Search Gone Wrong?
TSP

http://idle.slashdot.org/story/
12/04/25/1519208/travelling-salesman-thriller-set-in-a-world-where-pnp
Depth First Search

Strategy: expand deepest node first

Implementation: Probably want to use recursion
Breadth First Search

Strategy: expand shallowest node first

Implementation: Fringe is a FIFO queue
In this problem the start state is $S$, and the goal state is $G$. The transition costs are next to the edges. Ignore “h”. Assume that ordering is defined by, and ties are always broken by, choosing the state that comes first alphabetically.
Search Algorithm Properties

- **Complete?** Guaranteed to find a solution if one exists?
- **Optimal?** Guaranteed to find the least cost path?
- **Time complexity?**
- **Space complexity?**

**Variables:**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Number of states in the problem</td>
</tr>
<tr>
<td>$b$</td>
<td>The average branching factor $B$ (the average number of successors)</td>
</tr>
<tr>
<td>$C^*$</td>
<td>Cost of least cost solution</td>
</tr>
<tr>
<td>$s$</td>
<td>Depth of the shallowest solution</td>
</tr>
<tr>
<td>$m$</td>
<td>Max depth of the search tree</td>
</tr>
</tbody>
</table>
Infinite paths make DFS incomplete...

- How can we fix this?
With cycle checking, DFS is complete.*

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS w/ Path Checking</td>
<td>Y</td>
<td>N</td>
<td>O(b^{m+1})</td>
<td>O(b^m)</td>
</tr>
</tbody>
</table>

* Or graph search
When is BFS optimal?

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS w/ Path Checking</td>
<td>Y</td>
<td>N</td>
<td>$O(b^{m+1})$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>N*</td>
<td>$O(b^{s+1})$</td>
<td>$O(b^s)$</td>
</tr>
</tbody>
</table>
Search tree:
• Sequentially place queens
• Goal condition: no queen can take another
Comparisons

• When will BFS outperform DFS?

• When will DFS outperform BFS?
One solution to the eight queens puzzle

24683175
Genetic Algorithms

1. Generate population
2. Evaluate solutions
   - Fitness function defines “goodness”
3. Evolve new population
   - Direct selection
   - Crossover
   - Mutation
4. Goto line 2
• Genetic algorithms use a natural selection metaphor
• Probably the most misunderstood, misapplied (and even maligned) technique around!
Example: N-Queens

- Why does crossover make sense here?
- When wouldn’t it make sense?
- What would mutation be?
- What would a good fitness function be?
• http://electricsheep.org/
NEAT
[Stanley & Miikkulainen, ‘02]

- **NeuroEvolution of Augmenting Topologies:**
  - Evolves neural networks with a genetic algorithm
- Evaluate each policy in a population
- Learns structure and weights via:
  - Mutation
  - Genetic crossover
  - Adding nodes
  - Adding links
Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less.
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.
   ....and so on.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>w/ Path Checking</td>
<td>Y</td>
<td>N</td>
<td>O(b^{m+1})</td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td>Y</td>
<td>N*</td>
<td>O(b^{s+1})</td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td>Y</td>
<td>N*</td>
<td>O(b^{s+1})</td>
</tr>
</tbody>
</table>
Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path. We will quickly cover an algorithm which does find the least-cost path.
Uniform Cost Search

Expand cheapest node first:
Fringe is a priority queue
Priority Queue Refresher

- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pq.push(key, value)</td>
<td>inserts (key, value) into the queue.</td>
</tr>
<tr>
<td>pq.pop()</td>
<td>returns the key with the lowest value, and removes it from the queue.</td>
</tr>
</tbody>
</table>

- You can decrease a key’s priority by pushing it again
- Unlike a regular queue, insertions aren’t constant time, usually $O(\log n)$
- We’ll need priority queues for cost-sensitive search methods
# Uniform Cost Search

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td><strong>Y</strong></td>
<td><strong>N</strong></td>
<td>(O(b^{m+1}))</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td>BFS</td>
<td><strong>Y</strong></td>
<td><strong>N</strong></td>
<td>(O(b^{s+1}))</td>
<td>(O(b^s))</td>
</tr>
<tr>
<td>UCS</td>
<td><strong>Y</strong></td>
<td><strong>Y</strong></td>
<td>(O(b^{c*/\varepsilon}))</td>
<td>(O(b^{c*/\varepsilon}))</td>
</tr>
</tbody>
</table>

* UCS can fail if actions can get arbitrarily cheap
Done with Search?

UCS!
Uniform Cost Issues

• Remember: explores increasing cost contours

• The good: UCS is complete and optimal!

• The bad:
  – Explores options in every “direction”
  – No information about goal location