Informed Search and Beyond

Chapters 3 (3.5-3.7), 4 (4.1)
Introduction

- Uninformed searches – good building blocks for learning about search
- But vastly inefficient
- Can we do better?
Previous algorithms differed in how to select next node for expansion eg:

- Breadth First
  - Fringe nodes sorted old -> new

- Depth First
  - Fringe nodes sorted new -> old

- Uniform cost
  - Fringe nodes sorted by path cost: small -> big

- Used little (no) “external” domain knowledge
Overview

- Heuristic Search
  - Best-First Search Approach
    - Greedy
    - A*
  - Heuristic Functions

- Local Search and Optimization
  - Hill-climbing
  - Simulated Annealing
Informed Searching

- An *informed search* strategy uses knowledge beyond the definition of the problem.
- The knowledge is embodied in an *evaluation function* $f(n)$. 

Best-First Search

- An algorithm in which a node is selected for expansion based on an evaluation function \( f(n) \)
  - Fringe nodes ordered by \( f(n) \)
  - Traditionally the node with the lowest evaluation function is selected
  - Not an accurate name...expanding the best node first would be a straight march to the goal.
  - Choose the node that *appears* to be the best
Best-First Search

- Remember: Uniform cost search
  - $F(n) = g(n)$

- Best-first search:
  - $F(n) = h(n)$

- Later, a-star search:
  - $F(n) = g(n) + h(n)$
Best-First Search (cont.)

- Some BFS algorithms also include the notion of a heuristic function $h(n)$
  - $h(n) = \text{estimated cost of the cheapest path from node } n \text{ to a goal node}$
- Best way to include informed knowledge into a search
- Examples:
  - How far is it from point A to point B
  - How much time will it take to complete the rest of the task at current node to finish
Greedy Best-First Search

- Expands node estimated to be closest to the goal
  - \( f(n) = h(n) \)
- Consider the route finding problem.
  - Can we use additional information to avoid costly paths that lead nowhere?
  - Consider using the straight line distance (SLD)
Route Finding: Greedy Best First

Arad

\[ f(n) = 366 \]
Route Finding: Greedy Best First

Arad

Sibiu 253
Timisoara 329
Zerind 374

\( f(n) = 366 \)
Route Finding: Greedy Best First

f(n) = 366

Arad

Sibiu 253

Timisoara 329

Zerind 374

Arad 366

Fagaras 176

Oradea 380

Rimnicu Vilcea 193
Route Finding: Greedy Best First

f(n) = 366

Arad

Sibiu 253

Timisoara 329

Zerind 374

Arad 366

Fagaras 176

Oradea 380

Rimnicu Vilcea 193

Bucharest 0

Sibiu 253
So is Arad->Sibiu->Fagaras->Bucharest optimal?
Greedy Best-First Search

- Not optimal.
- Not complete.
  - Could go down a path and never return to try another.
  - e.g., Iasi → Neamt → Iasi → Neamt → ...

Space Complexity
- \( O(b^m) \) – keeps all nodes in memory

Time Complexity
- \( O(b^m) \) (but a good heuristic can give a dramatic improvement)
Heuristic Functions

- Example: 8-Puzzle
  - Average solution cost for a random puzzle is 22 moves
  - Branching factor is about 3
    - Empty tile in the middle -> four moves
    - Empty tile on the edge -> three moves
    - Empty tile in corner -> two moves
  - $3^{22}$ is approx $3.1 \times 10^{10}$
    - Get rid of repeated states
    - 181,440 distinct states
Heuristic Functions

- $h_1 = $ number of misplaced tiles
- $h_2 = $ sum of distances of tiles to goal position.
Heuristic Functions

- $h_1 = 7$
- $h_2 = 4+0+3+3+1+0+2+1 = 14$
Admissible Heuristics

- A heuristic function $h(n)$ is *admissible* if it never overestimates the cost to reach the goal from $n$.

- Is $h_1$ (number of displaced tiles) admissible?
  - Yes

- Is $h_2$ (Manhattan distance) admissible?
  - Yes
Dominance

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

- Typical search costs (average number of nodes expanded):
  - $d=12$  IDS = 3,644,035 nodes
    - $A^*(h_1) = 227$ nodes
    - $A^*(h_2) = 73$ nodes
  - $d=24$  IDS = too many nodes
    - $A^*(h_1) = 39,135$ nodes
    - $A^*(h_2) = 1,641$ nodes
Heuristic Functions

- Heuristics are often obtained from relaxed problem
  - Simplify the original problem by removing constraints
  - The cost of an optimal solution to a relaxed problem is an admissible heuristic.
8-Puzzle

- Original
  - A tile can move from $A$ to $B$ if $A$ is horizontally or vertically **adjacent** to $B$ and $B$ is **blank**.

- Relaxations
  - Move from $A$ to $B$ if $A$ is adjacent to $B$ (remove “blank”)
    - $h_2$ by moving each tile in turn to destination
  - Move from $A$ to $B$ (remove “adjacent” and “blank”)
    - $h_1$ by simply moving each tile directly to destination
How to Obtain Heuristics?

- Ask the domain expert (if there is one)
- Solve example problems and generalize your experience on which operators are helpful in which situation (particularly important for state space search)
- Try to develop sophisticated evaluation functions that measure the closeness of a state to a goal state (particularly important for state space search)
- Run your search algorithm with different parameter settings trying to determine which parameter settings of the chosen search algorithm are “good” to solve a particular class of problems.
- Write a program that selects “good parameter” settings based on problem characteristics (frequently very difficult) relying on machine learning
A* Search

- The greedy best-first search does not consider how costly it was to get to a node.
  - $f(n) = h(n)$
- Idea: avoid expanding paths that are already expensive
- Combine $g(n)$, the cost to reach node $n$, with $h(n)$
  - $f(n) = g(n) + h(n)$
  - estimated cost of cheapest solution through $n$
A* Search

- When $h(n) = \text{actual cost to goal}$
  - Only nodes in the correct path are expanded
  - Optimal solution is found

- When $h(n) < \text{actual cost to goal}$
  - Additional nodes are expanded
  - Optimal solution is found

- When $h(n) > \text{actual cost to goal}$
  - Optimal solution can be overlooked
A* Search

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

- **Time**
  - Exponential in \([\text{relative error of } h \times \text{length of soln}]\)
  - The better the heuristic, the better the time
    - Best case \( h \) is perfect, \( O(d) \)
    - Worst case \( h = 0 \), \( O(b^d) \) same as BFS

- **Space**
  - Keeps all nodes in memory and save in case of repetition
  - This is \( O(b^d) \) or worse
  - A* usually runs out of space before it runs out of time

- **Optimal**
  - Yes, cannot expand \( f_{i+1} \) unless \( f_i \) is finished
Route Finding

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: 241
- Neamt: 234
- Oradea: 380
- Pitesti: 98
- Rimnicu Vilcea: 193
- Sibiu: 253
- Timisoara: 329
- Urziceni: 80
- Vaslui: 199
- Zerind: 374
A* Example

<table>
<thead>
<tr>
<th>Straight-line distance to Bucharest</th>
<th></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>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>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 Vilea</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>
A* Search

Arad

f(n) = 0 + 366

Sibiu

393

= 140 + 253

Timisoara

447

Zerind

449

Arad

646

Fagaras

415

Oradea

671

Rimnicu Vilcea

413

Things are different now!
A* Search Continued
Local Search / Optimization

- Idea is to find the best state.
- We don’t really care how to get to the best state, just that we get there.
- The best state is defined according to an objective function
  - Measures the “fitness” of a state.

Problem: Find the optimal state
  - The one that maximizes (or minimizes) the objective function.
State Space Landscapes

Objective Function

- global max
- local max
- shoulder

State Space
Problem Formulation

- Complete-state formulation
  - Start with an approximate solution and perturb
- n-queens problem
  - Place n queens on a board so that no queen is attacking another queen.
Problem Formulation

- Initial State: n queens placed randomly on the board, one per column.
- Successor function: States that obtained by moving one queen to a new location in its column.
- Heuristic/objective function: The number of pairs of attacking queens.
Local Search Algorithms

- Hill climbing
- Simulated annealing
- Local beam search
- Genetic Algorithms
Hill Climbing (or Descent)

Objective Function

State Space
Hill Climbing Problems

“like climbing Everest in fog with amnesia”
n-Queens

What happens if we move 3rd queen?
Possible Improvements

- Stochastic hill climbing
  - Choose at random from uphill moves
  - Probability of move could be influenced by steepness

- First-choice hill climbing
  - Generate successors at random until one is better than current.

- Random-restart
  - Execute hill climbing several times, choose best result.
  - If \( p \) is probability of a search succeeding, then expected number of restarts is \( 1/p \).
Simulated Annealing

- Similar to stochastic hill climbing
  - Moves are selected at random
  - If a move is an improvement, accept
  - Otherwise, accept with probability less than 1.

- Probability gets smaller as time passes and by the amount of “badness” of the move.