$
Pattern Databases

- $h(n) =$ cost to get $\{1,2,3,4\}$ in right place
  - Compute once for all possible configurations and store
- Can use multiple sub-problems (e.g., $\{5,6,7,8\}$) and combine with max
  - Or, ignore * moves and add disjoint subproblems
Summary of A* Search

- Expands node n with minimum \( f(n) = g(n) + h(n) \)
  \[ = \text{path cost so far} + \text{heuristic estimate} \]

- Optimal for *admissible* heuristic \( h(n) \)
  - I.e. \( h \) that underestimates true path cost

- Designing good heuristics is crucial for performance
  - One method: Relaxed problems

- Combining heuristics
  - Take max or add “disjoint” heuristics
Outline

• Greedy best-first search
• A* search
• Heuristics
• Local search algorithms
• Hill-climbing search
• Simulated annealing search
• Local beam search
• Genetic algorithms
Local search algorithms

• In many optimization problems, the path to the goal is irrelevant
  – the goal state itself is the solution

• State space = set of "complete" configurations
• Find configuration satisfying constraints, e.g., n-queens

• In such cases, we can use local search algorithms
• keep a single "current" state, try to improve it
Example: $n$-queens

• Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
Hill-climbing search

• "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
Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima
Hill-climbing search: 8-queens problem

- $h =$ number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state
Hill-climbing search: 8-queens problem

• A local minimum with $h = 1$
Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but **gradually decrease** their 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 $e^{ΔE/T}$
```
Properties of simulated annealing search

• One can prove: If $T$ decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1.

• Widely used in VLSI layout, airline scheduling, etc.
Local beam search

• Keep track of $k$ states rather than just one
• Start with $k$ randomly generated states
• At each iteration, all the successors of all $k$ states are generated
• If any one is a goal state, stop; else select the $k$ best successors from the complete list and repeat.
Genetic algorithms

- A successor state is generated by combining two parent states
- Start with $k$ randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation
Genetic algorithms

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
  - 24/(24+23+20+11) = 31%
  - 23/(24+23+20+11) = 29% etc
Genetic algorithms

• Genetic algorithm is “stochastic beam search”
  – Key difference: combine multiple parents

For which problems is this helpful?
Continuous Optimization

• Many AI problems require optimizing a function $f(x)$, which takes continuous values for input vector $x$

• Huge research area

• Examples:
  – Machine Learning
  – Signal/Image Processing
  – Computational biology
  – Finance
  – Weather forecasting
  – Etc., etc.
Gradient Ascent

- Idea: move in direction of steepest ascent (gradient)

- \( \mathbf{x}_k = \mathbf{x}_{k-1} + \eta \nabla f(\mathbf{x}_{k-1}) \)
Types of Optimization

• Linear vs. non-linear

• Analytic vs. Empirical Gradient

• Convex vs. non-convex

• Constrained vs. unconstrained
Continuous Optimization in Practice

• *Lots* of previous work on this

• Use packages