Analyzing and Improving Search

1/31/18
From Wednesday: Measuring Performance

- **Completeness**: Is the search guaranteed to find a solution (if one exists)?
- **Optimality**: Is the search guaranteed to find the lowest-cost solution (if it finds one)?
- **Time complexity**: How long does it take to find a solution?
  - How many nodes are expanded?
- **Space complexity**: How much memory is needed to perform the search?
  - How many nodes get stored in frontier + visited
Exercise: fill in the table

<table>
<thead>
<tr>
<th></th>
<th>BFS</th>
<th>DFS</th>
<th>LCFS</th>
<th>A*</th>
<th>Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>optimal?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time efficient?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space efficient?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BFS</td>
<td>DFS</td>
<td>UCS</td>
<td>A*</td>
<td>Greedy</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>-----------</td>
<td>------</td>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td><strong>complete?</strong></td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><strong>optimal?</strong></td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><strong>time efficient?</strong></td>
<td>no</td>
<td>occasionally</td>
<td>no</td>
<td>sort of</td>
<td>often</td>
</tr>
<tr>
<td><strong>space efficient?</strong></td>
<td>no</td>
<td>yes!!!</td>
<td>no</td>
<td>no</td>
<td>sometimes</td>
</tr>
</tbody>
</table>
From Wednesday: Devising Heuristics

• Must be **admissible**: never overestimate the cost to reach the goal.

• Should strive for **consistency**: $h(s) + c(s)$ non-decreasing along paths.

• The higher the estimate (subject to admissibility), the better.

Key idea: simplify the problem.

• Traffic Jam: ignore some of the cars.

• Path Finding: assume straight roads.
Exercise: devise a heuristic

8-puzzle:
• Actions: a tile orthogonally adjacent to the empty space can slide into it.
• Goal: arrange the tiles in increasing order.
Why is A* complete and optimal?

• Let C* be the cost of the optimal solution path.
• A* will expand all nodes with \( c(s) + h(s) < C^* \).
• A* will expand some nodes with \( c(s) + h(s) = C^* \) until finding a goal node.
• With an admissible heuristic, A* is optimal because it can’t miss a better path.
• Given a positive step cost and a finite branching factor, A* is also complete.
Why is A* optimally efficient?

• For any given admissible heuristic, no other optimal algorithm will expand fewer nodes.

• Any algorithm that does NOT expand all nodes with $c(s) + h(s) < C^*$ runs the risk of missing the optimal solution.

• Only possible difference could be in which nodes are expanded when $c(s) + h(s) = C^*$. 
Iterative Deepening

- Inherits the completeness and shortest-path properties from BFS.
- Requires only the memory complexity of DFS.

Key idea:
- Run a depth-limited DFS.
- Increase the depth limit if goal not found.
IDA*; Branch and Bound

• Use DFS, but with a bound on $c(s) + h(s)$.
• If bound < $c(\text{goal})$, the search will fail and we’ll have to increase the bound.
  • IDA* starts with a low bound and gradually increases it.
• If bound > $c(\text{goal})$, we may find a sub-optimal solution
  • We can re-run with $c(\text{solution}) - \varepsilon$ as the new bound
  • Branch and bound starts with a high bound and lowers it each time a solution is found.
• We can alternate these two to narrow in on the right bound.
• With reasonable bounds, these will explore an asymptotically similar number of nodes to A*, with a lower memory overhead.
Bidirectional Search

• Also search from the goal(s) toward the start.

• Requires a known, small set of goals.
Island-driven search

• Identify way-points (islands) that indicate progress toward the goal.
• Search for a path to the next waypoint.
• Not optimal unless you’re sure that the waypoint is on the optimal path.